INTERNATIONAL STANDARD

ISO/IEC 14496-11

First edition
2005-12-15

Information technology — Coding of audio-visual objects —

Part 11: Scene description and application engine

Technologies de l'information — Codage des objets audiovisuels —

Partie 11: Description de scène et moteur d'application

Reference number
ISO/IEC 14496-11:2005(E)

© ISO/IEC 2005

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

PDF disclaimer

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

Adobe is a trademark of Adobe Systems Incorporated.

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

© ISO/IEC 2005

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

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

Published in Switzerland

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS

© ISO/IEC 2005 – All rights reserved

Not for Resale

Contents

Page

Foreword v

0 Introduction vii
0.1 Scene Description vii
0.2 Extensible MPEG-4 Textual Format ix
0.3 MPEG-J x

1 Scope 1

2 Normative references 1

3 Additional reference 2

4 Terms and definitions 2

5 Abbreviations and Symbols 8

6 Conventions 8

7 MPEG-4 Systems Node Semantics 9
7.1 Scene Description 9
7.2 Node Semantics 27
7.3 Informative: Differences Between MPEG-4 Scripts and ECMA Scripts 181
7.4 Informative: FlexTime behavior 182
7.5 Informative: Implementation of MaterialKey node 183
7.6 Informative: Example implementation of spatial audio processing (perceptual approach) 184
7.7 Informative: MPEG-4 Audio TTS application with Facial Animation 189
7.8 Informative: 3D Mesh Coding in BIFS scenes 190
7.9 Profiles 190
7.10 Metric information for resident fonts 216
7.11 Font metrics for SANS SERIF font (Albany) 216
7.12 Font metrics for SERIF font (Thorndale) 223
7.13 Font metrics for TYPEWRITER font (Cumberland) 229

8 BIFS 235
8.1 Introduction 235
8.2 Decoding tables, data structures and associated functions 235
8.3 Quantization 240
8.4 Compensation process 251
8.5 BIFS Configuration 252
8.6 BIFS Command Syntax 256
8.7 BIFS Scene 266
8.8 BIFS-Anim 297
8.9 Interpolator compression 303
8.10 Definition of bodySceneGraph nodes 342
8.11 Adaptive Arithmetic Decoder for BIFS-Anim 350
8.12 Informative : Adaptive Arithmetic Encoder for BIFS-Anim 352
8.13 View Dependent Object Scalability 354

9 The Extensible MPEG-4 Textual Format 357
9.1 Introduction 357
9.2 XMT-A Format 357
9.3 XMT-Ω Format 410
9.4 XMT-C Modules 456
9.5 XMT Schemas 464
9.6 Informative: XMT/X3D Compatibility 464
9.7 Informative: The usage of XMT-A BitWrapper element in authoring side 465

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10 MPEG-J 478
10.1 Architecture 478
10.2 MPEG-J Session 480
10.3 Delivery of MPEG-J Data 482
10.4 MPEG-J API List 484
10.5 Informative: Starting the Java Virtual Machine 492
10.6 Informative: Examples of MPEG-J API usage 492

Annex A (normative) **Curve-based animators 502**

Annex B (normative) **Procedural textures algorithms 505**

Annex C (informative) **Text Processing in BIFS 510**

Annex D (informative) **Patent statements 512**

Bibliography 513

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Foreword

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

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

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

ISO/IEC 14496-11 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

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

— *Part 1: Systems*

— *Part 2: Visual*

— *Part 3: Audio*

— *Part 4: Conformance testing*

— *Part 5: Reference software*

— *Part 6: Delivery Multimedia Integration Framework (DMIF)*

— *Part 7: Optimized reference software for coding of audio-visual objects* [Technical Report]

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

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

— *Part 10: Advanced Video Coding*

— *Part 11: Scene description and application engine*

— *Part 12: ISO base media file format*

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

— *Part 14: MP4 file format*

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

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

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— *Part 17: Streaming text format*

— *Part 18: Font compression and streaming*

— *Part 19: Synthesized texture stream*

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

The following parts are under preparation:

— *Part 21: MPEG-J GFX*

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2005 – All rights reserved
Not for Resale

0 Introduction

0.1 Scene Description

0.1.1 Overview

ISO/IEC 14496 addresses the coding of audio-visual objects of various types: natural video and audio objects as well as textures, text, 2- and 3-dimensional graphics, and also synthetic music and sound effects. To reconstruct a multimedia scene at the terminal, it is hence not sufficient to transmit the raw audio-visual data to a receiving terminal. Additional information is needed in order to combine this audio-visual data at the terminal and construct and present to the end-user a meaningful multimedia scene. This information, called scene description, determines the placement of audio-visual objects in space and time and is transmitted together with the coded objects as illustrated in Figure 1. Note that the scene description only describes the structure of the scene. The action of assembling these objects in the same representation space is called composition. The action of transforming these audio-visual objects from a common representation space to a specific presentation device (i.e. speakers and a viewing window) is called rendering.

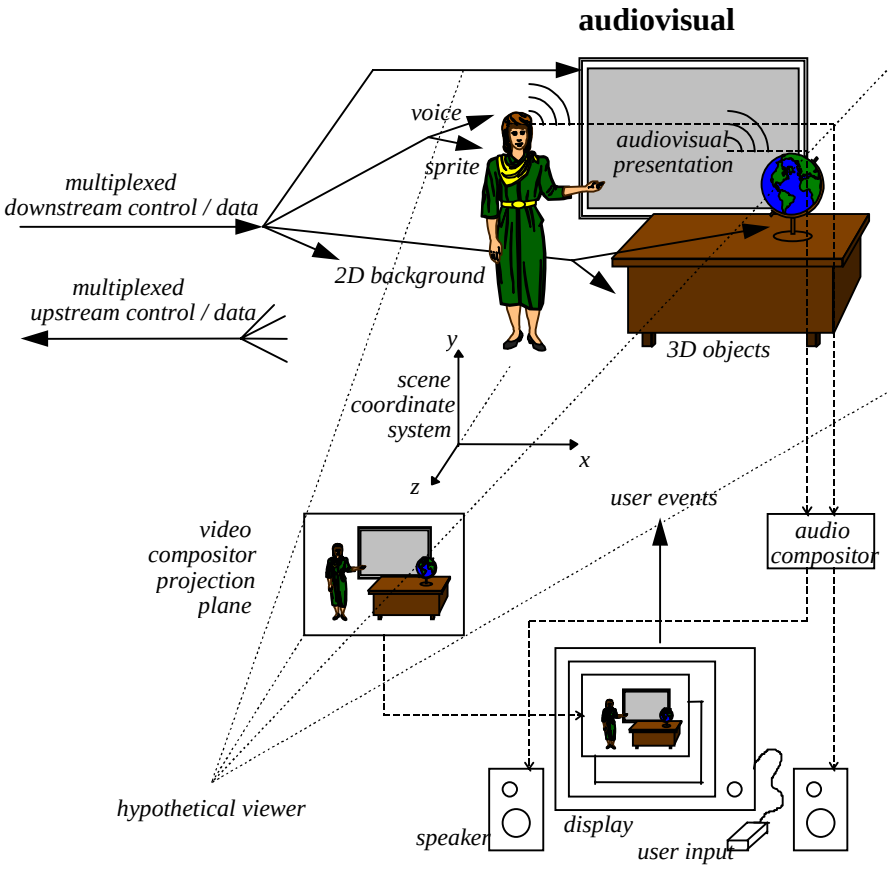


Figure 1 — An example of an object-based multimedia scene

Independent coding of different objects may achieve higher compression, and also brings the ability to manipulate content at the terminal. The behaviors of objects and their response to user inputs can thus also be represented in the scene description.

The scene description framework used in this part of ISO/IEC 14496 is based largely on ISO/IEC 14772-1:1998 (Virtual Reality Modeling Language – VRML).

0.1.2 Composition and Rendering

ISO/IEC 14496-11 defines the syntax and semantics of bitstreams that describe the spatio-temporal relationships of audio-visual objects. For visual data, particular composition algorithms are not mandated since they are implementation-

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

dependent; for audio data, subclause 7.1.1.2.13 and the semantics of the AudioBIFS nodes normatively define the composition process. The manner in which the composed scene is presented to the user is not specified for audio or visual data. The scene description representation is termed “BInary Format for Scenes” (BIFS).

0.1.3 Scene Description

In order to facilitate the development of authoring, editing and interaction tools, scene descriptions are coded independently from the audio-visual media that form part of the scene. This permits modification of the scene without having to decode or process in any way the audio-visual media. The following clauses detail the scene description capabilities that are provided by ISO/IEC 14496-11.

0.1.3.1 Grouping of audio-visual objects

A scene description follows a hierarchical structure that can be represented as a graph. Nodes of the graph form audio-visual objects, as illustrated in Figure 2. The structure is not necessarily static; nodes may be added, deleted or be modified.

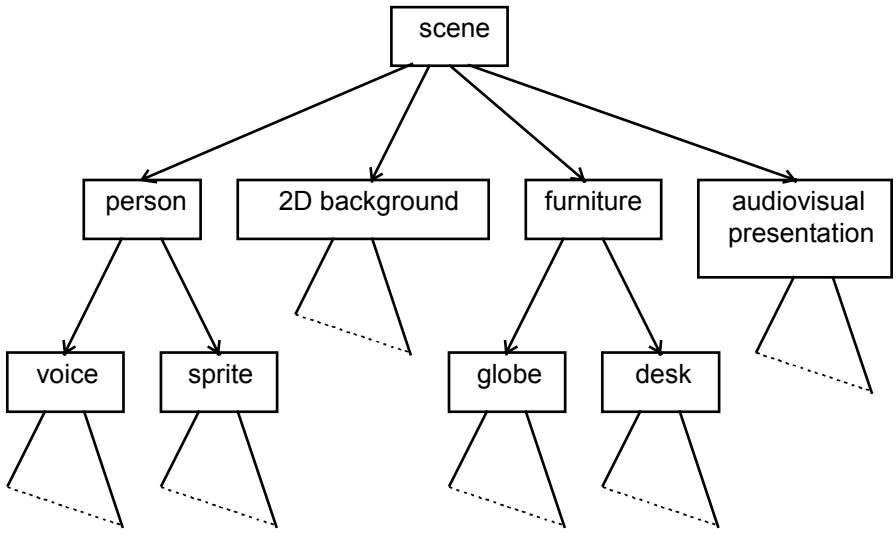


Figure 2 — Logical structure of example scene

0.1.3.2 Spatio-Temporal positioning of objects

Audio-visual objects have both a spatial and a temporal extent. Complex audio-visual objects are constructed by combining appropriate scene description nodes to build up the scene graph. Audio-visual objects may be located in 2D or 3D space. Each audio-visual object has a local co-ordinate system. A local co-ordinate system is one in which the audio-visual object has a pre-defined (but possibly varying) spatio-temporal location and scale (size and orientation). Audio-visual objects are positioned in a scene by specifying a co-ordinate transformation from the object’s local co-ordinate system into another co-ordinate system defined by a parent node in the scene graph.

0.1.3.3 Attributes of audio-visual objects

Scene description nodes expose a set of parameters through which aspects of their appearance and behavior can be controlled.

EXAMPLE — the volume of a sound; the color of a synthetic visual object; the source of a streaming video.

0.1.3.4 Behavior of audio-visual objects

ISO/IEC 14496-11 provides tools for enabling dynamic scene behavior and user interaction with the presented content. User interaction can be separated into two major categories: client-side and server-side. Client-side interaction is an integral part of the scene description described herein. Server-side interaction is not dealt with.

Client-side interaction involves content manipulation that is handled locally at the end-user’s terminal. It consists of the modification of attributes of scene objects according to specified user actions.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLE — A user can click on a scene to start an animation or video sequence. The facilities for describing such interactive behavior are part of the scene description, thus ensuring the same behavior in all terminals conforming to ISO/IEC 14496-11.

0.2 Extensible MPEG-4 Textual Format

0.2.1 Overview

The Extensible MPEG-4 Textual format (XMT) is a framework (illustrated in Figure 3) for representing MPEG-4 scene description using a textual syntax. The XMT allows the content authors to exchange their content with other authors, tools or service providers, and facilitates interoperability with both the Extensible 3D (X3D) being developed by the Web3D and the Synchronized Multimedia Integration Language (SMIL) from the W3C.

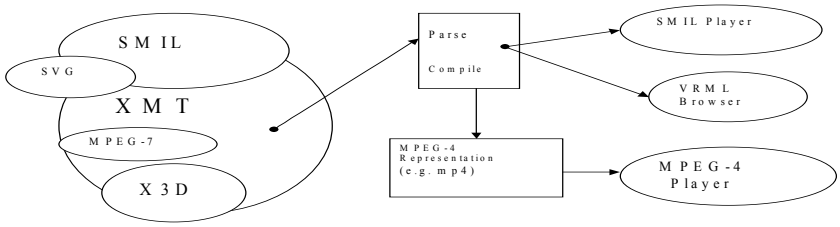


Figure 3 — Overview of the XMT Framework

0.2.2 Interoperability of XMT

The XMT format can be interchangeable between SMIL players, VRML players, and MPEG-4 players. The format can be parsed and played directly by a W3C SMIL player, preprocessed to Web3D X3D and played back by a VRML player, or compiled to an MPEG-4 representation such as MP4, which can then be played by an MPEG-4 player. See below for a graphical description of interoperability of the XMT.

0.2.3 Two-tier Architecture: XMT-A and XMT-Ω Formats

The XMT framework consists of two levels of textual syntax and semantics: the XMT-A format and the XMT-Ω format, which we will abbreviate by A and Ω, respectively, and use them interchangeably where there is no confusion.

The XMT-A is an XML-based version of MPEG-4 content, which contains a subset of the X3D. Also contained in XMT-A is an MPEG-4 extension to the X3D to represent MPEG-4 specific features. The XMT-A provides a straightforward, one-to-one mapping between the textual and binary formats.

The XMT-Ω is a high-level abstraction of MPEG-4 features designed based on the W3C SMIL. The XMT provides a default mapping from Ω to A, for there is no deterministic mapping between the two, and it also provides content authors with an escape mechanism from Ω to A.

In addition an XMT-C (Common) section contains the definition of elements and attributes that may be used within either XMT-A or XMT-Ω.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

0.3 MPEG-J

0.3.1 Overview

MPEG-J is a flexible programmatic control system that represents an audio-visual session in a manner that allows the session to adapt to the operating characteristics when presented at the terminal. Two important characteristics are supported: first, the capability to allow graceful degradation under limited or time varying resources, and second, the ability to respond to user interaction and provide enhanced multimedia functionality.

More specifically, 9.7 normatively defines:

The format and delivery of Java byte code by specifying the MPEG-J stream format and the delivery mechanism of such a stream (Java byte code and associated data);

The MPEG-J Session and the MPEG-J application lifecycle; and

The interactions and behavior of byte code through the specification of Java APIs.

0.3.2 Organization MPEG-J specification

10.1 gives an overall architecture of the MPEG-J system. MPEG-J Session start-up is walked through in 10.2. The Delivery of MPEG-J data to the terminal is specified in 10.3. 10.4 specifies the different categories of APIs that a program in the form of Java bytecode would use. 10.5 is an informative annex on starting the Java Virtual Machine. The electronic annex attached to this document lists the normative MPEG-J APIs in the HTML format. 10.6 illustrates the usage of MPEG-J APIs through a few examples.

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

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

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

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

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Information technology — Coding of audio-visual objects —

Part 11:
Scene description and application engine

1 Scope

This part of ISO/IEC 14496 specifies:

1. the coded representation of the spatio-temporal positioning of audio-visual objects as well as their behavior in response to interaction (scene description);
2. the Extensible MPEG-4 Textual (XMT) format, a textual representation of the multimedia content described in ISO/IEC 14496 using the Extensible Markup Language (XML); and
3. a system level description of an application engine (format, delivery, lifecycle, and behavior of downloadable Java byte code applications).

2 Normative references

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

ISO 639-2:1998*, Codes for the representation of names of languages — Part 2: Alpha-3 code*

ISO 3166-1:1997, *Codes for the representation of names of countries and their subdivisions — Part 1: Country codes*

ISO 9613-1:1993, *Acoustics — Attenuation of sound during propagation outdoors — Part 1: Calculation of the absorption of sound by the atmosphere*

ISO/IEC 11172-2:1993, *Information technology — Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s — Part 2: Video*

ISO/IEC 11172-3:1993, *Information technology — Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s — Part 3: Audio*

ISO/IEC 13818-3:1998, *Information technology — Generic coding of moving pictures and associated audio information — Part 3: Audio*

ISO/IEC 13818-7: 2004, *Information technology — Generic coding of moving pictures and associated audio information — Part 7: Advanced Audio Coding (AAC)*

ISO/IEC 14496-2:2004, *Information technology — Coding of audio-visual objects — Part 2: Visual*

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 14772-1:1997/Amd.1:2003, *Information technology — Computer graphics and image processing — The Virtual Reality Modeling Language — Part 1: Functional specification and UTF-8 encoding — Amendment 1: Enhanced interoperability*

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

ISO/IEC 13818-2:2000, *Information technology — Generic coding of moving pictures and associated audio information — Part 2: Video*

ISO/IEC 10918-1:1994, *Information technology — Digital compression and coding of continuous-tone still images: Requirements and guidelines*

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IEEE Std 754-1985, *Standard for Binary Floating-Point Arithmetic*

Addison-Wesley:September 1996, *The Java Language Specification,* by James Gosling, Bill Joy and Guy Steele, ISBN 0-201-63451-1

Addison-Wesley:September 1996, *The Java Virtual Machine Specification,* by T. Lindholm and F. Yellin, ISBN 0-201-63452-X

Addison-Wesley:July 1998, *Java Class Libraries Vol. 1 The Java Class Libraries,* Second Edition Volume 1, by Patrick Chan, Rosanna Lee and Douglas Kramer, ISBN 0-201-31002-3

Addison-Wesley:July 1998, *Java Class Libraries Vol. 2 The Java Class Libraries,* Second Edition Volume 2, by Patrick Chan and Rosanna Lee, ISBN 0-201-31003-1

Addison-Wesley, May 1996, *Java API, The Java Application Programming Interface, Volume 1: Core Packages,* by J. Gosling, F. Yellin and the Java Team, ISBN 0-201-63453-8

DAVIC 1.4.1 specification Part 9: Information Representation

ANSI/SMPTE 291M-1996, *Television — Ancillary Data Packet and Space Formattin*g

SMPTE 315M -1999, *Television — Camera Positioning Information Conveyed by Ancillary Data Packets*

3 Additional reference

ISO/IEC 13522-6:1998, *Information technology — Coding of multimedia and hypermedia information — Part 6: Support for enhanced interactive applications.* This reference contains the full normative references to Java APIs and the Java Virtual Machine as described in the normative references above.

4 Terms and definitions

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

4.1
Access Unit (AU)

individually accessible portion of data within an *elementary stream*

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

4.2
Alpha map

representation of the transparency parameters associated with a texture map

4.3
atom

object-oriented building block defined by a unique type identifier and length

4.4
audio-visual object

representation of a natural or synthetic object that has an audio and/or visual manifestation

NOTE The representation corresponds to a node or a group of nodes in the BIFS scene description. Each audio-visual object is associated with zero or more *elementary streams* using one or more *object descriptors.*

4.5
audio-visual scene (AV scene)

set of audio-visual objects together with scene description information that defines their spatial and temporal attributes including behaviors resulting from object and user interactions

4.6
Binary Format for Scene (BIFS)

coded representation of a parametric scene description format

4.7
buffer model

model that defines how a terminal complying with ISO/IEC 14496 manages the buffer resources that are needed to decode a presentation

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.8
byte aligned
position in a coded bit stream with a distance of a multiple of 8-bits from the first bit in the stream

4.9
chunk
contiguous set of samples stored for one stream

4.10
clock reference
special time stamp that conveys a reading of a time base

4.11
composition
process of applying scene description information in order to identify the spatio-temporal attributes and hierarchies of audio-visual objects

4.12
Composition Memory (CM)
random access memory that contains composition units

4.13
Composition Time Stamp (CTS)
indication of the nominal composition time of a composition unit

4.14
Composition Unit (CU)
individually accessible portion of the output that a decoder produces from access units

4.15
compression layer
layer of a system according to the specifications in ISO/IEC 14496 that translates between the coded representation of an elementary stream and its decoded representation

NOTE It incorporates the decoders.

4.16
container atom
atom whose sole purpose is to contain and group a set of related atoms

4.17
decoder
entity that translates between the coded representation of an elementary stream and its decoded representation

4.18
decoding buffer (DB)
buffer at the input of a decoder that contains access units

4.19
decoder configuration
configuration of a decoder for processing its elementary stream data by using information contained in its elementary stream descriptor

4.20
Decoding Time Stamp (DTS)
indication of the nominal decoding time of an access unit

4.21
delivery layer
generic abstraction for delivery mechanisms (computer networks, etc.) able to store or transmit a number of multiplexed elementary streams or M4Mux streams

4.22
descriptor
data structure that is used to describe particular aspects of an elementary stream or a coded audio-visual object

4.23
DMIF Application Interface (DAI)
interface specified in ISO/IEC 14496-6

NOTE It is used here to model the exchange of SL-packetized stream data and associated control information between the sync layer and the delivery layer.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.24
Elementary Stream (ES)

consecutive flow of mono-media data from a single source entity to a single destination entity on the compression layer

4.25
Elementary Stream Descriptor (ESD)

structure contained in object descriptors that describes the encoding format, initialization information, sync layer configuration, and other descriptive information about the content carried in an elementary stream

4.26
Elementary Stream Interface (ESI)

conceptual interface modeling the exchange of elementary stream data and associated control information between the compression layer and the sync layer

4.27
M4Mux Channel (FMC)

label to differentiate between data belonging to different constituent streams within one M4Mux Stream

NOTE A sequence of data in one M4Mux channel within a M4Mux stream corresponds to one single SL-packetized stream.

4.28
M4Mux packet

smallest data entity managed by the M4Mux tool, consisting of a header and a payload

4.29
M4Mux stream

sequence of M4Mux Packets with data from one or more SL-packetized streams that are each identified by their own M4Mux channel

4.30
M4Mux tool

tool that allows the interleaving of data from multiple data streams

4.31
graphics profile

profile that specifies the permissible set of graphical elements of the BIFS tool that may be used in a scene description stream

NOTE Note BIFS comprises both graphical and scene description elements.

4.32
hint track

special track which contains instructions for packaging one or more tracks into a TransMux

NOTE It does not contain media data (an elementary stream).

4.33
hinter

tool that is run on a completed file to add one or more hint tracks to the file to facilitate streaming

4.34
inter

mode for coding parameters that uses previously coded parameters to construct a prediction

4.35
interaction stream

elementary stream that conveys user interaction information

4.36
intra

mode for coding parameters that does not make reference to previously coded parameters to perform the encoding

4.37
initial object descriptor

special object descriptor that allows the receiving terminal to gain initial access to portions of content encoded according to ISO/IEC 14496

NOTE It conveys profile and level information to describe the complexity of the content.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.38
Intellectual Property Identification (IPI)
unique identification of one or more elementary streams corresponding to parts of one or more audio-visual objects

4.39
Intellectual Property Management and Protection System (IPMP)
generic term for mechanisms and tools to manage and protect intellectual property

NOTE Only the interface to such systems is normatively defined.

4.40
media node
following list of time dependent nodes that refers to a media stream through a URL field: **AnimationStream**, **AudioBuffer, AudioClip**, **AudioSource**, **Inline**, **MovieTexture**

4.41
media stream
one or more elementary streams whose ES descriptors are aggregated in one object descriptor and that are jointly decoded to form a representation of an AV object

4.42
media time line
time line expressing normal play back time of a media stream

4.43
movie atom
container atom whose sub-atoms define the meta-data for a presentation ('moov')

4.44
movie data atom
container atom which can hold the actual media data for a presentation ('mdat')

4.45
MP4 file
name of the file format described in this specification

4.46
Object Clock Reference (OCR)
clock reference that is used by a decoder to recover the time base of the encoder of an elementary stream

4.47
Object Content Information (OCI)
additional information about content conveyed through one or more elementary streams

NOTE It is either aggregated to individual elementary stream descriptors or is itself conveyed as an elementary stream.

4.48
Object Descriptor (OD)
descriptor that aggregates one or more elementary streams by means of their elementary stream descriptors and defines their logical dependencies

4.49
Object Descriptor Command
command that identifies the action to be taken on a list of object descriptors or object descriptor IDs, e.g. update or remove

4.50
Object Descriptor Profile
profile that specifies the configurations of the object descriptor tool and the sync layer tool that are allowed

4.51
Object Descriptor Stream
elementary stream that conveys object descriptors encapsulated in object descriptor commands

4.52
Object Time Base (OTB)
time base valid for a given elementary stream, and hence for its decoder

NOTE The OTB is conveyed to the decoder via object clock references. All time stamps relating to this object's decoding process refer to this time base.

4.53
Parametric Audio Decoder
set of tools for representing and decoding speech signals coded at bit rates between 6 Kbps and 16 Kbps, according to the specifications in ISO/IEC 14496-3

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.54
Quality of Service (QoS)
performance that an elementary stream requests from the delivery channel through which it is transported

NOTE QoS is characterized by a set of parameters (e.g. bit rate, delay jitter, bit error rate, etc.).

4.55
random access
process of beginning to read and decode a coded representation at an arbitrary point within the elementary stream

4.56
reference point
location in the data or control flow of a system that has some defined characteristics

4.57
rendering
action of transforming a scene description and its constituent audio-visual objects from a common representation space to a specific presentation device (i.e., speakers and a viewing window)

4.58
rendering area
portion of the display device's screen into which the scene description and its constituent audio-visual objects are to be rendered

4.59
sample
access unit for an elementary stream

NOTE In hint tracks, a sample defines the formation of one or more TransMux packets.

4.60
sample table
packed directory for the timing and physical layout of the samples in a track

4.61
scene description
information that describes the spatio-temporal positioning of audio-visual objects as well as their behavior resulting from object and user interactions

NOTE The scene description makes reference to elementary streams with audio-visual data by means of pointers to object descriptors.

4.62
scene description stream
elementary stream that conveys scene description information

4.63
scene graph elements
elements of the BIFS tool that relate only to the structure of the audio-visual scene (spatio-temporal positioning of audio-visual objects as well as their behavior resulting from object and user interactions) excluding the audio, visual and graphics nodes as specified in 14496-11

4.64
scene graph profile
profile that defines the permissible set of scene graph elements of the BIFS tool that may be used in a scene description stream

NOTE Note BIFS comprises both graphical and scene description elements.

4.65
seekable
media stream is seekable if it is possible to play back the stream from any position

4.66
SL-Packetized Stream (SPS)
sequence of sync layer packets that encapsulate one elementary stream

4.67
stream object
media stream or a segment thereof

NOTE A stream object is referenced through a URL field in the scene in the form "OD:n" or "OD:n#<segmentName>".

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.68
structured audio

method of describing synthetic sound effects and music as defined by ISO/IEC 14496-3

4.69
Sync Layer (SL)

layer to adapt elementary stream data for communication across the DMIF Application Interface, providing timing and synchronization information, as well as fragmentation and random access information

NOTE The sync layer syntax is configurable and can be configured to be empty.

4.70
Sync Layer Configuration

configuration of the sync layer syntax for a particular elementary stream using information contained in its elementary stream descriptor

4.71
Sync Layer Packet (SL-Packet)

smallest data entity managed by the sync layer consisting of a configurable header and a payload

NOTE The payload may consist of one complete access unit or a partial access unit.

4.72
Syntactic Description Language (SDL)

language defined by ISO/IEC 14496-1 that allows the description of a bitstream's syntax

4.73
Systems Decoder Model (SDM)

model that provides an abstract view of the behavior of a terminal compliant to ISO/IEC 14496

NOTE It consists of the buffer model and the timing model.

4.74
System Time Base (STB)

time base of the terminal

NOTE Its resolution is implementation-dependent. All operations in the terminal are performed according to this time base.

4.75
terminal

system that sends, or receives and presents the coded representation of an interactive audio-visual scene as defined by ISO/IEC 14496-11

NOTE It can be a stand alone system, or part of an application system complying with ISO/IEC 14496.

4.76
time base

notion of a clock, equivalent to a counter that is periodically incremented

4.77
timing model

model that specifies the semantic meaning of timing information, how it is incorporated (explicitly or implicitly) in the coded representation of information, and how it can be recovered at the receiving terminal

4.78
time stamp

indication of a particular time instant relative to a time base

4.79
track

collection of related samples in an MP4 file

NOTE For media data, a track corresponds to an elementary stream. For hint tracks, a track corresponds to a TransMuxchannel

4.80
interaction stream

elementary stream that conveys user interaction information

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5 Abbreviations and Symbols

AFX	Animation Framework eXtension
AU	Access Unit
AV	Audio-visual
BIFS	Binary Format for Scene
CM	Composition Memory
CTS	Composition Time Stamp
CU	Composition Unit
DAI	DMIF Application Interface (see ISO/IEC 14496-6)
DB	Decoding Buffer
DTS	Decoding Time Stamp
ES	Elementary Stream
ESI	Elementary Stream Interface
ESID	Elementary Stream Identifier
FAP	Facial Animation Parameters
FAPU	FAP Units
FDP	Facial Definition Parameters
FIG	FAP Interpolation Graph
FIT	FAP Interpolation Table
FMC	M4Mux Channel
FMOD	The floating point modulo (remainder) operator which returns the remainder of x/y such that: Fmod(x/y) = x – k*y, where k is an integer, sgn(fmod(x/y)) = sgn(x), and abs(fmod(x/y)) < abs(y)
IP	Intellectual Property
IPI	Intellectual Property Identification
IPMP	Intellectual Property Management and Protection
NCT	Node Coding Tables
NDT	Node Data Type
NINT	Nearest INTeger value
OCI	Object Content Information
OCR	Object Clock Reference
OD	Object Descriptor
ODID	Object Descriptor Identifier
OTB	Object Time Base
PLL	Phase Locked Loop
QoS	Quality of Service
SAOL	Structured Audio Orchestra Language
SASL	Structured Audio Score Language
SDL	Syntactic Description Language
SDM	Systems Decoder Model
SL	Synchronization Layer
SL-Packet	Synchronization Layer Packet
SPS	SL-Packetized Stream
STB	System Time Base
TTS	Text-To-Speech
URL	Universal Resource Locator
VOP	Video Object Plane
VRML	Virtual Reality Modeling Language

6 Conventions

For the purpose of unambiguously defining the syntax of the various bitstream components defined by the normative parts of ISO/IEC 14496 a *syntactic description language* is used. This language allows the specification of the mapping of the various parameters in a binary format as well as how they are placed in a serialized bitstream. The definition of the language is provided in 14496-1.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7 MPEG-4 Systems Node Semantics

7.1 Scene Description

7.1.1 Concepts

7.1.1.1 BIFS Elementary Streams

7.1.1.1.1 Overview

BIFS is a compact binary format representing a pre-defined set of audio-visual objects, their behaviors, and their spatio-temporal relationships. The BIFS scene description may, in general, be time-varying. Consequently, BIFS data is carried in a dedicated elementary stream and is subject to the provisions of the systems decoder model (see 8.2). Portions of BIFS data that become valid at a given point in time are contained in BIFS `CommandFrames` or `AnimationFrames` and are delivered within time-stamped access units. Note that the initial BIFS scene is sent as a BIFS-Command, although it is not required, in general, that a BIFS `CommandFrame` contains a complete BIFS scene description.

7.1.1.1.2 BIFS Decoder Configuration

BIFS configuration information is contained in a `BIFSConfig` (see 8.5.2) syntax structure, which is transmitted as `DecoderSpecificInfo` for the BIFS elementary stream in the corresponding object descriptor (see 7.2.6.7, ISO/IEC 14496-1). This gives basic information that must be known by the terminal in order to parse the BIFS elementary stream. In particular, it indicates whether the stream consists of BIFS-Command or BIFS-Anim entities.

7.1.1.1.3 BIFS Access Units

A BIFS data access unit consists of one BIFS `CommandFrame` or `AnimationFrame`, as defined in 8.6.2 and 8.8.2 respectively. The BIFS `CommandFrame` or `AnimationFrame` shall convey all the data that is to be processed at any given instant in time. Access units in BIFS streams shall be labelled and time-stamped by suitable means. This shall be done via the related flags and the composition time stamps (CTS), respectively, in the SL packet header (see 7.3.2.4, ISO/IEC 14496-1). The composition time indicates the point in time at which the `CommandFrame` or `AnimationFrame` embedded in a BIFS access unit shall become valid. This means that any changes to audio-visual objects that are described in the BIFS access unit will become visible or audible at precisely this time in an ideal compositor, unless a different behavior is specified by the fields of their nodes. Decoding and composition time for a BIFS access unit shall always have the same value.

An access unit does not necessarily convey a complete scene. In that case it just modifies the persistent state of the scene description. However, if an access unit conveys a complete scene as required at a given point in time it shall set the `randomAccessPointFlag` in the SL packet header to '1' for this access unit. Otherwise, the `randomAccessPointFlag` shall be set to '0'.

7.1.1.1.4 Time base for BIFS streams

The time base associated to a BIFS stream shall be indicated by suitable means. This shall be done by means of object clock reference time stamps in the SL packet headers (see 7.3.2.4, ISO/IEC 14496-1) for this stream or by indicating the elementary stream from which this BIFS stream inherits the time base (see 7.3.2.3, ISO/IEC 14496-1). All time stamps in the SL-packetized BIFS stream refer to this time base.

7.1.1.1.5 Multiple BIFS streams

Scene description data may be conveyed in more than one BIFS elementary streams. Two distinct mechanisms exist to associate a set of BIFS elementary streams to a single scene.

The first method uses **Inline** nodes (see 7.2.2.70) in a BIFS scene description. Each such node refers to further BIFS elementary streams. In this case, multiple BIFS streams have a hierarchical dependency. Each **Inline** node opens a new name scope for the identifiers used to label BIFS elements (`nodeID`, `ROUTEID`, `objectDescriptorID`). Therefore, it is not possible to pass events between parts of a scene that reside below different **Inline** nodes.

EXAMPLE 1 — An application of hierarchical BIFS streams is a multi-user virtual conferencing scene, where sub-scenes originate from different sources. Usually, it is neither possible nor useful to specify interaction between two such disjoint parts of the scene.

The second method to associate multiple BIFS elementary streams to a single scene is to group their elementary stream descriptors in a single object descriptor (see 7.2.7.2.2, ISO/IEC 14496-1). In this case, these BIFS streams share the

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

same scope for the identifiers they use (`nodeID`, `ROUTEID`, `objectDescriptorID`). This allows a single scene to be partitioned into multiple streams.

EXAMPLE 2 — An application may offer a presentation with different levels of detail, corresponding to different data rates and different computational complexity. By sharing the same name scope, the more detailed scene description can build on the simple one, rather than sending the entire scene again.

7.1.1.1.6 Time

7.1.1.1.6.1 Stream Objects

A Media Stream consists of one or more elementary streams whose ES descriptors are aggregated in one object descriptor and that are jointly decoded to form a representation of an AV object. Such streams may be streamed in response to player requests, in particular in the case of Media nodes that control play back of media. Streams may be seekable, in which case the stream can be played from any (randomly accessible) time position in the stream, or they may be non-seekable, in which case the player has no control over the playback of the stream, as is the case in broadcast scenarios.

7.1.1.1.6.2 Time-dependent Media Nodes

This specification defines the notion of a Media Node. Such nodes control the opening and playback of remote streams and are time-dependent nodes.

The **url** field of a media node shall contain at most one element which must point to a complete media stream, i.e. it is of the form “OD:n”. Media Nodes may become active or inactive based on the value of their **startTime** and **stopTime** fields. The mediaTime of the played stream is controlled by a MediaControl node, and is not dependent on the **startTime** and **stopTime** in the media Node.

The semantics of the **loop**, **startTime** and **stopTime** exposedFields and the **isActive** eventOut in time-dependent nodes are as described in ISO/IEC 14772-1:1998, Subclause 4.6.9. **startTime**, **stopTime** and **loop** apply only to the local start, pause and restart of these nodes. In the case of media Nodes, these fields affect the delivery of the stream attached to media nodes as described below. The following media nodes exist: **AnimationStream**, **AudioBuffer, AudioClip**, **AudioSource**, **MovieTexture.**

When a media node becomes active and the stream associate with that media node is already active, the media node simply joins the session. If the stream is not active when the media node becomes active, the stream becomes active; i.e. it is played.

When a media node becomes inactive, the stream shall become inactive if there are no other active media nodes referring to that stream, otherwise the stream remains active.

For media nodes that reference streams that are synchronized with the scene, reaching **startTime** and **stopTime** shall not cause the streams to become active or inactive. Instead, the media shall be muted (i.e. hidden) and unmated, respectively.

Loop and **speed** in a **MediaControl** node shall over-ride the same fields, when they exist, in any media node referencing the controlled stream. These fields retain their semantics when no controlling **MediaControl** node is present in the scene.

7.1.1.1.6.3 Time fields in BIFS nodes

SFTime field in BIFS nodes define either a time duration or a point in time.

Durations expressed by relative SFTime values like the **cycleTime** field of the **TimeSensor** node are determined unambiguously by the time base of the BIFS stream as defined in 7.1.1.1.4. Note that the time base of a stream (and hence the scene time of a scene sub-tree) can be modified by the **TemporalTransform** node which is used to synchronize different streams.

The **startTime** and **stopTime** SFTime fields in BIFS Media nodes and the **TimeSensor** node represent an absolute position on the time line of the BIFS stream. This absolute position is defined as follows:

Each node in the scene description has an associated point in time at which it is inserted in the scene graph or at which an absolute-position SFTime field in such a node is updated through a `CommandFrame` in a BIFS access unit (see 7.1.1.1.3). The value of such an SFTime field in the scene graph shall be calculated as the sum of the value encoded in the BIFS command (or the default value, if no value is encoded in the BIFS command) and the scene time as evaluated in the scene graph (e.g. the value of the **time** field of a **TimeSensor**) when the command is composed.

Offset to **startTime** / **stopTime** fields shall be applied in the following cases:

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- insertion of the node with explicit or default value for the **startTime** / **stopTime** fields;
- replacement of the value of the **startTime** / **stopTime** field through a BIFS Command ;
- replacement of the PROTO interface field to which the **startTime** / **stopTime** field is Ised;
- insertion of a PROTO instance whose body contains **startTime** / **stopTime** fields.

No offset shall be applied to a st**a**rtTime / **stopTime** field in the following cases:

- modification through a ROUTE;
- replacement of the value of the **startTime** / **stopTime** field through a Script or MPEG-J.

NOTE 1 — Absolute time in ISO/IEC 14772-1:1998 is defined slightly differently. Due to the non-streamed nature of the scene description in that case, absolute time corresponds to wallclock time in ISO/IEC 14772-1.

EXAMPLE — The example in Figure 4 shows a BIFS access unit that is to become valid at CTS. It conveys a node that has an associated media elementary stream. The startTime of this node is set to a positive value Δt. Hence, startTime will occur Δt seconds after the CTS of the BIFS access unit that has incorporated this node (or the value of the startTime field) in the scene graph.

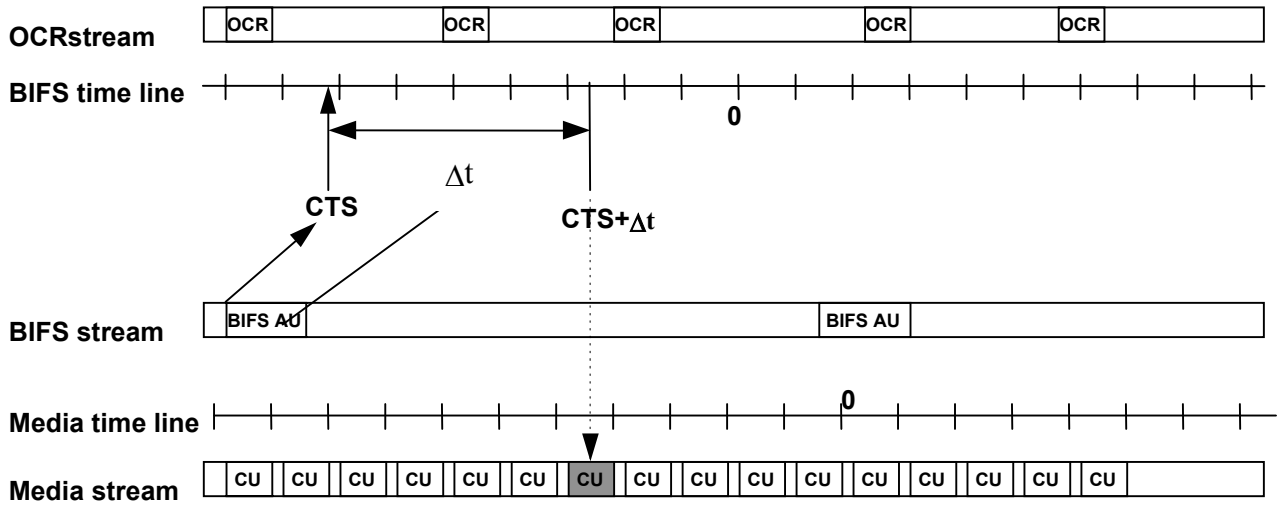


Figure 4 — Media start times and CTS

7.1.1.1.7 Sensor Reaction to Scene Changes

When the whichChoice field of a Switch node is changed or when a BIFS-command is executed, the current pointing device position shall be reevaluated with respect to all sensors in the scene.

7.1.1.2 BIFS Scene Graph

7.1.1.2.1 Structure of the BIFS scene graph

Conceptually, BIFS scenes represent (as in ISO/IEC 14772-1:1998) a set of visual and audio primitives distributed in a directed acyclic graph, in a 3D space. However, BIFS scenes may fall into several sub-categories representing particular cases of this conceptual model. In particular, BIFS scene descriptions support scenes composed of:

2D primitives (only)

3D primitives (only)

A combination of 2D and 3D primitives

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Audio primitives (only)

In scenes combining 2D and 3D primitives, the following possibilities exist:

Complete 2D and 3D scenes layered in a 2D space with depth

2D and 3D scenes used as texture maps for 2D or 3D primitives

2D scenes drawn in the local X-Y plane of the local co-ordinate system in a 3D scene

Figure 5 describes a typical BIFS scene structure.

A BIFS scene shall start with a one of the following nodes: **OrderedGroup**, **Group**, **Layer2D**, **Layer3D**. When the profile used enables visual elements to be composed, the first node indicates the co-ordinate system and context (2D or 3D) to be used for the children of that node. The following rules apply:

- Scene starts with a **Layer2D** or **OrderedGroup** node: A 2D co-ordinate system and context is assumed.
- Scene starts with a **Layer3D** or **Group** node : A 3D co-ordinate system and context is assumed.

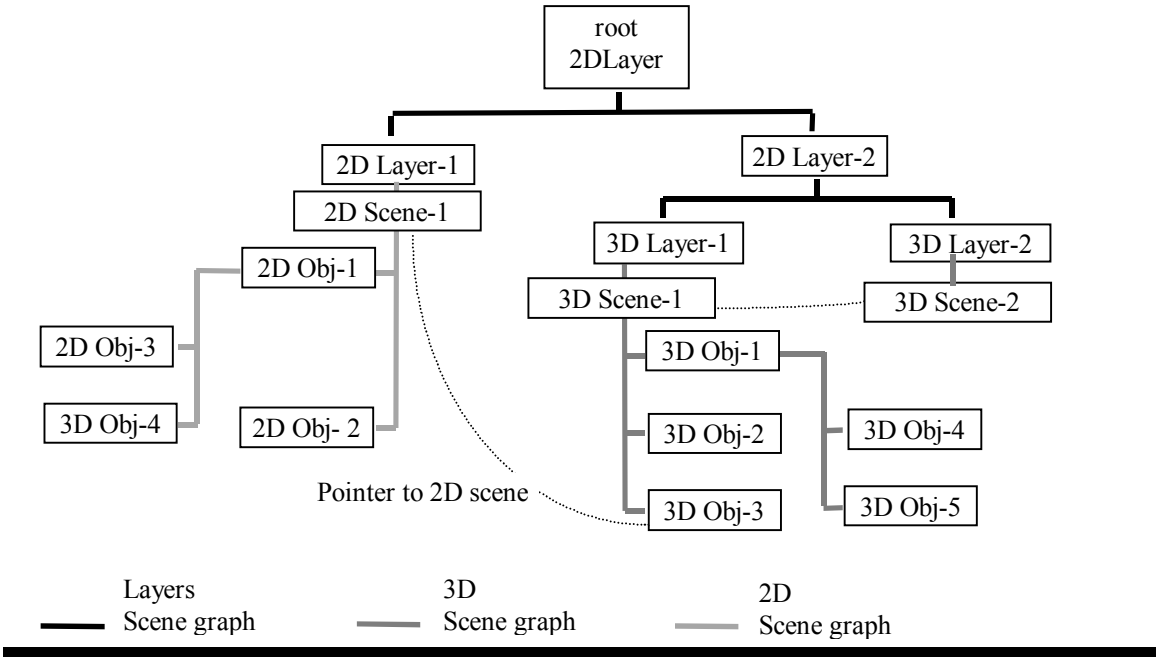


Figure 5 — Scene graph example.

The hierarchy of three different scene graphs is shown: a 2D graphics scene graph and two 3D graphics scene graphs combined with the 2D scene via layer nodes. As shown in the picture, the 3D Layer-2 is the same scene as 3D Layer-1, but the viewpoint may be different. The 3D Obj-3 is an Appearance node that uses the 2D Scene-1 as a texture node.

7.1.1.2.2 2D Co-ordinate System

The origin of the 2D co-ordinate system is positioned in the center of the rendering area, the x-axis is positive to the right, and the y-axis is positive upwards.

The width of the rendering area represents -1.0 to +1.0 (meters) on the x-axis (see Figure 6). The extent of the y-axis in the positive and negative directions is determined by the aspect ratio of the rendering area so that the unit of distance is equal in both directions. The rendering area is either the entire screen, or window on a computer screen, when viewing a single 2D scene, or the rectangular area defined by the texture used in a **CompositeTexture2D** node, or a **Layer2D** node that contains a subordinate 2D scene description.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

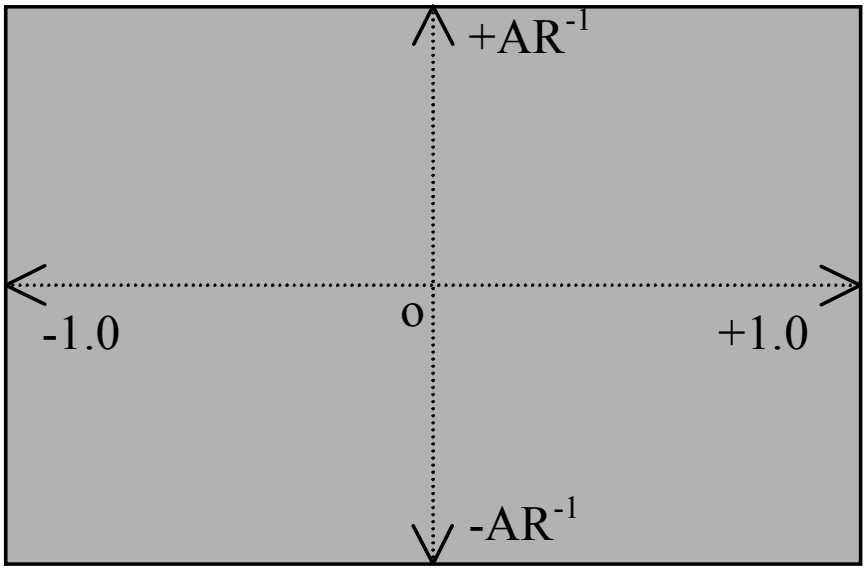


Figure 6 — 2D co-ordinate system (AR = Aspect Ratio)

7.1.1.2.3 3D Co-ordinate System

The 3D co-ordinate system is as described in ISO/IEC 14772-1:1998, subclause 4.4.5. When 2D objects are described in a 3D space, they are drawn in the local (x,y) plane (z=0), and the units used are those of the 3D co-ordinate system for the x and y directions.

7.1.1.2.4 Mixing 2D and 3D scenes

A single BIFS scene may contain both 2D and 3D elements. The following methods exist:

2D primitives may be placed in a 3D scene graph. In this cased, the 2D primitives are drawn in the local (x,y) plane, and use the local coordinate system, restricted to this (x,y) plane.

2D and 3D scenes may be composed and overlapped on the screen using **Layer2D** and **Layer3D** nodes. This is useful, for instance, when it is desirable to have 2D interfaces to 3D worlds ("head up" display), or a 3D insert in a 2D scene.

2D and 3D scenes may be mapped onto any given geometry using the **CompositeTexture2D** and **CompositeTexture3D** nodes. For instance, 2D scenes may be mapped onto animated 3D geometry to perform special effects.

7.1.1.2.5 Drawing Order

It is possible to specify the drawing order of elements of the scene, using the **OrderedGroup** node. This feature may be used for 2D or 3D scenes. 2D scenes are considered to have zero depth. Nonetheless, it is important to be able to specify the order in which 2D objects are composed, in order to describe their apparent depths. 3D scenes may use the drawing order facility to solve conflicts of coplanar polygons or other rendering optimizations.

The following rules determine the drawing order, including conflict resolution for objects having the same drawing order:

1. The object having the lowest drawing order shall be drawn first (taking into account negative values).

2. Objects having the same drawing order shall be drawn in the order in which they appear in the scene description.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.1.2.6 Pixel and Meter metrics

In addition to meter-based metrics, it is also possible to use pixel-based metrics. In this case, 1 meter is set to be equal to the distance between two pixels. This applies to both the horizontal (x-axis) and vertical (y-axis) directions.

The selection of the appropriate metrics is performed by the content creator. In particular, it is controlled by the `BIFSConfig` syntax (see 8.5.2).

When `pixelMetric` is set to 1, pixel metrics shall be used for the entire scene. This implies that rendered node sizes (such as for a Rectangle) and rendered node positions are integers. If non-integer values appear due to for example scaling, rounding shall be implied towards -infinity.

7.1.1.2.7 Nodes and fields

7.1.1.2.7.1 Nodes

The BIFS scene description consists of a collection of nodes that describe the scene structure. An audio-visual object in the scene is described by one or more nodes, which may be grouped together (using a grouping node). Nodes are grouped into node data types (NDTs) and the exact type of the node is specified using a `nodeType` field.

An audio-visual object may be completely described within the BIFS information, e.g. **Box** with **Appearance**, or may also require elementary stream data from one or more audio-visual objects, e.g. **MovieTexture** or **AudioSource**. In the latter case, the node includes a reference to an object descriptor that indicates which elementary stream(s) is (are) associated with the node, or directly to a URL description (see ISO/IEC 14772-1:1998, subclause 4.5.2). With the exception of the **Anchor** and **Script** nodes, a **url** field may only refer to content that conforms to a valid profile and level for the terminal. When a **url** field is not an OD ID url, the behaviour of the terminal is unspecified.

In the case of **InputSensor**, the node includes a reference to an object descriptor that indicates which user interaction stream is associated with the node.

7.1.1.2.7.2 Fields and Events

See ISO/IEC 14772-1:1998, subclause 5.1.

7.1.1.2.7.3 Object descriptor references in URL fields

The **url** fields in several nodes contain references to media streams. Depending on the profile and level settings (see subclause 7.9), references to media streams are made through object descriptor Ids. The textual syntax for the **url** fields in this case is as follows:

“od:<number>” - refers to the object descriptor with the id <number>.

“od:<number>#<segmentName>” - refers to the stream object defined within the object descriptor with the id <number> that has the name <segmentName>.

“od:<number>#<segmentName1>:<segmentName2>” - refers to all stream objects defined within the object descriptor with the id <number> that start at the same time or later as <segmentName1> and that and at the same time or earlier than <segmentName2>

“od:<number>#<segmentName1>+” - refers to all stream objects defined within the object descriptor with the id <number> start start at the same time or later as <segmentName1> until the end of the media stream.

7.1.1.2.7.4 Routing of Nodes

If a node is routed the target node of this route becomes as if it were a USE of the source. If the source is a USE itself, then it becomes as if it were a USE of the original DEF. If the source node is not an existing DEF or USE then it becomes an implicit DEF with the target node on the route becoming a USE of it. This behavior for node routing is thus copy by reference, rather than copy by value for the basic fields. An implementation shall behave as if it holds a reference count for nodes that are DEF’d, whether implicit or explicit. The reference count will be the number of nodes that use the definition (the count starting at one for the original DEF). Any nodes overwritten by the routing will have their reference count decremented. When the node is no longer required, i.e. the reference count has been reduced to zero, the node is deleted and is no longer available for re-use.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.1.2.8 Internal, ASCII and Binary Representation of Scenes

ISO/IEC 14496-1 describes the attributes of audio-visual objects using node structures and fields. These fields can be one of several types (see 7.1.1.2.7.2). To facilitate animation of the content and modification of the objects' attributes in time, within the terminal, it is necessary to use an internal representation of nodes and fields as described in the node specifications (see 7.2). This is essential to ensure deterministic behaviour in the terminal's compositor, for instance when applying ROUTEs or differentially coded BIFS-Anim frames. The observable behaviour of compliant terminals shall not be affected by the way in which they internally represent and transform data; that is, they shall behave as if their internal representation is as defined herein.

However, when encoding the BIFS scene description, different attributes may need to be quantized or compressed appropriately. Thus, the binary representation of fields may differ according to the types of fields, or according to the precision needed to represent a given audio-visual object's attributes. The semantics of nodes are described in subclause 7.2. The binary syntax which represents the binary format as transported in streams conforming to ISO/IEC 14496-11 is provided in clause 8 and uses the node coding parameters provided as an electronic attachment.

7.1.1.2.8.1 Binary Syntax Overview

7.1.1.2.8.1.1 Scene Description

The entire scene is represented by a binary encoding of the scene graph. This encoding restricts the VRML grammar as defined in ISO/IEC 14772-1:1997, Annex A, but still enables the representation of any scene that can be generated by this grammar.

EXAMPLE — One example of the grammatical differences is the fact that all ROUTEs are represented at the end of a BIFS scene, and that a global grouping node is required at the top level of the scene.

7.1.1.2.8.1.2 Node Description

Node types are encoded according to the context of the node. This improves efficiency by exploiting the fact that not all nodes are valid at all places in the scene graph. In many instances, only one of a subset of all BIFS nodes is valid at a particular place in the scene graph, and hence in the bitstream.

7.1.1.2.8.1.3 Fields description

Fields may be quantized to improve compression efficiency. Several aspects of the inverse quantization process can be controlled by adjusting the parameters of the **QuantizationParameter** node.

7.1.1.1.1.1 ROUTE description

All ROUTEs are described at the end of the scene. This improves bit efficiency by grouping these elements in a single location in the bitstream and removes the need for switches in the syntax to allow ROUTEs and nodes to be described in a mixed format.

7.1.1.2.9 Basic Data Types

There are two general classes of fields and events: fields/events that contain a single value (e.g. a single number or a vector), and fields/events that contain multiple values. Multiple-valued fields/events have names that begin with MF, whereas single valued begin with SF.

7.1.1.2.9.1 Numerical data and string data types

7.1.1.2.9.1.1 Introduction

For each basic data type, single field and multiple field data types are defined in ISO/IEC 14772-1:1998, subclause 5.2. Some further restrictions are described herein.

7.1.1.2.9.1.2 SFInt32/MFInt32

When routing values between two SFInt32s note shall be taken of the valid range of the destination. If the value being conveyed is outside the valid range, it shall be clipped to be equal to either the maximum or minimum value of the valid range, as follows:

if x > max, x := max

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

if x < min, x := min

7.1.1.2.9.1.3 SFTime

The SFTime field and event specifies a single time value. Time values shall consist of 64-bit floating point numbers indicating a duration in seconds or the number of seconds elapsed since the origin of time as defined in the semantics for each SFTime field.

7.1.1.2.9.2 Node data types

Nodes in the scene are also represented by a data type, namely SFNode and MFNode types. ISO/IEC 14496-1 also defines a set of sub-types, such as SFColorNode, SFMaterialNode. These node data types (NDTs) allow efficient binary representation of BIFS scenes, taking into account the usage context to achieve better compression. However, the generic SFNode and MFNode types are sufficient for internal representations of BIFS scenes.

7.1.1.2.10 Attaching nodeIDs to nodes

Each node in a BIFS scene graph may have a `nodeID` associated with it, to be used for referencing. ISO/IEC 14772-1:1998, subclause 4.6.2, describes the DEF statement which is used to attach names to nodes. In BIFS scenes, an integer value is used for the same purpose for `nodeIDs`. The number of bits used to represent these integer values is specified in the `BIFSConfig` syntax (see 8.5.2).

The following restrictions apply:

a) Nodes are identified by the use of `nodeIDs`, which are binary numbers conveyed in the BIFS bitstream.

b) The scope of `nodeIDs` is given in 7.1.1.1.5.

c) No two nodes in the scene graph may have the same `nodeID` at any point in time.

Nodes that have been assigned a `nodeID` may be re-used, as described in ISO/IEC 14772-1:1998, subclause 4.6.3. Note that this mechanism results in a scene description that is a directed acyclic graph, rather than a simple tree.

The mechanisms that allow modifications to the BIFS scene also depend on the use of `nodeIDs`.

7.1.1.2.11 Standard Units

As described in ISO/IEC 14772-1:1998, subclause 4.4.5, the standard units used in the scene description are the following:

Table 1 — Standard units

Category	Unit
Distance	Meter
Color Space	RGB [0,1] [0,1] [0,1]
Time	Seconds
Angle	Radians

7.1.1.2.12 Mapping of Scenes to Screens

BIFS scenes may contain still images and videos that are to be pixel-copied to the rendering device using their native dimensions as produced at the output of their terminals. The **Bitmap** node (see 7.2.2.22) provides a screen-aligned geometry that has the pixel dimensions of the texture that is mapped onto it.

NOTE — When **Bitmap** is used, the same scene will appear differently on screens with different resolutions. BIFS scenes that do not use the **Bitmap** node are independent from the screen on which they are viewed.

7.1.1.2.12.1 Transparency of visual objects

Content complying with ISO/IEC 14496-1 may include still images or video sequences with representations that include alpha values. These values provide transparency information and are to be treated as specified in ISO/IEC 14772-1:1998,

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

subclause 4.14. For video sequences represented according to ISO/IEC 14496-2, transparency is handled as specified in ISO/IEC 14496-2.

7.1.1.2.13 Special considerations for audio

7.1.1.2.13.1 Audio sub-graphs

Audio nodes are used to build audio scenes in the terminal from audio sources coded with tools specified in ISO/IEC 14496-3. The audio scene description capabilities provide two functionalities:

"Physical modelling" composition for virtual-reality applications, where the goal is to recreate the acoustic space of a real or virtual environment.

"Post-production" composition for traditional content applications, where the goal is to apply high-quality signal processing transformations.

Audio may be included in either 2D or 3D scene graphs. In a 3D scene, the audio may be spatially presented to sound as though it originates from a particular 3D direction, according to the positions of the object and the listener.

The **Sound**, **DirectiveSound, WideSound** and **SurroundingSound** nodes are used to attach audio to 3D scene graphs and the **Sound2D** node is used to attach audio to 2D scene graphs. However, by use of the **Transform3DAudio** node 3D audio nodes can also be used in 2D scenes. As with visual objects, an audio object represented by one of these nodes has a position in space and time, and is transformed by the spatial and grouping transforms of nodes hierarchically above it in the scene.

The nodes below the **Sound, DirectiveSound, WideSound, SurroundingSound**, **Sound2D** nodes, however, constitute an audio sub-graph.This sub-graph is used to describe a particular audio object through the mixing and processing of several audio streams. Rather than representing a hierarchy of spatio-temporal transformations, the nodes within the audio sub-graph represent a signal flow graph that describes how to create the audio object from the audio coded in the AudioSource streams. That is, each audio sub-graph node (**AudioSource**, **AudioMix**, **AudioSwitch**, **AudioFX**, **AudioClip**, **AudioBuffer**, **AdvancedAudioBuffer**, **AudioDelay, AudioChannelConfig**) accepts one or several channels of input audio, and describes how to turn these channels of input audio into one or more channels of output. The only sounds presented in the audio-visual scene are those which are the output of audio nodes that are children of a **Sound** / **DirectiveSound** / **WideSound** / **SurroundingSound** / **Sound2D** node (that is, the "highest" outputs in the audio sub-graph). The remaining nodes represent "intermediate results" in the sound computation process and the sound represented therein is not presented to the user.

The normative semantics of each of the audio sub-graph nodes describe the exact manner in which to compute the output audio the input audio for each node based on its parameters.

7.1.1.2.13.2 Overview of sound node semantics

This subclause describes the concepts for normative calculation of the audio objects in the scene in detail, and describes the normative procedure for calculating the audio signal which is the output of a **Sound, DirectiveSound**, **WideSound**, **SurroundingSound, Sound2D** node given the audio signals which are its input.

Recall that the audio nodes present in an audio sub-graph do not each represent a sound to be presented in the scene. Rather, the audio sub-graph represents a signal-flow graph which computes a single (possibly multi-channel) audio object based on a set of audio inputs (in **AudioSource** nodes) and parametric transformations. The only sounds which are presented to the listener are those which are the "output" of these audio sub-graphs, as connected to a **Sound**, **DirectiveSound**, **WideSound**, **SurroundingSound**, **Sound2D** node. This subclause describes the proper computation of this signal-flow graph and resulting audio object.

As each audio source is decoded, it produces data that is stored in composition memory (CM). At a particular time instant in the scene, the compositor shall receive from each audio decoder a CM such that the decoded time of the first audio sample of the CM for each audio source is the same (that is, the first sample is synchronized at this time instant). Each CM will have a certain length, depending on the sampling rate of the audio source and the clock rate of the system. In addition, each CM has a certain number of channels, depending on the audio source .

Each node in the audio sub-graph has an associated input buffer and output buffer, except for the **AudioSource** node which has no input buffer. The CM for the audio source acts as the input buffer of audio for the **AudioSource** with which the decoder is associated. As with CM, each input and output buffer for each node has a certain length, and a certain number of channels.

As the signal-flow graph computation proceeds, the output buffer of each node is placed in the input buffer of its parent node, as follows:

If an audio node, *N*, has *n* children, and each of the children produces *k(i)* channels of output, for 1 <= *i* <= *n*, then the node, *N*, shall have *k*(1) + *k*(2) + ... + *k*(*n*) channels of input, where the first *k*(1) channels [number 1 through *k*(1)] shall be

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

the channels of the first child, the next *k*(2) channels [number *k*(1)+1 through *k*(1)+*k*(2)] shall be the channels of the second child, and so forth.

Then, the output buffer of the node is calculated from the input buffer based on the particular rules for that node.

7.1.1.2.13.2.1 Sample-rate conversion

If the various children of a **Sound**, **DirectiveSound**, **WideSound**, **SurroundingSound**, **Sound2D** node do not produce output at the same sampling rate, then the lengths of the output buffers of the children do not match, and the sampling rates of the children's' output must be brought into alignment in order to place their output buffers in the input buffer of the parent node. The sampling rate of the input buffer for the node shall be the fastest of the sampling rates of the children. The output buffers of the children shall be resampled to be at this sampling rate. The particular method of resampling is non-normative, but the quality shall be close in accuracy to the DAC that the signal is targeted for, i.e. according to the rule `dB SNR = 6 * (nbits -1)`, where `nbits` is the number of bits corresponding to the maximum bit depth of any of the signals being so converted and/or composited. Aliasing artifacts may be at this level of signal-to-noise ratio. The noise level due to arithmetic accuracy and other uncorrelated noise sources should be below the rule `dB SNR = 6* nbits`.

The output sampling rate of a node shall be the output sampling rate of the input buffers after this resampling procedure is applied.

Content authors are advised that content which contains audio sources operating at many different sampling rates, especially sampling rates which are not related by simple rational values, may produce scenes with a high computational complexity.

EXAMPLE — Suppose that node N has children M1 and M2, all three audio nodes, and that M1 and M2 produce output at S1 and S2 sampling rates respectively, where S1 > S2. Then if the decoding frame rate is F frames per second, then M1's output buffer will contain S1/F samples of data, and M2's output buffer will contain S2/F samples of data. Then, since M1 is the faster of the children, its output buffer values are placed in the input buffer of N. The output buffer of M2 is resampled by the factor S1/S2 to be S1/F samples long, and these values are placed in the input buffer of N. The output sampling rate of N is S1.

7.1.1.2.13.2.2 Number of output channels

If the **numChan** field of an audio node, which indicates the number of output channels, differs from the number of channels produced according to the calculation procedure in the node description, or if the **numChan** field of an **AudioSource** node differs in value from the number of channels of an input audio stream, then the **numChan** field shall take precedence when including the source in the audio sub-graph calculation, as follows:

If the value of the **numChan** field is strictly less than the number of channels produced, then only the first **numChan** channels shall be used in the output buffer.

If the value of the **numChan** field is strictly greater than the number of channels produced, then the "extra" channels shall be set to all 0's in the output buffer.

7.1.1.2.13.3 Audio-specific BIFS Nodes

In the following table, nodes that are related to audio scene description are listed.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 2 — Audio-Specific BIFS Nodes

Node	Purpose	Subclause
AudioBuffer, AdvancedAudioBuffer	Interactively trigger snippets of sound	7.2.2.8, 7.2.2.2
AudioChannelConfig	Label channel configuration of audio data	7.2.2.9
AudioClip	Insert an audio clip into a scene	7.2.2.10
AudioDelay	Add delay to sound	7.2.2.11
AudioFX	Apply post-production effects to sound	7.2.2.12
AudioMix	Mix sounds	7.2.2.14
AudioSource	Define audio source input to a scene	7.2.2.15
AudioSwitch	Switching of audio sources in a scene	7.2.2.16
ListeningPoint	Define listening point in a scene	7.2.2.76
Sound, Sound2D, DirectiveSound, WideSound, SurroundingSound	Define properties of sound	7.2.2.116, 7.2.2.117, 7.2.2.46, 7.2.2.140, 7.2.2.121

7.1.1.2.13.4 Spatialization of sound sources according to the acoustic environment

This specification contains a set of nodes of extended node types, that can be used to include positional and directive sound sources to 3-D BIFS scenes, and process them in a way that the acoustics of the environment is taken into account. These nodes enable parametrization and rendering of the acoustic properties of a virtual environment according to the current relative positions of the sound source, the listening point, and the acoustically relevant objects in the BIFS scene. Such properties are, e.g., room reverberation time (and other statistical room acoustic parameters), speed of sound, acoustic properties of surfaces, and sound source directivity. Functionalities that are made possible with these parameters include immersive audiovisual rendering, room acoustic modeling, and enhanced 3-D sound presentation.

Two distinct approaches of acoustic environment rendering are incorporated in the 3-D sound processing. One is based on physical, or geometrical modeling of the acoustic scene while the second is based on the perceptual description of room acoustic effects. These two schemes of virtual acoustics rendering are referred to as the *physical* and the *perceptual* approach.

The nodes that are involved in the sound environment modeling are **AcousticScene**, **AcousticMaterial**, **DirectiveSound**, and **PerceptualParameters**, and their main functionalities are presented in the table below, and the rendering scheme where they are used is listed in the rightmost column:

Table 3 — Nodes for environmental spatialization of sound

Node	Purpose	Approach	Subclause
AcousticScene	Restrict each audio rendering process to a defined 3-D region in the BIFS scene, and specify a reverberation time that is applied to the sound sources currently within that region.	physical	7.2.2.3
AcousticMaterial	Define sound reflectivity and transmission properties (along with the visual properties) for each acoustically relevant (flat, polygonal) surface.	physical	7.2.2.1
DirectiveSound	Define a directive sound source that also enables natural distance dependent attenuation and air absorption modeling, as well as rendering of the propagation delay between the source and the listener.	physical and perceptual	7.2.2.46
PerceptualParameters	Node for attaching perceptual properties to a directive sound source (DirectiveSound) in order to simulate virtual room effects that do not need to relate to the geometrical and/or visual BIFS scene.	perceptual	7.2.2.93

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In the following, overviews of the physical and perceptual audio rendering schemes are presented.

7.1.1.2.13.4.1 Physical approach

In this approach the acoustics rendering is defined as creating a virtual auditory environment that models an existent or non-existent space. This rendering is called *auralization,* the relation of which to graphics (visualization) is understood as the creation of audiovisual scenes that are perceptually (visually and aurally) relevant. An example of this could be a virtual concert performance, where the acoustical behavior of the space as well as the graphical outlook is modeled. Another example could be a scene, where the listener moves from a very small room to a larger hall, and the changes in the acoustic and graphical rendering is immediately perceived. Also sound sources without a room acoustic response but with effects such as source directivity, Doppler effect, and echoes (distinctive sound reflections) can be modeled. The acoustical behaviors and properties are:

Acoustic properties of surface materials (walls), that enable modeling of sound reflections of surfaces, as well as transmission of sound through them. This way sound reflections are tracked and rendered according to the geometry of the walls and positions of the sound sources and the listener. Obstruction effects are automatically rendered when walls or obstacles are present between the source and the listener

Reverberation time of a specified region in the scene. This enables modeling of reverberating spaces by a simple parameter, and without the necessary need to describe the physical walls of a room.

Acoustic properties of the sound transmitting medium. These include the speed of sound, distance dependent attenuation and lowpass filtering effect caused by air absorption (see ISO 9613-1:1993). Speed of sound is used to control the sound propagation delay between source and the listener, and therefore also the strength of the Doppler effect which depends on the relative motion between the source and the listener.

Directivity characteristics of sound sources. This enables flexible modeling of different sound sources (e.g., human speaker, or a musical instrument). The directivity patterns can be frequency dependent, or it can be defined by a direction dependent coefficient, or in the simplest case the source can be omnidirectional.

In the physical approach, the *geometrical* and *physical sound propagation operator* is used in real time during playback in order to derive the auralization signal processing parameters to be applied to each sound source signal. This propagation operator exploits the knowledge of the positions of the sound sources and the listener relative to the walls to compute the arrival time, amplitude (and spectrum) and direction of arrival for each early reflection. This computation is performed in real time for a limited number of reflections per sound source, with dynamic refresh of reflection parameters according to movements of the sound sources or the listener.

7.1.1.2.13.4.2 Perceptual approach

In this model, the sound transformation associated with room reflections and reverberation is described by a set of perceptual attributes (such as source presence and brilliance, room reverberance, envelopment). These attributes may be manipulated directly and individually for each sound source in the scene.

This approach provides simple and intuitive parameters to the content provider, allowing:

Manipulation of environmental effects for each sound event directly (without requiring that the source or the point of view be moved).

Sound design adjustments beyond the physical constraints implied by the graphic representation, for example:

Distorted or exaggerated distance sensation and room-related effects

Unconstrained spatial sound effects for audio-only scene nodes (no visual correspondence) or when the point of view is out of the room

In this approach, an absolute (exocentric) representation of the sound scene containing several sources and the listener can be manipulated as follows:

The environment (room) is described by setting the values of the perceptual attributes for a reference source-listener distance. These attributes and their values make up a "preset", which specifies, at that reference distance and for an omnidirectional sound source, the delay and intensity of the early reflection, as well as the delay, decay time and spectrum of the late reverberation.

The sound transformation to be applied to each sound event is derived from the above preset by use of a *perceptual sound propagation operator* which takes into account the relative positions and orientations of the sources and the listener, and a model of the directivity of sound sources.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In this model, only the *relative* positions and orientations of the sound sources with respect to the listener are taken into account. The model does not exploit any knowledge of wall positions in order to compute the parameters of the early reflections. The temporal pattern of the early reflections is determined by the definition of the environment “preset”. The perceptual sound propagation operator adjusts one perceptual attribute (called "source presence") according to source-listener distance. Adjusting this single parameter produces a convincing sensation of proximity or remoteness of the sound source. Additionally, the operator takes into account the orientation of the source and its directivity pattern.

7.1.1.2.13.5 Channel configuration aspects in the audio subtree

Audio decoders require a channel configuration that is either known implicitly or is conveyed by some configuration information. Note that ISO/IEC 14496-3, Subpart 4 forbids the use of implicit channel mappings via the PCE (program config element) in subclause 4.5.1.2.1, if MPEG-4 Audio is used together with MPEG-4 Systems audio compositor (i.e., AudioBIFS). In an audio subtree the channel configuration of the decoders or from the DecoderSpecificInfo can normally be used for the loudspeaker mapping behind the sound node, especially in the multichannel case (numChan > 1) if the phase group flags in the audio nodes has been set. Therefore an MPEG-4 Player implementation has to pass this information from the decoder output via the audio nodes to the presenter. Some audio nodes (**AudioMix**, **AudioSwitch** and **AudioFX**) have channel-variant behavior. For these nodes conflicts in the channel configuration transmission can occur, as illustrated in Figure 7 where two different conflicts are shown.

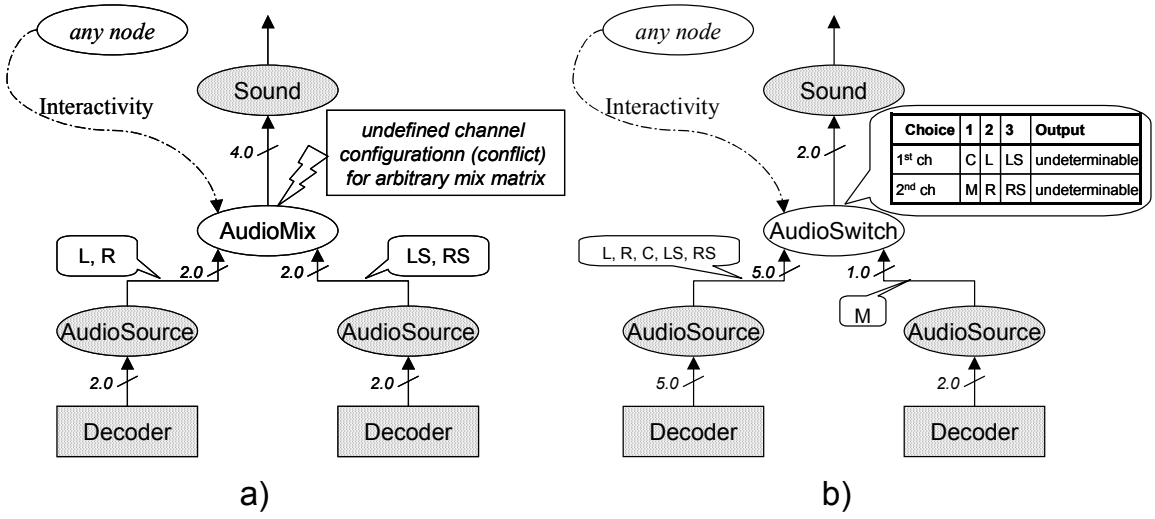


Figure 7 — Channel configuration flow conflicts

The first conflict occurs behind the mix node (Figure 7 a)), where a mix of a first stereo signal and a second stereo signal shall be mixed into 4 channels. The resulting channel configuration after the mix is undefined, since it is not obvious what kind of configuration has been produced. The second conflict occurs in a sequence of whichChoice field updates (Figure 7 b)) behind the **AudioSwitch** node (choice 1…3). In this sequence different channels from the Source node and a single channel from a second Source node are sequentially selected. There is no way to know the desired channel configuration at the output in both cases. Introucing an **AudioChannelConfig** node behind the conflicting node solves this problem by labeling the channels with a determined configuration.

The format of the transmitted channels is restricted to the formats of the decoder configurations. Despite these configuration presets of other configurations are possible and can be composed with the help of an AudioBIFS scene. Two problems can arise in general: missing channel descriptions and composition of formats. It is not foreseen in the channel configuration table of the MPEG-1/2 encoders to transmit multichannel subsets, for example only the 2 surround channels LS, RS. The composition of multichannel formats with the help of several 2-channel streams, for example encoded in the MPEG-1/2 Layer 3 format, cannot be done unambiguously without the **AudioChannelConfig** node as illustrated in Figure 8: a composition of 3 stereo channels to a 6-channel mix can be labeled as a 5.1 multichannel set or as a 6-channel Ambisonics® signal.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

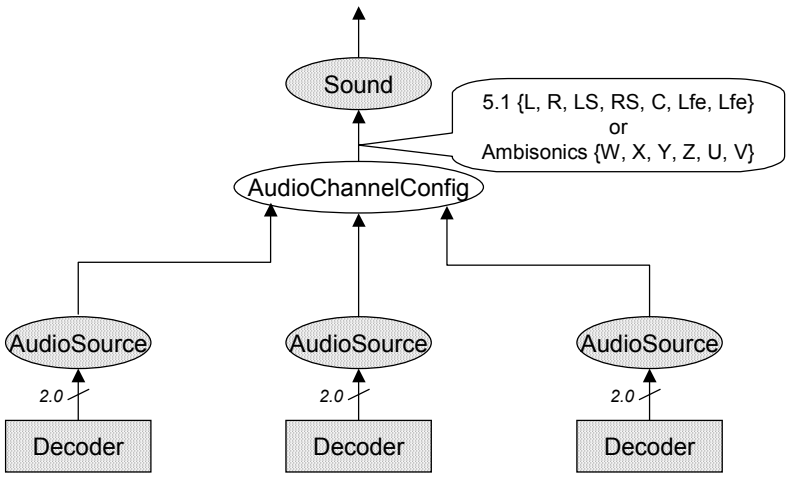


Figure 8 — The audioChannelConfig node for the composition of new formats

7.1.1.2.13.6 Implementation details on SurroundingSound transformations

For the case the SurroundingSound node is "fed" with multi-channel flow labeled as a 1st order Ambisonics® sound field, transformations rely on formulae that already exist. Rotation effects are a quite trivial matter and involve basic trigonometry formulae (see Figure 9). Angular or perspective distortions derive from the existing "Forward dominance" effect.

Forward dominance is an effect to parametrically enlarge or narrow the frontal sound scene (and inversely the back scene). It derives from the "Lorentz Transform" (see Figure 9) applied to the 1st order spherical harmonics. This transform is parameterized by a value λ which range is]0; +inf[, the value 1 corresponding to "no effect". The angle distortion is such that a sound source that was localized at azimuth θ will be moved to azimuth θ', with: $\cos\theta' = (m + \cos\theta)/(1 + \mu \cos\theta)$ and $\mu = (\lambda^2-1)/(\lambda^2+1)$.

If the ListeningPoint moves front or back in the scene, one can mimic the distortion of the angular perspective by applying the "Lorentz Transform" (see Figure 9) with a parameter being a function of the ListeningPoint distance (from the SurroundingSound center) weighted by the distortionFactor. If d is the displacement along the back-front axis, one can propose the following mapping law: $\lambda = \exp(-\alpha d)$, with α being the distortionFactor.

If the ListeningPoint moves along an arbitrary axis (not necessary the front-back axis), one has to rotate the soundfield towards this axis before applying the Lorentz Transform, then rotate it with the opposite angle, towards the front-back axis again.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

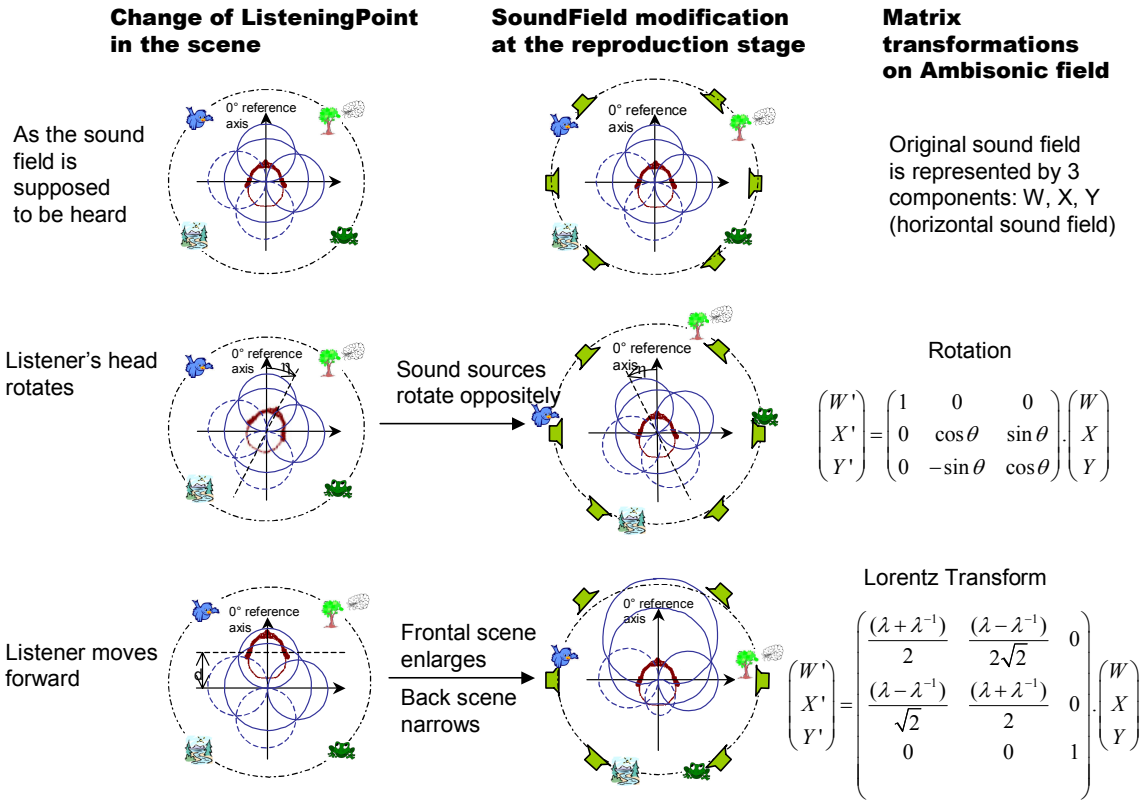


Figure 9 — Ambisonics® Sound Field transformations applied in connection with listener's moves

For non-Ambisonics® materials, angular transformations can be applied by panning the signals over the rendering layout, as if transformations would concern the originally dedicated loudspeakers as being the sources contained in the sound field.

7.1.1.2.14 Bindable Children Nodes

MPEG-4 extends the notion of bindable children nodes described in VRML, ISO/IEC 14772-1:1998. As in VRML, the scene maintains a collection of stacks, one for each type of bindable node. The bindable nodes are the **Viewpoint**, **Viewport**, **Background**, **Background2D**, **Fog**, and **NavigationInfo** nodes. In MPEG-4 some of these nodes may be bound to stacks held in the **Layer2D**, **Layer3D**, **CompositeTexture2D**, and **CompositeTexture3D** grouping nodes. Bindable nodes can be shared between these grouping nodes and the scene using the **set_bind** mechanism, but in the absence of such an event, the top most bindable node within a sub-scene will bind to its inner most grouping node that holds a stack for it.

The following rules shall apply in the absence of a set_bind event:

- The top-most **Background**, **Fog**, **NavigationInfo**, and **Viewpoint** nodes in a **Layer3D** or **CompositeTexture3D** node shall be bound to the respective stacks within that node.
- The top-most **Background2D** and **Viewport** nodes in a **Layer2D** or **CompositeTexture2D** node shall be bound to the respective stacks within that node.
- The top-most **Background2D** node in a **Layer3D** or **CompositeTexture3D** node shall be bound to the same stack as the **Background** node and shall appear in the **background** field of those nodes.
- The top-most bindable node that is not a child of any grouping node that holds a stack for it shall be bound to the respective scene stack for that node.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTE - For example, a **Background2D** node that is a child of an **OrderedGroup** or **Group** node would bind to the scene stack. A **Background2D** node that is a child of a **Layer3D** or **CompositeTexture3D** node would be bound using the same stack as for **Background** in that node and would render behind all other geometries in the frame of that node. A **Background** node in a **Layer2D** would bind to the scene stack.

7.1.1.3 Sources of modification to the scene

7.1.1.3.1 Interactivity and behaviors

To describe interactivity and behavior of scene objects, the event architecture defined in ISO/IEC 14772-1:1998, subclause 4.10, is used. Sensors and routes describe interactivity and behaviors. Sensor nodes generate events based on user interaction or a change in the scene. These events are routed to interpolator or other nodes to change the attributes of these nodes. If routed to an interpolator, a new parameter is interpolated according to the input value, and is finally routed to the node which must process the event.

Events shall be generated and evaluated in the order in which their ROUTEs appear in or are inserted into the scene. Events are collected and applied to the scene together, with subsequently generated events collected and applied together repeatedly in an event cascade.

NOTE – the event cascade model differs from a depth-first, immediate execution model. In the latter, side effects from node deletions can create race conditions which would lead to different results on different implementations.

7.1.1.3.1.1 Attaching ROUTEIDs to routes

`ROUTEIDs` may be attached to routes using the DEF mechanism, described in ISO/IEC 14772-1:1998, subclause 4.6.2. This allows routes to be subsequently referenced in BIFS-Command structures. `ROUTEIDs` are integer values and the namespace for routes is distinct from that of `nodeIDs`. The number of bits used to represent these integer values is specified in the BIFS `DecoderConfigDescriptor`.

The scope of `ROUTEIDs` is defined in see 7.1.1.1.5. The following restrictions apply:

a) Routes are identified by the use of `ROUTEIDs`, which are binary numbers conveyed in the BIFS bitstream.

b) The scope of `ROUTEIDs` is given in 7.1.1.1.5.

c) No two routes in the scene graph may have the same `ROUTEID` at any point in time.

The mechanisms that allow modifications to the BIFS scene also depend on the use of `nodeIDs` (see 7.1.1.2.10). The USE mechanism shall not be used with routes.

7.1.1.3.1.2 Conditional node

The **Conditional** node (see 7.2.2.36) allows BIFS-Commands to be described in the scene which shall only be applied to the scene graph when an event is received on one of the **Conditional** node's inputs.

7.1.1.3.2 External modification of the scene: BIFS-Commands

The BIFS-Command mechanism enables the change of properties of the scene graph, its nodes and behaviors.

EXAMPLE — **Transform** nodes can be modified to move objects in space; **Material** nodes can be changed to modify an object's appearance, and fields of geometric nodes can be totally or partially changed to modify the geometry of objects.

7.1.1.3.2.1 Overview

BIFS-Commands are used to modify a set of properties of the scene at a given time instant in time. Commands are grouped into `CommandFrames` (see 8.6.2) in order to be able to send several commands in a single access unit. The following four basic commands are defined:

1. Replacement of an entire scene
2. Insertion
3. Deletion
4. Replacement

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The first of these commands allows the replacement of the entire BIFS scene. The replacement of the entire scene requires a scene graph representing a valid BIFS scene to be transmitted. The `SceneReplace` command is the only random access point in the BIFS stream.

The other three commands can be used to update the following structures:

1. A node
2. An eventIn, exposedField or an indexed value in an MFField
3. A ROUTE

In order to modify the scene the sender must transmit a BIFS `CommandFrame` that contains one or more update commands. A single source of BIFS-Commands is assumed. The identification of a node in the scene is provided by a `nodeID`. Note that it is the sender's responsibility to provide this `nodeID`, which must be unique (see 7.1.1.1.5). The identification of a node's fields is provided by sending the `INid` of the field (see node coding tables in electronic attachment).

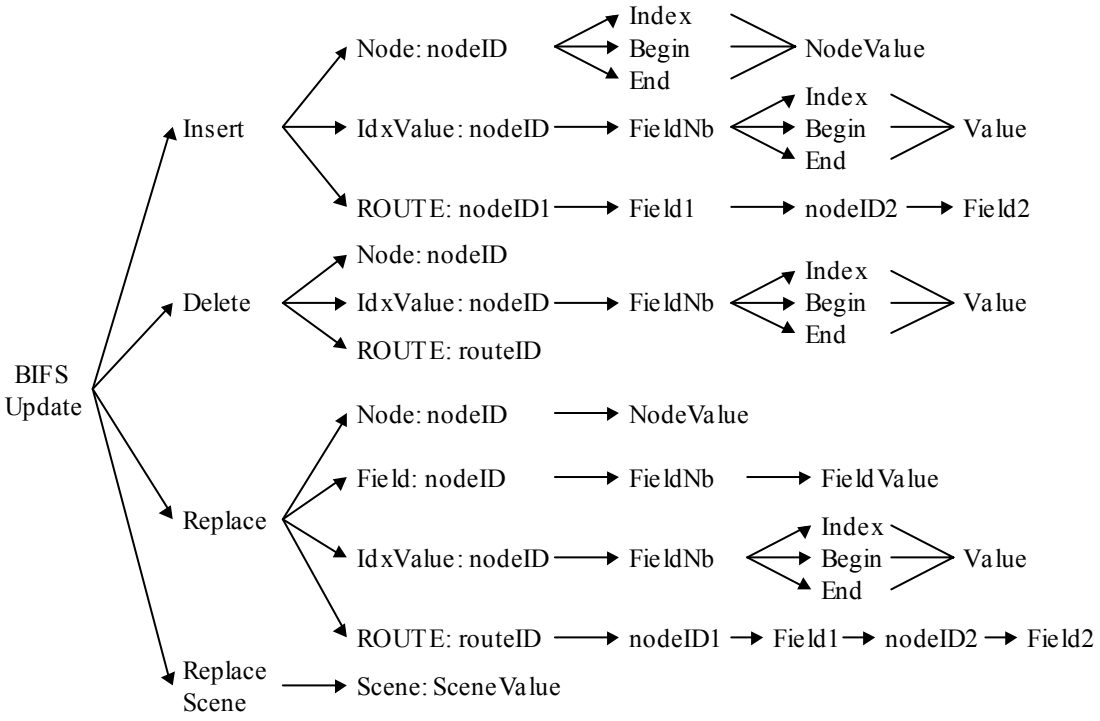


Figure 10 — BIFS-Command Types

7.1.1.3.2.2 Modification of indexed values

Insertion of an indexed value in a field implies that all later values in the field have their indices incremented and the length of the field increases accordingly. Appending a value to an indexed value field also increases the length of the field but the indices of existing values in the field do not change.

Deletion of an indexed value in a field implies that all later values in the field have their indices decremented and the length of the field decreases accordingly.

7.1.1.3.2.3 Timing of BIFS-Commands

The time at which a BIFS-Command is applied shall be the composition time stamp of the access unit in which the command is contained, as defined in the sync layer (see 7.3.2, ISO/IEC 14496-1).

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.1.3.3 External animation of the scene: BIFS-Anim

BIFS-Anim provides for the continuous update of the certain fields of nodes in the scene graph. BIFS-Anim is used to integrate different kinds of animation, including the ability to animate face models as well as meshes, 2D and 3D positions, rotations, scale factors, and color attributes. Although BIFS-Anim and BIFS-Command have the same elementary stream type (see Table 5, ISO/IEC 14496-1) they may not occupy the same elementary stream. BIFS-Anim information is conveyed in a separate elementary stream from that which carries BIFS-Command elements.

7.1.1.3.3.1 Overview

BIFS-Anim elementary streams consist of a sequence of `AnimationFrames`. The `AnimationMask`, which is required to interpret these `AnimationFrames`, is transmitted in the `DecoderSpecificInfo` for the BIFS-Anim elementary stream in the corresponding object descriptor (see 7.2.6.7, ISO/IEC 14496-1).

7.1.1.3.3.2 BIFS-Anim configuration

The `AnimationMask` contains one `ElementaryMask` for each node that is to be animated. These `ElementaryMasks` specify the fields that are contained in the `AnimationFrames` for a given animated node, and their associated quantization parameters. Only eventIn or exposedField fields that have an animation method (see node coding tables electronic attachment and 7.1.1.3.3.3) can be modified using BIFS-Anim. Such fields are called dynamic fields. In addition, the animated field must be part of an updateable node; that is, a node that has been assigned a `nodeID`. The `AnimationMask` is composed of several elementary masks defining these parameters.

7.1.1.3.3.3 BIFS-Anim animation parameters

Animation parameters are transmitted as a sequence of `AnimationFrames`. `AnimationFrames` specify the values of the dynamic fields of updateable nodes that are being animated in BIFS-Anim streams. An `AnimationFrame` contains the new values of all animated parameters at a specified time, unless if it is specified that, for some frames, these parameters are not sent. The parameters can be sent in Intra (the absolute value is sent) and Predictive modes (the difference between the current and previous values is sent).

Animation parameters can be applied to any eventIn or exposedField of any updateable node of a scene which has an assigned animation method (see node coding tables in electronic attachments).

NOTE — Some node tables, in the electronic attachment, contain an eventIn or exposedField that has an animation method but for which there is no associated `dynID`. This is the case when only one exposedField or eventIn in a node has an animation method. In such cases, it is not necessary for the field to have a `dynID` since the terminal can assume that BIFS-Anim animations for this type of node refer to the only dynamic field of the node.

The types of dynamic fields are:

SFInt32/MFInt32

SFFloat/MFFloat

SFRotation/MFRotation

SFColor/MFColor

SFVec2f/MFVec2f

SFVec3f/MFVec3f

SFVec4f/MFVec4f

7.1.1.3.4 Order of application of modifications to the scene

Where modifications to the scene graph, resulting from the use of more than one of the permitted methods, must be applied simultaneously, the following order of application shall be observed:

1. BIFS-Anim
2. Conditional node
3. BIFS-Command

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2 Node Semantics

7.2.1 Overview

The BIFS nodes include nodes that have been defined in ISO/IEC 14772-1:1998. For these nodes, the semantic information is given by normative reference with any restrictions defined herein.

7.2.2 Node specifications

7.2.2.1 AcousticMaterial

7.2.2.1.1 Node interface

AcousticMaterial {

field	SFFloat	**reffunc**	0
field	SFFloat	**transfunc**	1
field	MFFloat	**refFrequency**	[]
field	MFFloat	**transFrequency**	[]
exposedField	SFFloat	**ambientIntensity**	0.2
exposedField	SFColor	**diffuseColor**	0.8, 0.8, 0.8
exposedField	SFColor	**emissiveColor**	0, 0, 0
exposedField	SFFloat	**shininess**	0.2
exposedField	SFColor	**specularColor**	0, 0, 0
exposedField	SFFloat	**transparency**	0

}

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.1.2 Functionality and semantics

The **AcousticMaterial** node is used for attaching acoustic and visual properties to surfaces (planar polygons) defined by an **IndexedFaceSet** node that is a sibling or exist in a sub-graph of a sibling of an **AcousticScene** node. The fields of this node define the visual appearance properties, as well as sound reflection and transmission properties of the **IndexedFaceSet** surfaces it is attached to. It is used in the material field of an **Appearance** node that is attached to a **Shape** node under which the **IndexedFaceSet** is defined. Each polygon in an **IndexedFaceSet** that **AcousticMaterial** is associated with can produce a single specular reflection to sound whenever a corresponding sound image source is visible to the listening point (**Viewpoint** or **ListeningPoint**), or obstruct sound transmission when it appears between the sound source and the listener. Note that these reflectivity and sound transmission properties of a surface are only applied to sounds that are attached to a 3-D scene with a **DirectiveSound** and **WideSound** nodes. The delay of a reflection (a predelay that is added to sound) is computed from the relative distance between the image source corresponding to the reflection, and the speed of sound which is given as a field in the **DirectiveSound** and **WideSound** nodes (see 7.2.2.46).

There are two different ways of defining the reflectivity and transmission properties of **AcousticMaterial:**

The **reffunc** and **refFrequency** fields specify the sound reflectivity of the material. If **refFrequency** is an empty vector, **reffunc** is a system function representation of a linear, time-invariant system, the reflectivity transfer function of a digital filter for that material. Generally, a system function $H(z)$ is represented in the z-domain as a division of the z-transform of the output sequence $Y(z)$ with the z-transform of the input sequence $X(z)$:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}}.$$

The reflection function is given as digital filter coefficients in the following order:

$$[b_0 b_1 b_2 ... a_1 a_2 ...]$$

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Thus, a simple scalar value b_0 can be given to a material for frequency-independent reflectivity of a surface. On the other hand, complex reflection functions can also be represented using this formulation. For example, if the **reffunc** field is 1, the amplitude of the reflection of sound off a surface will be the same as that of the incident sound, and if the field is set to 0, no sound will reflect off that surface. The default value of this field is 0, implying no reflectivity.

If **refFrequency** is different from an empty vector, the semantics of the **reffunc** is different than described above. In this case **refFrequency** specifies a set of frequencies (in Hz) at which the gains in **reffunc** field are valid; The filter applied to sound when it is reflected off this surface implements a frequency magnitude response where at the given frequencies (in **refFrequency** field) the gains in **reffunc** field are valid. An example of **refFrequency** field is:

[250 1000 2000 4000],

and an example of **reffunc** in this approach is:

[0.75 0.9 0.9 0.2]

The **transfunc** and **transFrequency** fields specify the transmission properties of the material, e.g., the filtering that is applied to sound when it passes through an **IndexedFaceSet** surface this **AcousticMaterial** is attached to, when the **IndexedFaceSet** surface appears on the direct path between the sound source and the listener. The transmission function is given similarly as in the reflectivity in **reffunc** and **refFrequency** fields with two different ways of expressing the filtering.

The fields **ambientIntensity, diffuseColor, emissiveColor,** and **shininess** are used for the visual appearance rendering similarly as in the **Material** node.

7.2.2.2 AdvancedAudioBuffer

7.2.2.2.1 Node interface

AdvancedAudioBuffer {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFBool	**loop**	FALSE
exposedField	SFFloat	**pitch**	1.0
exposedField	SFTime	**startTime**	0
exposedField	SFTime	**stopTime**	0
exposedField	SFTime	**startLoadTime**	0
exposedField	SFTime	**stopLoadTime**	0
exposedField	SFInt32	**loadMode**	0
exposedField	SFInt32	**numAccumulatedBlocks**	0
exposedField	SFInt32	**deleteBlock**	0
exposedField	SFInt32	**playBlock**	0
exposedField	SFFloat	**length**	0.0
field	SFInt32	**numChan**	1
field	MFInt32	**phaseGroup**	[1]
eventOut	SFTime	**duration_changed**	
eventOut	SFBool	**isActive**	

}

NOTE — For binary encoding of this node see node coding tables in electronic attachment

7.2.2.2.2 Functionality and semantics

The **AdvancedAudioBuffer** node provides an interface for stored sound. It can be used instead of the **AudioBuffer** node. This node has corrected functionality and enhanced reload mechanism compared to the **AudioBuffer** node, e.g to accumulate snippets of sound in the **AdvancedAudioBuffer**. These snippets can be accessed directly or as the full accumulated content.

The functionality of the **children, loop**, **pitch, startTime, stopTime, numChan, phaseGroup, duration_changed** and **isActive** fields is described in 7.2.2.8, **AudioBuffer** node.

The **length** field specifies the length in seconds of the audio buffer. Audio data should be buffered at the instantiation of the node and whenever the **length** field changes. Additional modes to control loading, deletion and demand of data can be selected with **loadMode** as defined in Table 4.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 4 — loadMode association table

Value	LoadMode	Functionality
0	compatibility mode	Audio data should be buffered immediately after the node has been instantiated and whenever the **length** field changes. **startLoadTime, stopLoadTime, numAccumulatedBlocks, deleteBlock** and **playBlock** have no effect.
1	reload mode	**startLoadTime** and **stopLoadTime** are valid. When the time in **startLoadTime** is reached, the internal buffer is cleared and the samples at the input of the node are stored until **stopLoadTime** is reached or the stored data have the length defined in **length** . If **startLoadTime** >= **stopLoadTime** a data block with the length defined in the **length** field will be loaded when **startLoadTime** is reached. **numAccumulatedBlocks, deleteBlock** and **playBlock** have no effect.
2	accumulate mode	An audio data block defined by the interval between **startLoadTime** and **stopLoadTime** will be appended at the end of the buffer. The index of the internal audio data blocks has to be continous to be addressable. When the limit defined by **length** will be reached loading will be finished. **numAccumulatedBlocks** has no effect.
3	continuous accumulate mode	An audio data block defined by the interval between **startLoadTime** and **stopLoadTime** will be appended at the end of the buffer. The index of the internal audio data blocks has to be continous to be addressable. When the limit defined by **length** will be reached the oldest audio data block(s) has/have to be discarded until the data are stored. **numAccumulatedBlocks** has no effect.
4	accumulate mode with limited number of buffer blocks	In the accumulate mode the number of stored blocks are limited to **numAccumulatedBlocks.** **length** has no effect.
5…7	reserved for ISO use	

A transition from 0 to a value below 0 in the **deleteBlock** field starts deletion of a data block relative to the latest data block in the following modes: ‘accumulate mode’, ‘continuous accumulate mode’ and ‘accumulate mode with limited number of buffer blocks’. The latest block will be addressed with –1 the previous audio data block with –2 etc.

playBlock defines the block to be played. If **playBlock** is set to 0 (default) the whole content will be played regarding to the **startTime** and **stopTime** conditions (compatibility mode). A value < 0 addresses a block relative to the latest block**.** The latest block will be addressed with –1 the previous audio data block with –2 etc.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.3 AcousticScene

7.2.2.3.1 Node interface

```
AcousticScene {
    field          SFVec3f     center          0 0 0
    field          SFVec3f     Size            -1 –1 –1
    field          MFTime      reverbTime      0
    field          MFFloat     reverbFreq      1000
    exposedField   SFFloat     reverbLevel     0.4
    exposedField   SFTime      reverbDelay     0.5
}
```

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.3.2 Functionality and semantics

AcousticScene is a node the parameters of which are used for rendering of the acoustic response of the environment, together with the acoustic reflectivity or transmission defined in the siblings or their sub-graphs of this **AcousticScene**. **AcousticScene** also defines three fields (**reverbTime**, **reverbFreq**, **reverbLevel**, and **reverbDelay**) which can be used to add reverberation to sounds that are affected by this node. Only audio that has been attached to the scene through a **DirectiveSound** node performing the *physical* rendering scheme is spatialized according to these definitions.

Only those **IndexedFaceSet** nodes that **AcousticMaterial** node is associated with, and that are defined in the siblings of **AcousticScene** (or in the sub-graph of the siblings) have effect on the room acoustic response that is applied to sound sources. Only **DirectiveSound** nodes that are currently positioned in a 3-D rectangular region defined by **center** and **size** fields, are affected by these acoustic surfaces. **size** field defines the size of a rectangular 3-D region where the parameters of **AcousticScene** and the acoustic surfaces in the siblings or sibling sub-graphs of the **AcousticScene** are taken into account in the auralization process (sound processing according to the acoustics of the environment). The default value of this field is –1, -1, -1.

center specifies the center of the above described region in the local coordinate system of the scene. Only when at the decoder a **DirectiveSound** and the **Viewpoint** (or **ListeningPoint**) are located within the same **AcousticScene** region defined by its **center** and **size** the sound attached to the **DirectiveSound** is heard. The default value (-1, –1, –1) of size equals to an infinite rectangular region (i.e., the sound is heard everywhere in the scene). **DirectiveSound** is rendered at one time only according to one **AcousticScene**, i.e., if the source and the viewpoint are in an overlapping area of several **AcousticScene**s, the one which is the first in the rendering order has effect on the **DirectiveSound.**

The **reverbTime** field specifies the reverberation time (time of 60 dB attenuation in the late reverberation response) at frequencies given in **reverbFreq** field to be applied to each **DirectiveSound** node that is within the 3-D region specified by the **AcousticScene**. This information is used for producing late reverberation at the maximum quality possible. With the default value 0, late reverberation is not added to the room response. It should be noted, however, that this field is useful for enabling simple room response modeling whenever there is not enough computational power to render several room reflections, or when the reflective properties of the surfaces are not specified. I.e., it is possible to specify a reverberant room with the boundaries defined by the **size** and **center** fields, even without specifying the reflectivity of individual surfaces. If only one value of **reverbTime** is given, it is taken as the reverberation time at the 1kHz frequency, and the decision about the frequency dependence of the reverberation time would be decided at the terminal (in natural environments the reverberation time decreases as a function of frequency). An example of **reverbTime** field is:

[2.0 0.5],

and example of **reverbFreq** is:

[0 16000],

yielding a late reverberation with a reverberation time of 2.0 s at 0 Hz frequency, and 0.5 s at 16000 Hz frequency.

reverbDelay specifies the time delay between the direct sound and the start of the reverberation in seconds. **reverbLevel** defines the level of the first output from the reverberator with respect to the direct sound.

In order to define which **AcoustcScene** is applied to **DirectiveSound** in the case that it is positioned in an overlapping area of more than one **AcousticScene**, an **OrderedGroup** can be used above the various **AcousticScene**s.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.4 Anchor

7.2.2.4.1 Node interface

```
Anchor {
    eventIn         MFNode      addChildren
    eventIn         MFNode      removeChildren
    exposedField    MFNode      children        []
    exposedField    SFString    description     ""
    exposedField    MFString    parameter       []
    exposedField    MFString    url             []
    eventIn         SFBool      activate
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.4.2 Functionality and semantics

The semantics of the **Anchor** node are specified in ISO/IEC 14772-1:1998, subclause 6.2. ISO/IEC 14496-1 does not support the bounding box parameters (**bboxCenter** and **bboxSize**). Upon reception of an SFBool event of value TRUE on the **activate** eventIn the Anchor node will activate and behavior will be identical to that which would result from activating (e.g. clicking) any geometry contained within its **children** field.

7.2.2.5 AnimationStream

7.2.2.5.1 Node interface

```
AnimationStream {
    exposedField    SFBool      loop            FALSE
    exposedField    SFFloat     speed           1.0
    exposedField    SFTime      startTime       0
    exposedField    SFTime      stopTime        0
    exposedField    MFString    url             [""]
    eventOut        SFBool      isActive
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.5.2 Functionality and semantics

The **AnimationStream** node is designed to implement control parameters for a scene description stream.

The **loop**, **startTime**, and **stopTime** exposedFields and the **isActive** eventOut, and their effects on the **AnimationStream** node are described in 7.1.1.1.6.2.

The semantics of the **speed** exposedField are identical to those for the **MovieTexture** node (see 7.2.2.82).

The **url** field specifies the data source to be used. The data source referred to shall be a BIFS-Anim stream (see also 7.1.1.3.3) or a BIFS-Command stream. In both cases, the stream shall operate within the same name scope as the scene containing the **AnimationStream** node.

7.2.2.6 Appearance

7.2.2.6.1 Node interface

```
Appearance {
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

exposedField	SFNode	**material**	NULL
exposedField	SFNode	**texture**	NULL
exposedField	SFNode	**textureTransform**	NULL

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.6.2 Functionality and semantics

The semantics of the **Appearance** node are specified in ISO/IEC 14772-1:1998, subclause 6.3.

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/IEC14772-1 (the object is unlit and has color 1 1 1).

7.2.2.7 ApplicationWindow

7.2.2.7.1 Node interface

ApplicationWindow {

exposedField	SFBool	**isActive**	FALSE
exposedField	SFTime	**startTime**	0
exposedField	SFTime	**stopTime**	0
exposedField	SFString	**description**	“”
exposedField	MFString	**parameter**	[]
exposedField	MFString	**url**	[]
exposedField	SFVec2f	**size**	0, 0

}

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.7.2 Functionality and semantics

ApplicationWindow is an SF2DNode that allows an external application such as a web browser to exist within the MPEG-4 scene graph. Unlike a texture node, the windowed region is controlled and rendered by the external application, allowing natural user interaction with the application. The particular application to be opened is signaled in the **url** field, and any required parameters for starting the application may be placed in the **parameter** field.

The position of the application, its dimension and whether the application is active or not, is specified through BIFS scene authoring.

The **startTime** exposed field indicates when the application is to be started. The application is given control of the rendering window defined by the size field.

The **stopTime** exposedField indicates that the application is finished and should be shut down. The rendering window defined by the size field is returned to the MPEG-4 player.

The **isActive** exposedField signals the application to relinquish its rendering window to the MPEG-4 player, but to continue to run.

The **description** exposedField allows a prompt to be displayed as an alternative to the url in the **url** field. This choice should be user selectable.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The **parameter** exposedField carries parameters to be interpreted by the application decoder when the application window is instantiated.

The **url** exposedField carries the location of the windowed application.

The **size** exposedField provides the dimension (width and height) of the application window.

7.2.2.8 AudioBuffer

7.2.2.8.1 Node interface

AudioBuffer {

exposedField	SFBool	**loop**	FALSE
exposedField	SFFloat	**pitch**	1.0
exposedField	SFTime	**startTime**	0
exposedField	SFTime	**stopTime**	0
exposedField	MFNode	**children**	[]
exposedField	SFInt	**numChan**	1
exposedField	MFInt	**phaseGroup**	[1]
exposedField	SFFloat	**length**	0.0
eventOut	SFTime	**duration_changed**	
eventOut	SFBool	**isActive**	

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.8.2 Functionality and semantics

The **AudioBuffer** node provides an interface to short snippets of sound to be used in an interactive scene.

EXAMPLE — Sounds triggered as “auditory icons” upon mouse clicks.

It buffers the audio generated by its children to support random restart capability upon interaction events. It differs from the **AudioClip** node in the following ways:

AudioBuffer can be used in broadcast and other one-way applications in which URLs from remote locations cannot be retrieved interactively

AudioBuffer can be used to trigger sounds made from processed sound (ie, with the other sound nodes) rather than only raw sound data as transmitted in the elementary stream

The **loop**, **startTime**, and **stopTime** exposedFields and the **isActive** eventOut, and their effects on the **AudioBuffer** node are described in 7.1.1.1.6.2.

The **length** field specifies the length in seconds of the audio buffer. Audio shall be buffered at the instantiation of the node, and whenever the **length** field changes.

The **pitch** field specifies a pitch-shift to apply to the output sound. The pitch-shift is calculated by simple resampling; that is, a pitch-shift of 2 corresponds to playing the sound twice as fast and an octave higher. If **pitch** is negative, the buffer is played backwards at the indicated speed, beginning at the last sample in the buffer and proceeding to the first, then returning to the last sample if **loop** is TRUE.

The **children** field specifies the child nodes that provide the sound for this node. Each child shall be an AudioBIFS node; that is, one of the following: **AudioSource**, **AudioDelay**, **AudioMix**, **AudioSwitch**, **AudioFX**, **AudioClip, AudioChannelConfig**, **AudioBuffer** or **AdvancedAudioBuffer**.

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.

The **numChan** field specifies the number of output channels of this node. If there are more output channels than input channels, the “extra” channels shall contain all 0s; if there are more input channels than output channels, the “extra” channels shall be ignored.

The **phaseGroup** field specifies phase relationships in the output of the node, see 7.1.1.2.13 and 7.2.2.15.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The output of this node is not calculated based on the current input values, but according to the **startTime** event, the **pitch** field and the contents of the clip buffer. When the **startTime** is reached (that is, the current scene time is greater than or equal to **startTime**), the sound output shall begin at the beginning of the clip buffer and **isActive** shall be set to TRUE. At each time step thereafter, the value of the output buffer shall be the value of the next portion of the clip buffer, upsampled or downsampled as necessary according to **pitch**. When the end of the clip buffer according to the value of **length** is reached, if **loop** is TRUE, the audio shall begin again from the beginning of the clip buffer; if **loop** is FALSE, the playback shall cease. This playback shall be continued until **stopTime** is reached. When the current scene time is greater than or equal to **stopTime**, the node shall cease to produce sound.

The clip buffer shall be calculated as follows. When the node is instantiated, or whenever the **length** field is changed, the first **length** seconds of the audio input to the **AudioBuffer** node shall be copied to the clip buffer. That is, after t seconds, where t < length, audio sample number t * S of channel i (where 0 <= i < **numChan**) in the buffer is set to contain the audio sample corresponding to time t of channel i of the input, where S is the sampling rate of this node. After the first length seconds, the input to this node has no effect. Changes to the **length** field that are received when **isActive** is TRUE shall be ignored.

When the playback is not active, the audio output of the node is all 0s.

7.2.2.9 AudioChannelConfig

7.2.2.9.1 Node interface

AudioChannelConfig {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[ ]
exposedField	SFInt32	**generalChannelFormat**	0
exposedField	SFInt32	**fixedPreset**	0
exposedField	SFInt32	**fixedPresetSubset**	0
exposedField	SFInt32	**fixedPresetAddInf**	0
exposedField	MFInt32	**channelCoordinateSystems**	[ ]
exposedField	MFFloat	**channelSoundLocation**	[ ]
exposedField	MFInt32	**channelDirectionalPattern**	[ ]
exposedField	MFVec3f	**channelDirection**	[ ]
exposedField	SFInt32	**ambResolution2D**	1
exposedField	SFInt32	**ambResolution3D**	0
exposedField	SFInt32	**ambEncodingConvention**	0
exposedField	SFFloat	**ambNfcReferenceDistance**	1.5
exposedField	SFFloat	**ambSoundSpeed**	340.0
exposedField	SFInt32	**ambArrangementRule**	0
exposedField	SFInt32	**ambRecombinationPreset**	0
exposedField	MFInt32	**ambComponentIndex**	[]
exposedField	MFFloat	**ambBackwardMatrix**	[]
exposedField	MFInt32	**ambSoundfieldResolution**	[]
field	SFInt32	**numChannel**	0

}

NOTE — For binary encoding of this node see node coding tables in electronic attachment

7.2.2.9.2 Functionality and semantics

This node is used to label the audio data in the audio subtree to supply the audio presenter with the required information for multichannel or soundfield signals. This is necessary in the following cases:

- channel configuration conflicts
  Some audio nodes have channel-variant behavior (**AudioMix**, **AudioSwitch** and **AudioFX**). For these nodes conflicts in the channel configuration can occur, e.g. if the **matrix** field in an **AudioMix** specifies an interchannel mixing.

- insufficient information
  A decoder can be used to transport subsets of multichannel formats that are not defined in the audioChannel config table (ISO/IEC 14496-3), e.g. the LS and RS channels of a 5.1 multichannel system.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- construction of soundfields
  E.g., a 5.1 soundfield can be transmitted with 3 two-channel audio decoders. This soundfield can be composed and labeled with the **AudioChannelConfig** node in the audio subtree.

- unsupported soundfield formats
  A decoder can be used to transport new or unsupported soundfields, for example an Ambisonics® soundfield. The soundfield formats are not defined in the audioChannel config table (ISO/IEC 14496-3). These soundfields can be composed and labeled with the AudioChannelConfig node in the audio subtree.

The node has the standard audio node interfaces, but no signal processing capability. The samples are passed through and get new channel configuration information.

A heterogenous audiochannel configuration cannot be described with this node – and does not make any sense. Therefore a **phaseGroup** does not exist in this node. Due to ambiguity between information provided in the **phaseGroup** field and the **AudioChannelConfig** node all nodes above the **AudioChannelConfig** node shall ignore the **phaseGroup** fields.

The **addChildren** eventIn specifies a list of nodes that shall be added to **children**.

The **removeChildren** eventIn specifies a list of nodes that shall be removed from **children**.

The **children** array specifies the nodes affected by the **AudioChannelConfig**. Each child shall be one of the following nodes: **AudioSource**, **AudioDelay**, **AudioMix**, **AudioSwitch**, **AudioFX**, **AudioClip** or **AudioBuffer.**

The **numChan** field specifies the number of channels of audio output by this node.

The generalChannelFormat determines the general format of the channel label as described in Table 5**.**

Table 5 — generalChannelFormat table

Value	Configuration	Description	Valid fields
0	pass through	no labeling	none
1	ChannelPreset	Select the fixed values from the Channel Configuration Table in ISO/IEC 14496-3	**fixedPreset** **fixedPresetAddInf**
2	ChannelPreset-Subset	Select a subset from the Channel Configuration Table in ISO/IEC 14496-3 according to the mask in the **fixedPresetSubset** field	**fixedPreset** **fixedPresetSubset** **fixedPresetAddInf**
3	ParametricChan-nelOriented	parametric channel oriented configuration	**channelCoordinateSystems** **channelSoundLocation** **channelDirectionalPattern** **channelDirection**
4	ParametricAmbi-sonicsOriented	parametric Ambisonics® configuration	**ambResolution2D** **ambResolution3D** **ambEncodingConvention** **ambNfcReferenceDistance** **ambSoundSpeed** **ambArrangementRule** **ambRecombinationPreset** **ambComponentIndex** **ambBackwardMatrix** **ambSoundfieldResolution**
5..7	reserved for ISO use		

If generalChannelFormat is set to 1 (ChannelPreset) a fixed preset from the Channel Configuration in ISO/IEC 14496-3 is selected by the value of **fixedPreset**. Additional channel preset information can be specified from Table 6 by the value of **fixedPresetAddInf**. The order of the channels in AudioBIFS shall be adopted from Table 1.11 in ISO/IEC 14496-3, Subpart 1. I.e., the order starts with the center speaker (if present), continues with the

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

stereo pairs, each beginning with the left followed by the right channel and ends with a back center speaker (if present) and the LFE channel (if present).

If **generalChannelFormat** is set to 2 (ChannelPresetSubset), a fixed preset of Channel Configuration in ISO/IEC 14496-3 is selected by the value of **fixedPreset.** A subset of the selected fixed preset is masked by **fixedPresetSubset**. Each bit of the value in this field represents an input channel according to the channel to speaker mapping column of the Channel Configuration Table in ISO/IEC 14496-3 where the first channel is indicated with the LSB of the **fixedPresetSubset** field.

EXAMPLE — fixedPreset=6 (5+1 multichannel) is the channel to speaker mapping C, L, R, LS, RS, LFE. This preset can be used to describe a 2-channel stream containing the LS and RS channels by masking the 4th and the 5th element of the fixed preset with **fixedPresetSubset** set to [MSB…LSB] [...0 0 1 1 0 0 0] = 24_{10}

Additional channel preset information can be specified from Table 6 by the value of **fixedPresetAddInf.**

The **fixedPresetAddInf** can set additional properties of the channels labeled with **fixedPreset** and **fixedPresetSubset.** Values for these additional labels can be found in Table 6:

Table 6 — fixedPresetAddInf table

Value	Configuration
0	none
1	Binaural
2	Transaural
3..7	reserved for ISO use

If **generalChannelFormat** is set to 3 (ParametricChannelOriented), **channelCoordinateSystems, channelSoundLocation**, **channelDirectionalPattern** and **channelDirection** for a generic format are valid. For its functional description see 7.2.2.9.2.1.

If **generalChannelFormat** is set to 4 (ParametricAmbisonicsOriented), **ambResolution2D, ambResolution3D, ambEncodingConvention, ambNfcReferenceDistance, ambSoundSpeed, ambArrangementRule, ambComponentIndex, ambBackwardMatrix,** and **ambSoundfieldResolution** are valid. For its functional description see 7.2.2.9.2.2.

7.2.2.9.2.1 Generic channel configuration parameters

The **channelCoordinateSystems** field determines the coordinate system for the corresponding channels. Each channel can have its own coordinate system as described in Table 7. Measurement units are meter for the cartesian coordinates and radians for the angle coordinates. In polar coordinates or cylindric coordinates a radius with the value r=0 indicates an unused component. An example is the 5.1 format where no radius is specified.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 7 — channelCoordinateSystems table

Value	Coordinate system	ChannelSoundLocation format
0 (default)	Cartesian origin = scene origin (locations relative to the scene)	{x, y, z}
1	Cartesian origin = user position (locations relative to the user)	{x, y, z}
3	Polar origin = scene origin (locations relative to the scene)	{r, azimuth, elevation}
4	Polar origin = user position (locations relative to the user)	{r, azimuth, elevation}
5	Cylindric origin = scene origin (locations relative to the scene)	{r, azimuth, z}
6	Cylindric origin = user position (locations relative to the user)	{r, azimuth, z}
7	reserved for ISO use	

channelSoundLocation determines the location of the sound of the corresponding channel. Each location is a 3-element vector. This field is in MFFloat-format instead of MFVec3f due to its multiple format possibility. Each vector consists of three elements**.**

For example in a 5-channel configuration the second channel can be configured as follows: **channelCoordinateSystems[1]** == 4 ➔ **channelSoundLocation[3:5]** is a 3D-vector in polar coordinates relative to the listener describing the position of the sound of the second channel.

The **channelDirectionalPattern** is an integer vector restricted to the values from Table 8. Each element describes the desired directional pattern of the sound of the corresponding channel.

Table 8 — channelDirectionalPattern association table

Value	Polarity
0	Sphere 0th order (monopole)
1	Sphere 1st order (dipole)
2	Cardioide
3..7	reserved for ISO use

This is useful for example in case of Dolby ProLogic where the front channels have monopole patterns and the surround channel has to have dipole characteristics (**channelDirectionalPattern[0:3]** = [0 0 0 1] for the four channels L, R, C, S).

The **channelDirection** field is valid for values of the **channelDirectionalPattern** > 0. It describes the direction of the **channelDirectionalPattern** in the coordinate system **channelCoordinateSystems** (see Table 7) defined for the corresponding channel.

7.2.2.9.2.2 Ambisonics® channel configuration parameters

These parameters allows interpreting the audio channels as "Ambisonics® components" B_{mn}^{σ}, which represent the spatial encoding/recording of the sound field according to the spherical harmonics Y_{mn}^{s} (see Figure 11).

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

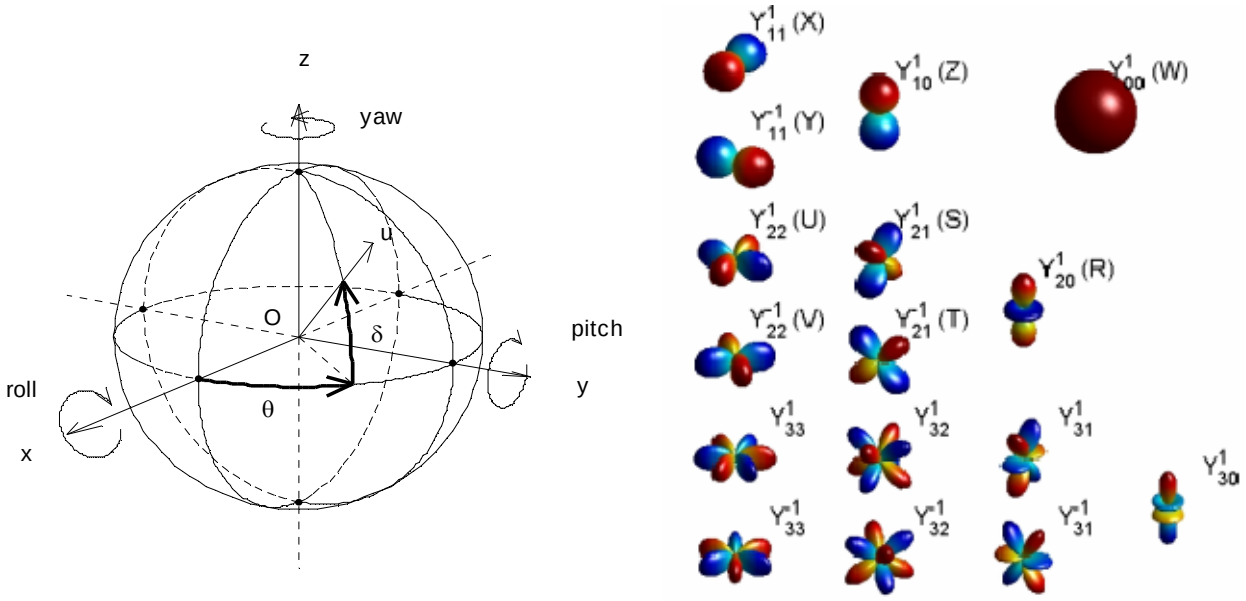


Figure 11 — Left part: spherical coordinate system (radius *r*, azimuth θ, elevation δ), X-axis being considered to be forward pointing while Y-axis is left-hand pointing. Right part: 3D view of spherical harmonics Y_{mn}^{s} with usual designation of associated Ambisonics® components. 0th and 1st order components W, X, Y and Z compose the so-called "B-format"

Each component B_{mn}^{σ} is re-paired in terms of indexes m, n (with $0 \le n \le m$) and $\sigma = \pm 1$. Components with $\sigma = +1$ have an azimuth dependence in cos nθ while components with $\sigma = -1$ have an azimuth dependence in sin nθ. In case of increasing m and n the newly considered components provide a more accurate angular resolution to the 3D sound scene description. Note that for each m there's only one component with n=0 which has no azimuth dependence. Components with n=m are "horizontal" or "2D" components, and are generally privileged over other components since in many cases rendering may be performed on horizontal-only loudspeaker configurations. For convenience Ambisonics® components will be referred to by a single index SID (Table 9) derived by enumerating indexes (m,n,σ) according to this rule: increase m from 0; for each m, decrease n from m to 0; for each n, choose $\sigma = +1$, then $\sigma = -1$ (except if n=0).

Table 9 — Single Index Designation (SID) for Ambisonics® Components

Usual Name	W	X	Y	Z	U	V	No more usual names							
(m,n,σ) designation	0,0 +1	1,1 +1	1,1 -1	1,0 +1	2,2 +1	2,2 -1	…	m,m +1	…	m,n +1	m,n -1	…	m,0 +1	…
Single Index Designation	0	1	2	3	4	5	…	m^2	…	m^2 +2(m-n)	m^2 +2(m−n)+1	…	$(m+1)^2$ -1	…

Further explanation of concepts and parameters of this part can be found in [1].

ambResolution2D: Upper order m of 2D (horizontal only) components (*i.e.* such that m=n).

ambResolution3D: Upper order m of 3D (non-horizontal) components (*i.e.* such that n<m); will be generally less than or equal to **ambResolution2D**.

ambEncodingConvention: The encoding convention characterized by the set of weighting factors applied to the components or spherical harmonics. Possible values are: 'N2D'=0, 'SN2D'=1, 'N3D'=2, 'SN3D'=3, 'MaxN'=4, 'FMH'=5, etc. Conversion formulae are described in details in [1]. Note that the specified convention applies to the Ambisonics® components finally obtained, for example after backward matrixing (or channel recombination) in case of occurance (see comments on **ambBackwardMatrix**).

The intrinsic parameter for Near-Field Compensated Ambisonics® sound fields is the ratio of a reference distance (=**ambNfcReferenceDistance)** and the sound speed (=**ambSoundSpeed)** as measured in the real world. The reference distance refers to the loudspeaker array radius for which the sound field rendering is preferably dedicated to (in the sense that spatial decoding requires no sound field curvature adaptation).

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ambArrangementRule: a value that describes how to interpret the conveyed channels as ambisonic components. Its binary coding can be interpreted as a "flag container". Bit number 0 is interpreted as "explicit ordering" flag: if binary masking of **ambArrangementRule** by 001 yields 001 (flag is true), then **ambComponentIndex** field is valid. Bits number 1 and number 2 are interpreted as respectively "recombination" flag and "explicit recombination" flag: if binary masking by 110 yields 010, **ambRecombinationPreset** field is valid and predefined recombination rules apply; if it yields 110, **ambBackwardMatrix** field is valid and explicit channel recombination applies.

ambRecombinationPreset: describes predefined rules (listed in Table 10) of channel recombination. Channel recombination operates the same way as described with **ambBackwardMatrix** below, but using parameters N and K of Table 10 and processing the first N channels with a predefined conversion rule instead of an explicit matrix.

Table 10 — Particular cases of channels recombination. "Value" is the 3-bit right-shifted value of the ambArrangementRule field. UHJ Format is described in [ref_UHJ]

Value	Special rules for channels recombination	N	K
0	2-channel UHJ: First 2 channels conform to UHJ format	2	2
1	3-channel UHJ: First 3 channels conform to UHJ format	3	3
2	4-channel UHJ: First 4 channels conform to UHJ format	4	4
3..15	reserved for ISO use		

ambComponentIndex: A vector containing the single indexes (as defined in Table 9) that identify Ambisonics® components, as resulting from the channel recombination described below if applied, or as being directly represented by the conveyed channels otherwise.

ambBackwardMatrix: A vector describing the recombination matrix. First 2 elements are the number of lines (K) and the number of columns (N). The other elements are the line-wise listed matrix coefficients $a_{i,j}$ ($1 \le i \le K$, $1 \le j \le N$):

$$\text{ambBackwardMatrix} = \left\lfloor K\ N\ a_{1,1} \ldots a_{1,j} \ldots a_{1,N} \ \ldots a_{i,1} \ldots a_{i,j} \ldots a_{i,N} \ \ldots a_{K,1} \ldots a_{K,j} \ldots a_{K,N} \right\rfloor$$

If not null the length of **ambBackwardMatrix** shall be 2+N*K. The presenter may apply this matrix to the first N elementary channels {C_1 … C_N} in order to extract K Ambisonic components {B_1 … B_K}: $B_i = \sum_{j=1}^{N} a_{i,j} C_j$, which can also be written as:

$$\begin{bmatrix} B_1 \\ \vdots \\ B_i \\ \vdots \\ B_K \end{bmatrix} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,j} & \cdots & a_{1,N} \\ \vdots & \ddots & \vdots & & \vdots \\ a_{i,1} & \cdots & a_{i,j} & \cdots & a_{i,N} \\ \vdots & & \vdots & \ddots & \vdots \\ a_{K,1} & \cdots & a_{K,j} & \cdots & a_{K,N} \end{bmatrix} \begin{bmatrix} C_1 \\ \vdots \\ C_j \\ \vdots \\ C_N \end{bmatrix}$$

Then the list of restored Ambisonic components is completed by the remaining (numChannel-N) conveyed channels {C_{N+1}, … $C_{numChannel}$} that represent "unmatrixed" Ambisonic components {B_{K+1}, … $B_{numChannel-N+K}$}:

$$\begin{bmatrix} B_{K+1} \\ \vdots \\ B_{i'} \\ \vdots \\ B_{\text{numChannel}-N+K} \end{bmatrix} = \begin{bmatrix} C_{N+1} \\ \vdots \\ C_{i'+N-K} \\ \vdots \\ C_{\text{numChannel}} \end{bmatrix}$$

This way, the total number of Ambisonic components is numChannel-N+K.

Remind that **ambComponentIndex**, **ambSoundfieldResolution** and **ambEncodingConvention** fields actually apply to this resulting set of Ambisonic components {B_1 … $B_{numChannel-N+K}$} rather than the conveyed channels {C_1, ... $C_{numChannel}$}.

ambSoundfieldResolution: A vector containing information on the global sound field resolution associated to each component. Indeed when mixing Ambisonic sound fields of different resolutions (as defined by their respective upper order) the content creator may have to provide several versions of the lower order component(s), in order to make

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

possible some optimization of the spatial decoding at the rendering stage. For example: when mixing 1st and 2nd order sound fields, provide two W channels (SID=0), the one being the result of mixing both sound fields, the second being the one of either the 1st or the 2nd order sound field. The **ambSoundfieldResolution** should have the same number of elements as **ambComponentOrdering** and requires **ambOrderingRule**=1 since several components might have the same "Single Index Designation". Each element of **ambSoundfieldResolution**, when positive, refers to the upper sound field resolution the indexed channel belongs to (that means that it's the result of mixing sound fields having spatial resolutions up to the considered value). In the case of negative values each associated absolute value refers to a lower resolution instead of an upper resolution.

7.2.2.10 AudioClip

7.2.2.10.1 Node interface

AudioClip {

exposedField	SFString	**description**	""
exposedField	SFBool	**loop**	FALSE
exposedField	SFFloat	**pitch**	1.0
exposedField	SFTime	**startTime**	0
exposedField	SFTime	**stopTime**	0
exposedField	MFString	**url**	[]
eventOut	SFTime	**duration_changed**	
eventOut	SFBool	**isActive**	

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.10.2 Functionality and semantics

The semantics of the **Audioclip** node are specified in ISO/IEC 14772-1:1998, subclause 6.4.

The **loop**, **startTime**, and **stopTime** exposedFields and the **isActive** eventOut, and their effects on the **AudioClip** node are described in 7.1.1.1.6.2.

The **url** field specifies the data source to be used (see 7.1.1.2.7.1).

The use of this node should be avoided in new applications. It should only be used for VRML backward compatible applications.

7.2.2.11 AudioDelay

7.2.2.11.1 Node interface

AudioDelay {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFTime	**delay**	0
field	SFInt32	**numChan**	1
field	MFInt32	**phaseGroup**	[]

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.11.2 Functionality and semantics

The **AudioDelay** node allows sounds to be started and stopped under temporal control. The start time and stop time of the child sounds are delayed or advanced accordingly.

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

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

The **children** array specifies the nodes affected by the delay. Each child shall be an AudioBIFS node; that is, one of the following: **AudioSource**, **AudioDelay**, **AudioMix**, **AudioSwitch**, **AudioFX**, **AudioClip, AudioChannelConfig, AudioBuffer** or **AdvancedAudioBuffer**.

The **delay** field specifies the delay to apply to each child node.

The **numChan** field specifies the number of channels of audio output by this node.

The **phaseGroup** field specifies the phase relationships among the various output channels; see 7.1.1.1.6.2.

Implementation of the **AudioDelay** node requires the use of a buffer of size *d * S * n,* where *d* is the length of the delay in seconds, *S* is the sampling rate of the node, and *n* is the number of output channels from this node. At scene startup, a multichannel delay line of length *d* and width *n* is initialized to reside in this buffer.

At each time step, the *k * S* audio samples in each channel of the input buffer, where *k* is the length of the system time step in seconds, are inserted into this delay line. If the number of input channels is strictly greater than the number of output channels, the extra input channels are ignored; if the number of input channels is strictly less than the number of output channels, the extra channels of the delay line shall be taken as all 0's.

The output buffer of the node is the *k * S* audio samples which fall off the end of the delay line in this process. Note that this definition holds regardless of the relationship between *k* and *d*.

If the **delay** field is updated during playback, discontinuties (audible artefacts or "clicks") in the output sound may result. If the **delay** field is updated to a greater value than the current value, the delay line is immediately extended to the new length, and zero values inserted at the beginning, so that *d * S* seconds later there will be a short gap in the output of the node. If the **delay** field is updated to a lesser value than the current value, the delay line is immediately shortened to the new length, truncating the values at the end of of the line, so that there is an immediate discontinuity in sound output. Manipulation of the **delay** field in this manner is not recommended unless the audio is muted within the terminal or by appropriate use of an **AudioMix** node at the same time, since it gives rise to impaired sound quality.

7.2.2.12 AudioFX

7.2.2.12.1 Node interface

AudioFX {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
field	SFCommandBuffer	**orch**	""
field	SFCommandBuffer	**score**	""
exposedField	MFFloat	**params**	[]
field	SFInt32	**numChan**	1
field	MFInt32	**phaseGroup**	[]

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.12.2 Functionality and semantics

The **AudioFX** node is used to allow arbitrary signal-processing functions defined using structured audio tools to be included and applied to its children (see ISO/IEC 14496-3 subpart 5, subclause 5.15).

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** array contains the nodes operated upon by this effect. Each child shall be an AudioBIFS node; that is, one of the following: **AudioSource**, **AudioDelay**, **AudioMix**, **AudioSwitch**, **AudioFX**, **AudioClip, AudioChannelConfig**, **AudioBuffer** or **AdvancedAudioBuffer**. If this array is empty, the node has no function (the node may not be used to create new synthetic audio in the middle of a scene graph).

The **orch** command buffer contains a tokenised block of signal-processing code written in SAOL (Structured Audio Orchestra Language). This code block shall contain an orchestra header and some instrument definitions, and conform to the bitstream syntax of the orchestra class as defined in ISO/IEC 14496-3 Subpart 5 subclause 5.5.2.2 and subclause 5.8.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The **score** command buffer may contain a tokenized score for the given orchestra written in SASL (Structured Audio Score Language). This score may contain control operators to adjust the parameters of the orchestra, or even new instrument instantiations. A **score** is not required. If present it shall conform to the bitstream syntax of the `score_file` class as defined in ISO/IEC 14496-3 Subpart 5, subclause 5.5.2 and 5.11.

The **params** field allows BIFS commands and events to affect the sound-generation process in the orchestra. The values of **params** are available to the FX orchestra as the global array `global ksig params[128]`; see ISO/IEC 14496-3, Subpart 5, subclause 5.15.

The **numchan** field specifies the number of channels of audio output by this node.

The **phaseGroup** field specifies the phase relationships among the various output channels; see 7.1.1.1.6.2.

The node is evaluated according to the semantics of the orchestra code contained in the **orch** field. See ISO/IEC 14496-3, subpart 5, for the normative description of this process. Within the orchestra code, the multiple channels of input sound are placed on the global bus, *input_bus*; first, all channels of the first child, then all the channels of the second child, and so on. The orchestra header shall 'send' this bus to an instrument for processing. The **phaseGroup** arrays of the children are made available as the *inGroup* variable within the instrument(s) to which the *input_bus* is sent.

The orchestra code block shall not contain the `spatialize` statement.

The output buffer of this node is the sound produced as the final output of the orchestra applied to the input sounds, as described in ISO/IEC 14496-3, Subpart 5, subclause 5.7.3.

7.2.2.13 AudioFXProto

7.2.2.13.1 PROTO interface

AudioFxProto {
PROTO audio"Name"[

exposedField	MFNode	**audioFXChildren**	[ ]	
exposedField	MFFloat	**audioFXParams**	[ ]	
exposedField	SFInt32	**audioFXnumChannel**	1	
exposedField	MFInt32	**audioFXPhaseGroup**	[ ]	

]
DEF "Name" AudioFX {

eventIn	MFNode	**addChildren**		
eventIn	MFNode	**removeChildren**		
exposedField	MFNode	**children**	[ ]	children IS audioFXChildren
exposedField	SFCommand Buffer	**orch**	[ ]	only used in players with S.A. capability
exposedField	SFCommand Buffer	**score**	[ ]	only used in players with SA capability
exposedField	MFFloat	**params**	[ ]	params[128] IS audioFXParams[128]
Field	SFInt32	**numChannel**	1	numChannel IS audioFXNumChannel
Field	MFInt32	**phaseGroup**	[ ]	phaseGroup IS audioFXPhaseGroup

}
}

7.2.2.13.2 Functionality and semantics

The **AudioFxPROTO** node provides an implementation of a tailored subset of functionality available through the **AudioFX** node. The **AudioFX** node normally requires a Structured Audio implementation. The tailored subset allows players without Structured Audio capability to use these **standard audio effects**. These **standard audio effects** are identified by means of predefined values of the **protoName** field (see 8.7.2.3 PROTOinterfaceDefinition). For compatibility, the PROTO shall encapsulate the **AudioFX** node so enhanced MPEG 4 players with Structured Audio capability can decode the SAOL resp. SASL token streams directly in order to apply the effect. Players that are aware of the predefined

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

protoName values may identify the effects solely based on this name and apply these **standard audio effects** using internal effect representations, if available.

The description of the fields to be used in the **AudioFXProto** can be found in the description of the **AudioFX** node (subclause 7.2.2.12.2, **AudioFX**, Functionality and semantics).

The **protoName** string (see 8.7.2.3 PROTOinterfaceDefinition) shall be set to the appropriate value identified in Table 11. For the corresponding **protoName** strings the JAVA naming convention shall be used.

Table 11 — AudioFXProto names

audioChorus
audioCompressor
audioEcho
audioEqualizer
audioFilter
audioFlange
audioNaturalReverb
audioReverb
audioSpeedChange
audioStereoBase
audioVirtualStereo

The **standard audio effects** to be defined can be seen in Table 11. SpeedChange requires **MediaControl**, **AudioBuffer** or **AdvancedAudioBuffer**.

7.2.2.13.3 The standard audio effects

The Structured Audio effects as described in ISO/IEC 14496-3:2001, subclause 5.9.14, 'Effects' are not normative. If content authors wish to have exactly normative effects, they can be authored using strictly normative core opcodes. This means that an effect (in Structured Audio) is normative, if it uses these opcodes and combines them in a suitable way. There is no definition, how this combination has to be done and if it can injure the normativity, e.g. with the presence of additional code in the calling function. It is assumed that the orchestra adds no further functionality and combines the core opcodes in an "additive" way due to the routing defined in the route and send statements of the orchestra.

The following paragraphs describe the interfaces and the parameters to the above mentioned **standard audio effects**.

7.2.2.13.3.1 PROTO audioChorus

The audioChorus contains the following parameters:

Data type	Function	Default value	Range
float	**rate**	2	0..5
float	**depth**	0.5	0..1
float	**modulation**	1	0..16

The chorus PROTO creates a sound with a corusing effect. Its speed is defined by ***rate*** and its strength can be varied with ***depth***.

rate is specified in cycles per second.

modulation defines the modulation function as listed in Table 12.

Table 12 — audioChorus effect modulation

Value	Modulation function
0	Sine
1	Triangle
2..	reserved for ISO use

depth is specified as excursion with 0<=depth<=1.0.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

rate, ***modulation*** and ***depth*** shall map to the params[ ] array as follows:
rate **=** params [0]
depth = params [1]
modulation = params [2]

7.2.2.13.3.2 PROTO audioCompressor

The audioCompressor contains the following parameters:

Data type	Function	Default value	Range
float	**nfloor**	--	--
float	**threshold**	--	--
float	**loknee**	--	--
float	**hiknee**	--	--
float	**ratio**	--	--
float	**att**	--	--
float	**rel**	--	--
float	**look**	--	--
float	**delay**	--	--

For the description of compressor and parameter ranges see ISO/IEC 14496-3:2001, subclause 5.9.11.4.

nfloor, ***threshold***, ***loknee, hiknee***, ***ratio***, ***att***, ***rel***, ***look*** and ***delay*** shall map to the params[] array as follows :

nfloor = params [0]
threshold = params [1]
loknee = params [2]
hiknee = params [3]
ratio = params [4]
att = params [5]
rel = params [6]
look = params [7]
delay = params [8]

7.2.2.13.3.3 PROTO audioEcho

The audioEcho contains the following parameters:

Data type	Function	Default value	Range
float	**numChannels**	2	0..n
float	**numTaps**	1	0..n
float	**feedback**	25	0..100
float[]	**delayTime**	440	0..2000

The echo PROTO creates a sound with an typical echo effect.

numChannels defines the number of audio channels to be processed.

numTaps defines the number of echoes to be calculated. every tap gets its own delay time from the delayTime-field.

feedback defines the relative level in % of samples inserted into the delay-buffer again after processing.

delayTime[n] defines the time between original signal and echo tap n in milliseconds.

numChannels, ***numTaps***, ***feedback*** and ***delayTime*** shall be mapped to the params[ ] array as follows:

numChannels **=** params [0]
numTaps = params [1]
feedback = params [2]
delayTime = params [3 … numTaps]

7.2.2.13.3.4 PROTO audioEqualizer

The audioEqualizer contains the following parameters:

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2005 – All rights reserved
Not for Resale

Data type	Function	Default value	Range
float	**numFreqBands**	2	0..16
float[ ]	**centerFreq**	[]	0..20000
float[ ]	**bandwidth**	[]	0..20000
float[ ]	**gain**	1	0.1 .. 10

The ***numFreqBands*** value sets the number of frequency bands.
The default value is 2 for a typical shelving filter.

The ***centerFreq*** vector sets the center frequency of each frequency band. If the vector is not defined the effect will be switched off.

The ***bandwidth*** vector sets the bandwidth of each frequency band (see bandwidth definition in ISO/IEC 14496-3:2001, subclause 5.9.9.4). If the vector is not defined the effect will be switched off.

The ***gain*** vector sets the gain per frequency band.

numFreqBands, ***centerFreq***, ***bandwidth*** and ***gain*** shall map to the params[ ] array as follows:
numFreqBands = params [0]
centerFreq [0...numFreqBands-1] = params [1...numFreqBands]
bandwidth [0...numFreqBands-1] = params [numFreqBands+1...2*numFreqBands]
gain [0...numFreqBands1] = params [2*numFreqBands+1...3*numFreqBands]

7.2.2.13.3.5 PROTO audioFilter

The audioFilter contains the following parameters:

Data type	Function	Default value	Range
float	**filtertype**	2	{low, high, pass, stop}
float	**filterorder**	0	0..4
float	**frequency**	2000	0..20000
float	**bandwidth**	20000	0..20000
float	**characteristic**	0	{butterworth, bessel}

The AudioFilter PROTO is used to filter its input signal with a lowpass, highpass, bandpass or bandstop filter characteristic.

filtertype defines the type of the filter as listed in Table 13:

Table 13 — audioFilter filtertype function

Value	Filtertype function
0	lowpass
1	highpass
2	bandpass
3	bandstop
4..	reserved for ISO use

filterorder defines the order of the filter in a range from 0..4. If filterorder is 0, the filter is switched off.

frequency defines the cutoff frequency (***filtertype*** 0 or 1) resp. the center frequency (***filtertype*** 2 or 3).

bandwidth defines the bandwidth in filtertypes 2 or 3. It is not used with in filtertypes 0 or 1.

7.2.2.13.3.6 PROTO audioFlange

The audioFlange contains the following parameters:

Data type	Function	Default value	Range
float	**rate**	2	0..5
float	**depth**	0.5	0..1
float	**modulation**	1	0..16
float	**feedback**	0	0..1

The audioFlange PROTO creates a sound with a flanged effect. Its speed is defined by ***rate*** and its strength can be varied with ***depth***. The sound characteristic can also be changed with ***feedback***.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The ***rate*** is specified in cycles per second.

The ***modulation*** defines the modulation function as listed in Table 14.

Table 14 — audioFlange effect modulation

Value	Modulation function
0	Sine
1	Triangle
2..	reserved for ISO use

The ***depth*** is specified as excursion with 0<=depth<=1.0.

The ***feedback*** is specified as feedback gain factor. 0 means no feedback and 1.0 means maximum feedback at stable operation.

rate, ***modulation***, ***depth*** and ***feedback*** shall map to the params[ ] array as follows:
rate **=** params [0]
modulation = params [1]
depth = params [2]
feedback = params [3]

7.2.2.13.3.7 PROTO audioNaturalReverb

The audioNaturalReverb contains the following parameters:

First params[ ] field:

Data type	Function	Default value	Range
float	numParamsFields	1	1..60000
float	numImpResp	0	0..32
float	sampleRate		
float[ ]	reverbChannels	0	0,1,2,3,...,31
float	impulseResponseCoding	0	0..1
…			reserved

Following params[ ] fields:

Data type	Function	Default value	Range
float	impulseResponseLength	0	240000 *
float[ ]	impulseResponse		*
....			* numImpResp times

The NaturalReverb PROTO uses the impulse responses of different sound channels to create a reverberation effect. Since these impulse responses can be very long (several seconds for a big church or hall), one params[ ] array is not sufficient to transmit the complete data set. Therefore a bulk of consecutive params[ ] arrays is used in the following way:

The first block of params[ ] contains information about the following params[ ] fields:
The numParamsFields field determines the number of following params[ ] fields to be used. The NaturalReverb PROTO has to provide sufficient memory to store these fields.
The numImpResp defines the number of impulse responses (= number of channels used for reverberation). It must be smaller than **audioFXnumChannel** in the AudioFX PROTO node interface.

The **reverbChannels** field defines the mapping of the impulse responses to the input channels. The mapping is the same as for the matrix field in the AudioMix node of the MPEG-4 Systems standard (ISO/IEC 14496-11) whereas the linear factor of the matrix elements from the Audio Mix node (multiply operation) have to be replaced by the impulse responses (convultion operation).

The ***impulseResponseCoding*** field shows how the impulse response is coded (Table 15).

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 15 — audioNaturalReverb effect coding

Value	Coding function
0	consecutive samples
1	sample-number/sample
2..	reserved for ISO use

Coding 1 can be useful to reduce the length of sparse impulse responses.

The fields shall map to the first params[ ] array as follows:

numParamsFields = params [0]
numRevChan = params [1]
sampleRate = params [2]
reverbChannels [0... ***numRevChan*** -1] = params [3...3 + ***numRevChan*** - 1]
impulseResponseCoding = params [3 + ***numRevChan***]

The following params[ ] fields contain the ***numImpResp*** consecutive impulse responses as follows:

The ***impulseResponseLength*** gives the length of the following ***impulseResponse.***

The ***impulseResponseLength*** and the ***impulseResponse*** are repeated ***numImpResp*** times.

The fields shall map to the following params[ ] arrays as follows:

impulseResponseLength = params [0]
impulseResponse = params [1... 1 + ***impulseResponseLength***] ....

The exact method of calculating the reverberation according to the specified parameters is not normative.

The output shall be the reverberated sound signal.

7.2.2.13.3.8 PROTO audioReverb

The audioReverb contains the following parameters:

Data type	Function	Default value	Range
float	preset	2	0..5
float	length	1	0..10
float	numFreqBands	0	0..16
float[ ]	frequencyBand	0	0..20000
float[ ]	reverberation	0	0..10

The ***preset*** value is used for predefined reverb setups. It shall map to the params[ ] array as follows: ***presets*** **=** params [0]

The presets are defined in Table 16.

Table 16 — audioReverb presets

Value	Characteristic
0	none
1	Use **length**, **frequencyBand** and **reverberation**
2	Room
3	Hall
4	Church
5..	reserved for ISO use

If only the length is given it is taken as a full-range reverberation time which is the amount of time delay until the sound amplitude is attenuated 60 dB compared to the source sound (RT60).

length shall map to the params[ ] array as follows:
length **=** params [1]

If ***numFreqBands*** is greater than one and ***frequencyBand*** and ***reverberation*** are given, they represent the responses at different frequencies. At each ***frequencyBand*** the ***reverberation*** is given as the reverberation time (RT60).

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

numFreqBands, ***frequencyBand*** and ***reverberation*** shall map to the params[ ] array as follows:
numFreqBands = params [2]
frequencyBand [0...numFreqBands-1] = params [3...3+numFreqBands-1]
reverberation [0...numFreqBands-1] = params [numFreqBands + 3...3 + 2*numFreqBands - 1]

The exact method of calculating the reverberation according to the specified parameters is not normative.

The output shall be the reverberated sound signal.

7.2.2.13.3.9 PROTO audioSpeedChange

The audioSpeedChange contains the following parameters:

Data type	Function	Default value	Range
float	speedFactor	1	0.1..10

The SpeedChange PROTO changes the Speed of the signal without influence on its pitch. SpeedChange must use an **AudioBuffer** or **AdvancedAudioBuffer** as input.

The ***children*** field is restricted to an **AudioBuffer** or **AdvancedAudioBuffer** node only.

The ***speedFactor*** describes the amount of speed change and shall map to the params[ ] array as follows:

speedFactor = params [0]

7.2.2.13.3.10 PROTO audioStereoBase

The audioSteroBase contains the following parameters:

Data type	Function	Default value	Range
float	stereoBase	1	0..2

The audioStereoBase PROTO is used to manipulate the basewidth of a 2.0 or 2.1 channel signal. The basewidth is defined in as follows:

StereoBase	Basewidth
0	Mono
]0...1[	reduced base
1	Stereo
]1...2	widened base

stereoBase shall map to the params[ ] array as follows:
stereoBase = params [0]

7.2.2.14 AudioMix

7.2.2.14.1 Node interface

AudioMix {

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2005 – All rights reserved
Not for Resale

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFInt32	**numInputs**	1
exposedField	MFFloat	**matrix**	[]
field	SFInt32	**numChan**	1
field	MFInt32	**phaseGroup**	[]

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.14.2 Functionality and semantics

This node is used to mix together several audio signals in a simple, multiplicative way. Any relationship that may be specified in terms of a mixing matrix may be described using this node.

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 specifies which nodes' outputs to mix together. Each child shall be an AudioBIFS node; that is, one of the following: **AudioSource**, **AudioDelay**, **AudioMix**, **AudioSwitch**, **AudioFX**, **AudioClip, AudioChannelConfig**, **AudioBuffer** or **AdvancedAudioBuffer**.

The **numInputs** field specifies the number of input channels. It shall be the sum of the number of channels of the children.

The **matrix** array specifies the mixing matrix which relates the inputs to the outputs. **matrix** is an unrolled **numInputs** x **numChan** matrix which describes the relationship between **numInputs** input channels and **numChan** output channels. The **numInputs** * **numChan** values are in row-major order. That is, the first **numInputs** values are the scaling factors applied to each of the inputs to produce the first output channel; the next **numInputs** values produce the second output channel, and so forth.

That is, if the desired mixing matrix is $\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$, specifying a "2 into 3" mix, the value of the **matrix** field shall be [*a b c d e f*].

The **numchan** field specifies the number of channels of audio output by this node.

The **phaseGroup** field specifies the phase relationships among the various output channels; see 7.1.1.1.6.2.

The value of the output buffer for an **AudioMix** node is calculated as follows. For each sample number *x* of output channel *i*, 1 <= *i* <= **numChan**, the value of that sample is

matrix[(0) * **numChan** + **i**] * input[1][*x*] +

matrix[(1) * **numChan** + **i**] * input[2][*x*] + ...

matrix[(**numInputs** – 1) * **numChan** + **i**] * input[**numInputs**][*x*],

where input[*i*][*j*] represents the *j*th sample of the *i*th channel of the input buffer, and the **matrix** elements are indexed starting from 1.

7.2.2.15 AudioSource

7.2.2.15.1 Node interface

AudioSource {

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	MFString	**url**	[]
exposedField	SFFloat	**pitch**	1.0
exposedField	SFFloat	**speed**	1.0
exposedField	SFTime	**startTime**	0
exposedField	SFTime	**stopTime**	0
field	SFInt32	**numChan**	1
field	MFInt32	**phaseGroup**	[]

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.15.2 Functionality and semantics

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, subpart 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, section 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, Subpart 5, subclause 5.7.3.3.6, list item 8, describe 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, Subpart 5, subclause 5.9.14.4).

The **startTime** and **stopTime** exposedFields and their effects on the **AudioSource** node are described in 7.1.1.1.6.2.

The **numChan** field describes how many channels of audio are in the decoded bitstream.

The **phaseGroup** array specifies whether or not there are important phase relationships between the multiple channels of audio. If there are such relationships – for example, if the sound is a multichannel spatialized set or a “stereo pair” – it is in general dangerous to do anything more complex than scaling to the sound. Further filtering or repeated “spatialization” will destroy these relationships. The values in the array divide the channels of audio into groups; if **phaseGroup[i] = phaseGroup[j]** then channel **i** and channel **j** are phase-related. Channels for which the **phaseGroup** value is 0 are not related to any other channel.

The **url** field specifies the data source to be used (see 7.1.1.2.7.1).

The audio output from the decoder according to the bitstream(s), referenced in the specified URL, at the current scene time is placed in the output buffer for this node, unless the current scene time is earlier than the current value of **startTime** or later than the current value of **stopTime**, in which case 0 values are placed in the output buffer for this node for the current scene time.

For audio sources decoded using the main object of the structured audio decoder (ISO/IEC 14496-3, subpart 5), several variables from the scene description must be mapped into standard names in the orchestra. See ISO/IEC 14496-3, Subpart 5, subclause 5.15 and subclause 5.8.6.8.

If **AudioBuffer** children are provided for a structured audio decoder, the audio data buffered in the **AudioBuffer**(s) must be made available to the decoding process. See subclause ISO/IEC 14496-3, Subpart 5, subclause 5.10.2.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.16 AudioSwitch

7.2.2.16.1 Node interface

AudioSwitch {			
eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	MFInt32	**whichChoice**	[]
field	SFInt32	**numChan**	1
field	MFInt32	**phaseGroup**	[]
}			

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.16.2 Functionality and semantics

The **AudioSwitch** node is used to select a subset of audio channels from the child nodes specified.

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 specifies a list of child options. Each child shall be an AudioBIFS node; that is, one of the following: **AudioSource**, **AudioDelay**, **AudioMix**, **AudioSwitch**, **AudioFX**, **AudioClip,** **AudioChannelConfig**, **AudioBuffer** or **AdvancedAudioBuffer**.

The **whichChoice** field specifies which channels shall be passed through. If **whichChoice[i]** is 1, then the i-th child channel shall be passed through.

The **numchan** field specifies the number of channels of audio output by this node; ie, the number of channels in the passed child.

The **phaseGroup** field specifies the phase relationships among the various output channels; see subclause 7.1.1.1.6.2.

The values for the output buffer are calculated as follows:

For each sample number *x* of channel number *i* of the output buffer, 1 <= *i* <= **numChan**, the value in the buffer is the same as the value of sample number *x* in the *j*th channel of the input, where *j* is the least value such that **whichChoice[0] + whichChoice[1] + ... + whichChoice[*j*] =** *i*.

7.2.2.17 Background

7.2.2.17.1 Node interface

Background {			
eventIn	SFBool	**set_bind**	
exposedField	MFFloat	**groundAngle**	[]
exposedField	MFColor	**groundColor**	[]
exposedField	MFString	**backURL**	[]
exposedField	MFString	**bottomURL**	[]
exposedField	MFString	**frontURL**	[]
exposedField	MFString	**leftURL**	[]
exposedField	MFString	**rightURL**	[]
exposedField	MFString	**topURL**	[]
exposedField	MFFloat	**skyAngle**	[]
exposedField	MFColor	**skyColor**	0, 0, 0
eventOut	SFBool	**isBound**	
}			

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.17.2 Functionality and semantics

The semantics of the **Background** node are specified in ISO/IEC 14772-1:1998, subclause 6.5.

The **backUrl**, **bottomURL**, **frontUrl**, **leftUrl**, **rightUrl**, **topUrl** fields specify the data sources to be used (see 7.1.1.2.7.1).

7.2.2.18 Background2D

7.2.2.18.1 Node interface

Background2D {			
eventIn	SFBool	**set_bind**	
exposedField	SFColor	**backColor**	0 0 0
exposedField	MFString	**url**	[]
eventOut	SFBool	**isBound**	
}			

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.18.2 Functionality and semantics

There exists a **Background2D** stack, in which the top-most background is the current active background one. The **Background2D** node allows a background to be displayed behind a 2D scene. The functionality of this node can also be accomplished using other nodes, but use of this node may be more efficient in some implementations.

If **set_bind** is set to TRUE the **Background2D** is moved to the top of the stack.If **set_bind** is set to FALSE, the **Background2D** is removed from the stack so the previous background which is contained in the stack is on top again.

The **isBound** event is sent as soon as the backdrop is put at the top of the stack, so becoming the current backdrop.

The **url** field specifies the data source to be used (see 7.1.1.2.7.1).

The **backColor** field specifies a colour to be used as the background.

This is not a geometry node. The top-left corner of the image is mapped to the top-left corner of the **Layer2D** and the right-bottom corner of the image is stretched to the right-bottom corner of the **Layer2D**, regardless of the current transformation. Scaling and/or rotation do not have any effect on this node. The background image will always exactly fill the entire **Layer2D**, regardless of **Layer2D** size, without tiling or cropping.

When a **Background2D** node is included in a 3D context, that is in a **Group**, **Layer3D**, or **CompositeTexture3D** node, then it shall be rendered behind all other geometries and be scaled to fit in the enclosing frame. For Group node, this frame is the whole scene. For **Layer3D** and **CompositeTexture3D** the background image is scaled to fit in the frame of the node.

EXAMPLE — Changing the background for 5 seconds.

```
Group {
   children [
      …
      DEF TIS TimeSensor {
         startTime 5.0
         stopTime  10.0
      }
      DEF BG1 Background2D {
         …
      }
   ]
}
ROUTE TIS.isActive TO BG1.set_bind
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

7.2.2.19 BAP

7.2.2.19.1 Node interface

BAP {

exposedField	SFInt32	**sacroiliac_tilt**	+I
exposedField	SFInt32	**sacroiliac_torsion**	+I
exposedField	SFInt32	**sacroiliac_roll**	+I
exposedField	SFInt32	**l_hip_flexion**	+I
exposedField	SFInt32	**r_hip_flexion**	+I
exposedField	SFInt32	**l_hip_abduct**	+I
exposedField	SFInt32	**r_hip_abduct**	+I
exposedField	SFInt32	**l_hip_twisting**	+I
exposedField	SFInt32	**r_hip_twisting**	+I
exposedField	SFInt32	**l_knee_flexion**	+I
exposedField	SFInt32	**r_knee_flexion**	+I
exposedField	SFInt32	**l_knee_twisting**	+I
exposedField	SFInt32	**r_knee_twisting**	+I
exposedField	SFInt32	**l_ankle_flexion**	+I
exposedField	SFInt32	**r_ankle_flexion**	+I
exposedField	SFInt32	**l_ankle_twisting**	+I
exposedField	SFInt32	**r_ankle_twisting**	+I
exposedField	SFInt32	**l_subtalar_flexion**	+I
exposedField	SFInt32	**r_subtalar_flexion**	+I
exposedField	SFInt32	**l_midtarsal_flexion**	+I
exposedField	SFInt32	**r_midtarsal_flexion**	+I
exposedField	SFInt32	**l_metatarsal_flexion**	+I
exposedField	SFInt32	**r_metatarsal_flexion**	+I
exposedField	SFInt32	**l_sternoclavicular_abduct**	+I
exposedField	SFInt32	**r_sternoclavicular_abduct**	+I
exposedField	SFInt32	**l_sternoclavicular_rotate**	+I
exposedField	SFInt32	**r_sternoclavicular_rotate**	+I
exposedField	SFInt32	**l_acromioclavicular_abduct**	+I
exposedField	SFInt32	**r_acromioclavicular_abduct**	+I
exposedField	SFInt32	**l_acromioclavicular_rotate**	+I
exposedField	SFInt32	**r_acromioclavicular_rotate**	+I
exposedField	SFInt32	**l_shoulder_flexion**	+I
exposedField	SFInt32	**r_shoulder_flexion**	+I
exposedField	SFInt32	**l_shoulder_abduct**	+I
exposedField	SFInt32	**r_shoulder_abduct**	+I
exposedField	SFInt32	**l_shoulder_twisting**	+I
exposedField	SFInt32	**r_shoulder_twisting**	+I
exposedField	SFInt32	**l_elbow_flexion**	+I
exposedField	SFInt32	**r_elbow_flexion**	+I
exposedField	SFInt32	**l_elbow_twisting**	+I
exposedField	SFInt32	**r_elbow_twisting**	+I
exposedField	SFInt32	**l_wrist_flexion**	+I
exposedField	SFInt32	**r_wrist_flexion**	+I
exposedField	SFInt32	**l_wrist_pivot**	+I
exposedField	SFInt32	**r_wrist_pivot**	+I
exposedField	SFInt32	**l_wrist_twisting**	+I
exposedField	SFInt32	**r_wrist_twisting**	+I
exposedField	SFInt32	**skullbase_roll**	+I
exposedField	SFInt32	**skullbase_torsion**	+I
exposedField	SFInt32	**skullbase_tilt**	+I
exposedField	SFInt32	**vc1roll**	+I
exposedField	SFInt32	**vc1torsion**	+I
exposedField	SFInt32	**vc1tilt**	+I
exposedField	SFInt32	**vc2roll**	+I
exposedField	SFInt32	**vc2torsion**	+I

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

exposedField	SFInt32	**vc2tilt**	+I
exposedField	SFInt32	**vc3roll**	+I
exposedField	SFInt32	**vc3torsion**	+I
exposedField	SFInt32	**vc3tilt**	+I
exposedField	SFInt32	**vc4roll**	+I
exposedField	SFInt32	**vc4torsion**	+I
exposedField	SFInt32	**vc4tilt**	+I
exposedField	SFInt32	**vc5roll**	+I
exposedField	SFInt32	**vc5torsion**	+I
exposedField	SFInt32	**vc5tilt**	+I
exposedField	SFInt32	**vc6roll**	+I
exposedField	SFInt32	**vc6torsion**	+I
exposedField	SFInt32	**vc6tilt**	+I
exposedField	SFInt32	**vc7roll**	+I
exposedField	SFInt32	**vc7torsion**	+I
exposedField	SFInt32	**vc7tilt**	+I
exposedField	SFInt32	**vt1roll**	+I
exposedField	SFInt32	**vt1torsion**	+I
exposedField	SFInt32	**vt1tilt**	+I
exposedField	SFInt32	**vt2roll**	+I
exposedField	SFInt32	**vt2torsion**	+I
exposedField	SFInt32	**vt2tilt**	+I
exposedField	SFInt32	**vt3roll**	+I
exposedField	SFInt32	**vt3torsion**	+I
exposedField	SFInt32	**vt3tilt**	+I
exposedField	SFInt32	**vt4roll**	+I
exposedField	SFInt32	**vt4torsion**	+I
exposedField	SFInt32	**vt4tilt**	+I
exposedField	SFInt32	**vt5roll**	+I
exposedField	SFInt32	**vt5torsion**	+I
exposedField	SFInt32	**vt5tilt**	+I
exposedField	SFInt32	**vt6roll**	+I
exposedField	SFInt32	**vt6torsion**	+I
exposedField	SFInt32	**vt6tilt**	+I
exposedField	SFInt32	**vt7roll**	+I
exposedField	SFInt32	**vt7torsion**	+I
exposedField	SFInt32	**vt7tilt**	+I
exposedField	SFInt32	**vt8roll**	+I
exposedField	SFInt32	**vt8torsion**	+I
exposedField	SFInt32	**vt8tilt**	+I
exposedField	SFInt32	**vt9roll**	+I
exposedField	SFInt32	**vt9torsion**	+I
exposedField	SFInt32	**vt9tilt**	+I
exposedField	SFInt32	**vt10roll**	+I
exposedField	SFInt32	**vt10torsion**	+I
exposedField	SFInt32	**vt10tilt**	+I
exposedField	SFInt32	**vt11roll**	+I
exposedField	SFInt32	**vt11torsion**	+I
exposedField	SFInt32	**vt11tilt**	+I
exposedField	SFInt32	**vt12roll**	+I
exposedField	SFInt32	**vt12torsion**	+I
exposedField	SFInt32	**vt12tilt**	+I
exposedField	SFInt32	**vl1roll**	+I
exposedField	SFInt32	**vl1torsion**	+I
exposedField	SFInt32	**vl1tilt**	+I
exposedField	SFInt32	**vl2roll**	+I
exposedField	SFInt32	**vl2torsion**	+I
exposedField	SFInt32	**vl2tilt**	+I
exposedField	SFInt32	**vl3roll**	+I
exposedField	SFInt32	**vl3torsion**	+I

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

exposedField	SFInt32	**vl3tilt**	+|
exposedField	SFInt32	**vl4roll**	+|
exposedField	SFInt32	**vl4torsion**	+|
exposedField	SFInt32	**vl4tilt**	+|
exposedField	SFInt32	**vl5roll**	+|
exposedField	SFInt32	**vl5torsion**	+|
exposedField	SFInt32	**vl5tilt**	+|
exposedField	SFInt32	**l_pinky0_flexion**	+|
exposedField	SFInt32	**r_pinky0_flexion**	+|
exposedField	SFInt32	**l_pinky1_flexion**	+|
exposedField	SFInt32	**r_pinky1_flexion**	+|
exposedField	SFInt32	**l_pinky1_pivot**	+|
exposedField	SFInt32	**r_pinky1_pivot**	+|
exposedField	SFInt32	**l_pinky1_twisting**	+|
exposedField	SFInt32	**r_pinky1_twisting**	+|
exposedField	SFInt32	**l_pinky2_flexion**	+|
exposedField	SFInt32	**r_pinky2_flexion**	+|
exposedField	SFInt32	**l_pinky3_flexion**	+|
exposedField	SFInt32	**r_pinky3_flexion**	+|
exposedField	SFInt32	**l_ring0_flexion**	+|
exposedField	SFInt32	**r_ring0_flexion**	+|
exposedField	SFInt32	**l_ring1_flexion**	+|
exposedField	SFInt32	**r_ring1_flexion**	+|
exposedField	SFInt32	**l_ring1_pivot**	+|
exposedField	SFInt32	**r_ring1_pivot**	+|
exposedField	SFInt32	**l_ring1_twisting**	+|
exposedField	SFInt32	**r_ring1_twisting**	+|
exposedField	SFInt32	**l_ring2_flexion**	+|
exposedField	SFInt32	**r_ring2_flexion**	+|
exposedField	SFInt32	**l_ring3_flexion**	+|
exposedField	SFInt32	**r_ring3_flexion**	+|
exposedField	SFInt32	**l_middle0_flexion**	+|
exposedField	SFInt32	**r_middle0_flexion**	+|
exposedField	SFInt32	**l_middle1_flexion**	+|
exposedField	SFInt32	**r_middle1_flexion**	+|
exposedField	SFInt32	**l_middle1_pivot**	+|
exposedField	SFInt32	**r_middle1_pivot**	+|
exposedField	SFInt32	**l_middle1_twisting**	+|
exposedField	SFInt32	**r_middle1_twisting**	+|
exposedField	SFInt32	**l_middle2_flexion**	+|
exposedField	SFInt32	**r_middle2_flexion**	+|
exposedField	SFInt32	**l_middle3_flexion**	+|
exposedField	SFInt32	**r_middle3_flexion**	+|
exposedField	SFInt32	**l_index0_flexion**	+|
exposedField	SFInt32	**r_index0_flexion**	+|
exposedField	SFInt32	**l_index1_flexion**	+|
exposedField	SFInt32	**r_index1_flexion**	+|
exposedField	SFInt32	**l_index1_pivot**	+|
exposedField	SFInt32	**r_index1_pivot**	+|
exposedField	SFInt32	**l_index1_twisting**	+|
exposedField	SFInt32	**r_index1_twisting**	+|
exposedField	SFInt32	**l_index2_flexion**	+|
exposedField	SFInt32	**r_index2_flexion**	+|
exposedField	SFInt32	**l_index3_flexion**	+|
exposedField	SFInt32	**r_index3_flexion**	+|
exposedField	SFInt32	**l_thumb1_flexion**	+|
exposedField	SFInt32	**r_thumb1_flexion**	+|
exposedField	SFInt32	**l_thumb1_pivot**	+|
exposedField	SFInt32	**r_thumb1_pivot**	+|
exposedField	SFInt32	**l_thumb1_twisting**	+|

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

exposedField	SFInt32	**r_thumb1_twisting**	+I
exposedField	SFInt32	**l_thumb2_flexion**	+I
exposedField	SFInt32	**r_thumb2_flexion**	+I
exposedField	SFInt32	**l_thumb3_flexion**	+I
exposedField	SFInt32	**r_thumb3_flexion**	+I
exposedField	SFInt32	**humanoidRoot_tr_vertical**	+I
exposedField	SFInt32	**humanoidRoot_tr_lateral**	+I
exposedField	SFInt32	**humanoidRoot_tr_frontal**	+I
exposedField	SFInt32	**humanoidRoot_rt_body_turn**	+I
exposedField	SFInt32	**humanoidRoot_rt_body_roll**	+I
exposedField	SFInt32	**humanoidRoot_rt_body_tilt**	+I
exposedField	SFInt32	**extensionBap187**	+I
exposedField	SFInt32	**extensionBap188**	+I
exposedField	SFInt32	**extensionBap189**	+I
exposedField	SFInt32	**extensionBap190**	+I
exposedField	SFInt32	**extensionBap191**	+I
exposedField	SFInt32	**extensionBap192**	+I
exposedField	SFInt32	**extensionBap193**	+I
exposedField	SFInt32	**extensionBap194**	+I
exposedField	SFInt32	**extensionBap195**	+I
exposedField	SFInt32	**extensionBap196**	+I
exposedField	SFInt32	**extensionBap197**	+I
exposedField	SFInt32	**extensionBap198**	+I
exposedField	SFInt32	**extensionBap199**	+I
exposedField	SFInt32	**extensionBap200**	+I
exposedField	SFInt32	**extensionBap201**	+I
exposedField	SFInt32	**extensionBap202**	+I
exposedField	SFInt32	**extensionBap203**	+I
exposedField	SFInt32	**extensionBap204**	+I
exposedField	SFInt32	**extensionBap205**	+I
exposedField	SFInt32	**extensionBap206**	+I
exposedField	SFInt32	**extensionBap207**	+I
exposedField	SFInt32	**extensionBap208**	+I
exposedField	SFInt32	**extensionBap209**	+I
exposedField	SFInt32	**extensionBap210**	+I
exposedField	SFInt32	**extensionBap211**	+I
exposedField	SFInt32	**extensionBap212**	+I
exposedField	SFInt32	**extensionBap213**	+I
exposedField	SFInt32	**extensionBap214**	+I
exposedField	SFInt32	**extensionBap215**	+I
exposedField	SFInt32	**extensionBap216**	+I
exposedField	SFInt32	**extensionBap217**	+I
exposedField	SFInt32	**extensionBap218**	+I
exposedField	SFInt32	**extensionBap219**	+I
exposedField	SFInt32	**extensionBap220**	+I
exposedField	SFInt32	**extensionBap221**	+I
exposedField	SFInt32	**extensionBap222**	+I
exposedField	SFInt32	**extensionBap223**	+I
exposedField	SFInt32	**extensionBap224**	+I
exposedField	SFInt32	**extensionBap225**	+I
exposedField	SFInt32	**extensionBap226**	+I
exposedField	SFInt32	**extensionBap227**	+I
exposedField	SFInt32	**extensionBap228**	+I
exposedField	SFInt32	**extensionBap229**	+I
exposedField	SFInt32	**extensionBap230**	+I
exposedField	SFInt32	**extensionBap231**	+I
exposedField	SFInt32	**extensionBap232**	+I
exposedField	SFInt32	**extensionBap233**	+I
exposedField	SFInt32	**extensionBap234**	+I
exposedField	SFInt32	**extensionBap235**	+I

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

exposedField	SFInt32	**extensionBap236**	+I
exposedField	SFInt32	**extensionBap237**	+I
exposedField	SFInt32	**extensionBap238**	+I
exposedField	SFInt32	**extensionBap239**	+I
exposedField	SFInt32	**extensionBap240**	+I
exposedField	SFInt32	**extensionBap241**	+I
exposedField	SFInt32	**extensionBap242**	+I
exposedField	SFInt32	**extensionBap243**	+I
exposedField	SFInt32	**extensionBap244**	+I
exposedField	SFInt32	**extensionBap245**	+I
exposedField	SFInt32	**extensionBap246**	+I
exposedField	SFInt32	**extensionBap247**	+I
exposedField	SFInt32	**extensionBap248**	+I
exposedField	SFInt32	**extensionBap249**	+I
exposedField	SFInt32	**extensionBap250**	+I
exposedField	SFInt32	**extensionBap251**	+I
exposedField	SFInt32	**extensionBap252**	+I
exposedField	SFInt32	**extensionBap253**	+I
exposedField	SFInt32	**extensionBap254**	+I
exposedField	SFInt32	**extensionBap255**	+I
exposedField	SFInt32	**extensionBap256**	+I
exposedField	SFInt32	**extensionBap257**	+I
exposedField	SFInt32	**extensionBap258**	+I
exposedField	SFInt32	**extensionBap259**	+I
exposedField	SFInt32	**extensionBap260**	+I
exposedField	SFInt32	**extensionBap261**	+I
exposedField	SFInt32	**extensionBap262**	+I
exposedField	SFInt32	**extensionBap263**	+I
exposedField	SFInt32	**extensionBap264**	+I
exposedField	SFInt32	**extensionBap265**	+I
exposedField	SFInt32	**extensionBap266**	+I
exposedField	SFInt32	**extensionBap267**	+I
exposedField	SFInt32	**extensionBap268**	+I
exposedField	SFInt32	**extensionBap269**	+I
exposedField	SFInt32	**extensionBap270**	+I
exposedField	SFInt32	**extensionBap271**	+I
exposedField	SFInt32	**extensionBap272**	+I
exposedField	SFInt32	**extensionBap273**	+I
exposedField	SFInt32	**extensionBap274**	+I
exposedField	SFInt32	**extensionBap275**	+I
exposedField	SFInt32	**extensionBap276**	+I
exposedField	SFInt32	**extensionBap277**	+I
exposedField	SFInt32	**extensionBap278**	+I
exposedField	SFInt32	**extensionBap279**	+I
exposedField	SFInt32	**extensionBap280**	+I
exposedField	SFInt32	**extensionBap281**	+I
exposedField	SFInt32	**extensionBap282**	+I
exposedField	SFInt32	**extensionBap283**	+I
exposedField	SFInt32	**extensionBap284**	+I
exposedField	SFInt32	**extensionBap285**	+I
exposedField	SFInt32	**extensionBap286**	+I
exposedField	SFInt32	**extensionBap287**	+I
exposedField	SFInt32	**extensionBap288**	+I
exposedField	SFInt32	**extensionBap289**	+I
exposedField	SFInt32	**extensionBap290**	+I
exposedField	SFInt32	**extensionBap291**	+I
exposedField	SFInt32	**extensionBap292**	+I
exposedField	SFInt32	**extensionBap293**	+I
exposedField	SFInt32	**extensionBap294**	+I
exposedField	SFInt32	**extensionBap295**	+I

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    exposedField        SFInt32        extensionBap296                +I
}
```

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.19.2 Functionality and semantics

BAP defines the current look of the body by means of body animation parameters. The semantics of the fields of **BAP** is described in Annex C of ISO/IEC 14496-2: 2004.

7.2.2.20 BDP

7.2.2.20.1 Semantic Table

```
BDP {
    exposedField    MFNode        bodySceneGraph                 []
    exposedField    MFNode        bodyDefTables                  []
    exposedField    MFNode        bodySegmentConnectionHint      []
}
```

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.20.2 Funcionality and semantics

The **BDP** node is used to customize the proprietary body model of the decoder to a particular body, or to download a body model along with the information of how to animate it. The Body Definition Parameters (BDPs) are normally transmitted once per session, followed by a stream of coded Body Animation Parameters (BAPs). It is also possible to transmit BDPs more than once per session. If the decoder does not receive the BDPs, the use of a default model ensures that it can still interpret the FBA stream containing BAPs. This insures minimal operation in broadcast or teleconferencing applications.

BDPs specify the following properties:

1. Body surface geometry (with texture coordinates if texture is used)

- The body surface geometry is downloaded using the BIFS stream. The body geometry surfaces are specified using the BIFS **Segment** PROTO definitions as defined In subclause 8.9.

2. Joint center locations.

- The positions of the joints are specified using the BIFS **Joint** PROTO definitions.

- Texture images as part of the BIFS **Segment** definitions.

3. Deformation tables, that describe how to deform the body surfaces using the received BAPs.

The scene graph or a body definition is strongly based on ISO/IEC 14772-1 Amendment 1. The texture images can be defined for each surface. Note that the texture images are part of the PROTO SEGMENT geometry, defined in 8.9.

The following are the basic assumptions about BDP:

1. Default posture to initialize a human body model.

- Standing posture: This posture is defined as follows: the feet should point to the front direction, the two arms should be placed on the side of the body with the palm of the hands facing inward. This posture also implies that all BAPs have value zero (see ISO/IEC 14496-2:2004).

2. Establishing the coordinate system.

The origin of the body coordinate system is located at ground (y=0) level, between the humanoid's feet, with the lateral and frontal position the same as spine origin (l5tilt). The orientation of the coordinate is x points to the left, y points up, and z points to the front of the humanoid. The BDP node defines the body model to be used at the receiver. Two options are supported:

- The **bodyDefTables** is [], the body scene graph is downloaded, in which case the proprietary body of the decoder has to be replaced by the downloaded graph. The **bodySceneGraph** field has to be in the syntax described in 8.9.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- The **bodyDefTables** is different from [], in which case the decoder has to replace its local model by the downloaded graph. The **bodySceneGraph** field has to be in the format, as described below. The **bodyDefTables** field defines how the IndexedFaceSet child of **bodySceneGraph** Segment Node is modified based on sets of BAPs. By means of **bodyDefTables**, the skin or clothes surface geometry of the model can be deformed. The **bodyDefTables** field is defined below.

bodyDefTables defines the behavior of the deformation of the body based on BAP values. See 7.2.2.25.

bodySceneGraph defines the joint center, default geometry, and texture of the body. See 8.9.

bodySegmentConnectionHint contains a **BodySegmentConnectionHint** node (see 7.2.2.26).

7.2.2.21 Billboard

7.2.2.21.1 Node interface

```
Billboard {
   eventIn       MFNode     addChildren
   eventIn       MFNode     removeChildren
   exposedField  SFVec3f    axisOfRotation      0, 1, 0
   exposedField  MFNode     children            []
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.21.2 Functionality and semantics

The semantics of the **Billboard** node are specified in ISO/IEC 14772-1:1998, subclause 6.6. ISO/IEC 14496-1 does not support the bounding box parameters (**bboxCenter** and **bboxSize**).

7.2.2.22 Bitmap

7.2.2.22.1 Node interface

```
Bitmap {
   exposedField  SFVec2f    scale               -1, -1
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.22.2 Functionality and semantics

Bitmap is a geometry node centered at (0,0) in the local coordinate system, to be placed in the geometry field of a **Shape** node. It is a screen-aligned rectangle, which means that the surface normal of this rectangle will always be in the same direction as the screen surface normal, namely straight out to the viewer. It is for example not possible to view the **Bitmap** under an angle from the side. **Bitmap** has the dimensions of the texture that is mapped onto it, as specified in the **Appearance** node of its parent **Shape** node. However, the effective geometry of **Bitmap** is defined by the non-transparent pixels of the image or video that is mapped onto it. When no scaling is specified, a trivial texture-mapping (pixel copying) is performed.

The **scale** field specifies a scaling of the geometry in the x and y dimensions, respectively. The **scale** values shall be strictly positive or equal to -1. A **scale** value of -1 indicates that no scaling shall be applied in the relevant dimension. The special case where both scale dimensions are -1 indicates that the natural dimensions of the texture that is mapped onto the **Bitmap** shall be used.

Bitmap shall not be rotated but may be subject to translation.

Geometry sensors shall respond to the effective geometry of the **Bitmap**, which is defined by the non-transparent pixels of the texture that is mapped onto it.

If a **Material** or **Material2D** node is specified in the **appearance** of the parent **Shape** of a **Bitmap**, the final transparency of each pixel is given by the **Material** or **Material2D** transparency multiplied by the transparency (1-alpha) value of each pixel of the texture. If the texture has no alpha plane then the final transparency of each pixel is purely given by **Material** or **Material2D** transparency (as if the texture had an alpha value of 0).

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Example — To specify semi-transparent video:

```
Shape {
   appearance Appearance {
      texture MovieTexture {  // Visual object
         …
      }
      material Material2D {
         transparency  0.5   // semi-transparent
      }
   }
   geometry Bitmap {}
}
```

7.2.2.23 BitWrapper

7.2.2.23.1 Node interface

BitWrapper {			
field	SFNode	**node**	NULL
field	SFInt32	**type**	0
field	MFString	**url**	[]
field	SFString	**buffer**	“”
}			

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.23.2 Functionality and semantics

A node may have a dedicated node compression scheme. This compressed representation may be carried in the BIFS stream or in a separate stream.

The **node** field contains the node that has a compressed representation. The BitWrapper node can be used in lieu and place of the **node** it wraps. The **type** field indicates which node compression scheme must be used, 0 being the default. It is envisioned that future node compression schemes may be developed for the same node. For this specification, AFX object code table of ISO/IEC 14496-1 defines the default schemes.

The compressed representation is carried either in a separate stream or within the scene stream. The **url** field indicates the stream that contains the compressed representation and the **buffer** field contains the compressed representation when carried within the scene.

When the compressed representation is carried in separate streams by using **url** field, node decoders must be configured. In the object descriptor stream, a node decoder is indicated in the DecoderConfig descriptor for streamType 0x03, objectTypeIndication 0x05, and code defined in AFX object code table of ISO/IEC 14496-1. The decoder is configured with a AFXConfig descriptor

Note that **buffer** is an array of 8-bit values. It shall not be interpreted as a UTF-8 string. For in-band scenario, compressed media stream is transmitted within a scene description stream through **buffer** field as shown in Figure 12 (a). For out-band scenario, compressed media stream is transmitted outside scene description stream through **url** field in Figure 12 (b). It is used when the specific node requires upstream to send a specific information to a server.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

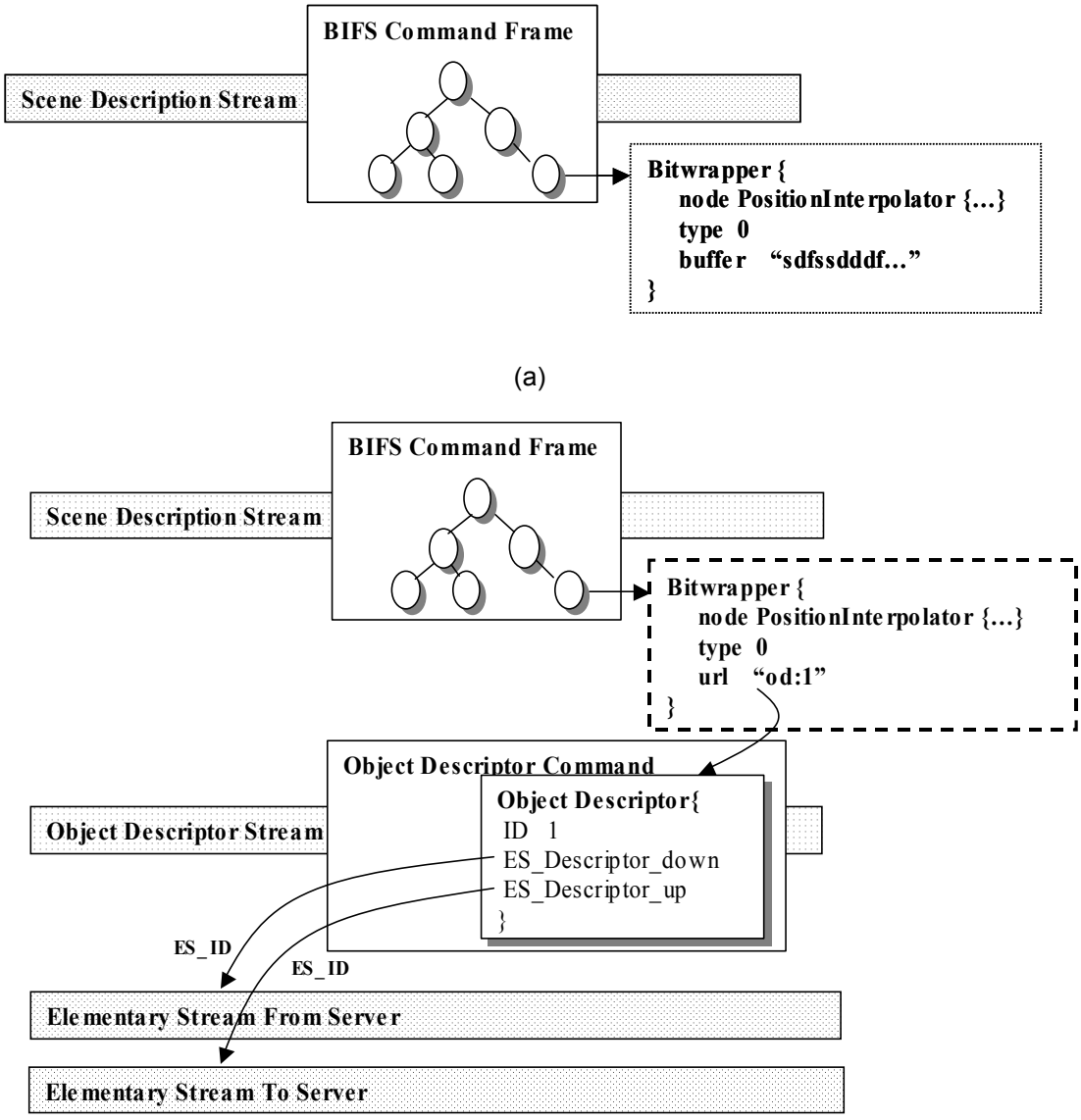


Figure 12 — Two scenarios of the carriage of the node associated bitstream by a generic BIFS stream using BitWapper; (a) in-band case, (b) out-band case

EXAMPLE 1 BitWrapper for behavioral nodes

```
# Stream 10 contains the compressed representation of MyInterp's PositionInterpolator node.

BitWrapper {
   node DEF MyInterp PositionInterpolator {}
   url  "od:10"
}
```

EXAMPLE 2 BitWrapper used in lieu and place as the node it wraps, e.g. a geometry node here.

```
Shape {
   geometry BitWrapper {
      node MeshGrid { … }
      url "od:10"
   }
}
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLE 3 Separation of concerns: BitWrapper updates a node defined in the scene.

```
Shape {
   geometry DEF MyNode MeshGrid { … }
}

…

BitWrapper {
      node USE MyNode
      url "od:10"
}
```

7.2.2.24 Body

7.2.2.24.1 Node interface

Body {			
exposedField	SFNode	**bdp**	NULL
exposedField	SFNode	**bap**	NULL
exposedField	MFNode	renderedBody	[]
}			

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.24.2 Functionality and semantics

The **Body** node organizes definition and animation of a body. The **bap** field shall be always specified. Defining the particular look of a body by means of downloading the position of joint centers or an entire model is optional. If the **bdp** field is NULL, i.e., the BDP node is not specified, the default body model of the decoder is used.

bdp contains a **BDP** node.

bap contains a **BAP** node.

renderedBody is the scene graph of the body after it is rendered (all BAP parameters are applied).

If the **bdp** field of the **Body** node is [] and the **Body** node is a child of a **Group** node that only has one **Face** and one **Body** node, then the **Body** node is associated to that **Face** node.

7.2.2.25 BodyDefTable

7.2.2.25.1 Node interface

BodyDefTable {			
exposedField	SFString	**bodySceneGraphNodeName**	NULL
exposedField	MFInt32	**bapIDs**	[]
exposedField	MFInt32	**vertexIds**	[]
exposedField	MFInt32	**bapCombinations**	[]
exposedField	MFVec3f	**displacements**	[]
exposedField	SFInt32	**numInterpolateKeys**	2
}			

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.25.2 Functionality and semantics

Defines the behavior of body animation parameters (BAPs) on a downloaded bodySceneGraph by specifying displacement vectors of moved vertices inside **IndexedFaceSet** objects as a function of a combination of BAPs. The listed vertices typically represent the deformable body skin surface, or clothe animation for the body.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The **BodyDefTable** node is transmitted directly after the BIFS bitstream of the BDP node. There is no limit on the number of **BodyDefTable** nodes transmitted for one body. A vertex can be listed on more than one **BodyDefTable** nodes. A BAP can be listed on more than one **BodyDefTable** nodes. In this case, the displacements of the same vertex from various BodyDefTable nodes are added to obtain resulting displacement.

Each **BodyDefTable** node contains a list of BAPs, and a list of vertices in the **bodySceneGraph** that are normally affected by these BAPs (for example, the upper and lower arm skin vertices are affected by the elbow joints).

Detailed semantics:

Contains a **BodySegmentConnectionHint** node contains the name of the segment containing an IndexedFaceSet node for which the deformation is defined. This node shall be part of the bodySceneGraph as defined in the BDP node. This node will be contained in the children field of the Segment node.

bapIDs contains the BAP indices, for which the deformation behavior is defined in the bodySceneGraphNodeName field. (Any number of BAPs can be listed in this field). The BAP Ids are defined in the Visual FPDAM1. The values between [1-186] denote the standard BAPs, the values [187-296] denote the user-defined. Other values are undefined.

vertexIDs contains a list of indices into the Coordinate node of the IndexedFaceSet node specified by the child of node with name bodySceneGraphNodeName.

bapCombinations contains a list of interval borders for BAP values, i.e. a list of possible BAP combinations, for the BAPs listed in the bapIDs field. The number of values in this field shall be an integer multiple of BAP indices as given in the bapIDs field. The entries shall be ordered as follows: first, the BAP combinations with the first listed BAP having lowest values are listed. If there are more than one entry with the same value for the first BAP, the entries are sorted considering the second listed BAP, etc.

displacements is a list of vectors; for each vertex indexed in the vertexIDs field, the displacement vectors are given for the BAP combinations defined in the bapCombinations field. There must be exactly (num(VertexIDs)*num(bapCombinations)/num(bapIDs)) values in this field.

numInterpolateKeys is the number of BAP keys for interpolation, as defined below. The allowed values are 1-5.

In most cases, the list of BAPs in the **bapIDs** field will be the related BAPs (for example, the shoulder BAPs will typically be listed in the same **BodyDefTable** node). During animation, when the decoder receives a list of BAPs, which affects one or more **IndexedFaceSets** of the body model, it finds the associated BAP combination entries in the **BodyDefTable** nodes, and displaces the vertices from the original surface, with the vector specified by the **displacement** field.

Example:

```
BodyDefTable {
   bodySceneGraphNodeName    "l_forearm"
   bapIDs          [ 38, 40 ]
   vertexIds       [ 50, 51, 52 ]
   bapCombinations [ 0, 0, 0, 100, 0, 200,
                   100, 0, 100, 100, 100, 200,
                   200, 0, 200, 100, 200, 200]
   displacements   [ 1 0 0,0.9 0 0,5.0 0.1 0.1,
                   0 0.3 0.3,0.4 0 0,5.0 0 0.1,
                   0.5 0.6 0,0.9 0 0,5 0.7 0.1]
}
```

This **BodyDefTable** node defines the deformation of the forearm based on the combination of l_elbow_flexion and l_elbow_twisting BAPs. The vertices with indices 50,51,52 on surface l_forearm are deformed. The displacements for vertex 50 are: (1 0 0), (0 0.3 0.3) and (0.5 0.6 0) for the BAP l_elbow_flexion and l_elbow_twisting combinations (0 0) (0 100) (0 200), respectively.

The number of entries in the **displacements** field is calculated as:

Ndisplacements = NbapCombinations * NvertexIds

where Nm represents the number of entries in node m.

NbapCombinations = O(NbapIDs , Nkey_postures)

Any number of BAPs can be listed in one table, and a number of BodyDefTable nodes can be used for the same bodySceneGraphNodeName.

Interpolation

When the current BAP set for one frame does not contain the bapCombinations as listed in the **BodyDefTable** node, the entries in this node need to be interpolated to obtain the deformations. (For example, let the deformation of the right forearm be defined by BAPs 39 and 41 (right_elbow_flexion and right_elbow_twisting). Let the bapCombination entries in

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

the table be (0,0), (0,10000), (10000,0). Then, when a BAP39-BAP41 combination of (5000,5000) is received, the displacements for the frame should be interpolated from the listed BAP values.)

Given several BAP keys $P_1, P_2, P_3,...,P_n$ computing linear interpolation at BAP point P. n represents the **numInterpolateKeys** field in the **BodyDefTable** node.

Let $\mathbf{d_1}, \mathbf{d_2}, \mathbf{d_3}...\mathbf{d_n}$ be respective distances from P to keys.

Let $v_1, v_2, v_3,..v_n$ be tabular displacement values of a vertex at the keys.

For any key P_i, the deformation contributed by P_i should be *inversely* proportional to distance $\mathbf{d_i}$ from point P. Let this proportionality factor be $\mathbf{f_i}$.

$$\text{Thus } DEF_i \text{ (deformation due to } P_i \text{)} = \mathbf{f_i} * v_i$$

$$DEF \text{ (Total deformation at } P \text{)} = \mathbf{f_1} * v_1 + \mathbf{f_2} * v_2 + ... + \mathbf{f_n} * v_n$$

With the condition that $\mathbf{f_1 + f_2 + ... + f_n} = 1.0$

computing $\mathbf{f_i}$ is obtained in the following way:

Let total distance $\mathbf{D = d_1 + d_2 + ... + d_n}$

$$\mathbf{f_i} = (1 - \mathbf{d_i/D})/(\mathbf{n}-1)$$

calculation:

First compute DIRECT proportionality factors t_i

$$\mathbf{t_1 = d_1/D, t_2 = d_2/D,...,t_n = d_n/D}$$

Now $\mathbf{t_1 + t_2 + ... + t_n} = 1.0$

If we take deformation contribution by P_i as $DEF_i = \mathbf{t_i} v_i$

then keys closest to point p contribute least. To have the opposite effect we get inverse proportionality by replacing $\mathbf{t_i}$ s

$$\mathbf{t_i} \text{ <-- } (1 - \mathbf{t_i}) \text{ <-- } (1 - \mathbf{d_i/D})$$

$$\mathbf{t_1} \text{ <-- } 1 - \mathbf{d_1/D}, \mathbf{t_2} \text{ <- } 1 - \mathbf{d_2/D}, ... \mathbf{t_n} \text{ <- } 1 - \mathbf{d_n/D}$$

But now

$$\mathbf{t_1 + t_2 + ... + t_n} = \mathbf{n} - 1$$

To make right hand side 1.0, we divide by **n**-1. Thus the final factor $\mathbf{f_i}$

$$\mathbf{f_i} = \mathbf{t_i}/(\mathbf{n}-1) = (1-\mathbf{d_i/D})/(\mathbf{n}-1)$$

Note that the default body posture is defined where all BAPs are 0 and the displacements are 0. This default posture shall not be used as a BAP combinations entry for interpolation, unless it is defined explicitly as BAP combinations in the **BodyDefTable**.

7.2.2.26 BodySegmentConnectionHint

7.2.2.26.1 Node Interface

BodySegmentConnectionHint {			
exposedField	SFString	**firstSegmentNodeName**	NULL
exposedField	SFString	**secondSegmentNodeName**	NULL
exposedField	MFInt32	**firstVertexIdList**	[]
exposedField	MFInt32	**secondVertexIdList**	[]
}			

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2005 – All rights reserved
Not for Resale

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.26.2 Functionality and semantics

Defines the connection information of segments as a hint for maintaining connected surfaces. Typically, two segments connected by a joint might require listing corresponding vertices in both segments, as a hint to the **BodyDefTable** interpreter to remove holes.

The **BodySegmentConnectionHint** node is transmitted after the BIFS bitstream of **BDP** and **BodyDefTable** nodes. There is no limit on the number of **BodySegmentConnectionHint** nodes transmitted for one body. This node is a hint to the BDP interpreter; it is not required to use this node.

Each **BodySegmentConnectionHint** node contains two segments, a list of vertex ids in the two segments that need to be connected to each other for smooth rendering (for example, vertices near the elbow joint can be listed).

Detailed Semantics:

firstSegmentNodeName is the name of the segment containing the first **IndexedFaceSet** node. This node shall be part of the bodySceneGraph as defined in the **BDP** node. This node will be contained in the children field of the Segment node.

secondSegmentNodeName is the name of the segment containing the second **IndexedFaceSet** node. This node shall be part of the bodySceneGraph as defined in the **BDP** node. This node will be contained in the children field of Segment node.

firstVertexIdList is a list of indices into the Coordinate node of the first IndexedFaceSet node specified by firstSegmentNodeName.

secondVertexIdList is a list of indices into the Coordinate node of the second IndexedFaceSet node specified by secondSegmentNodeName.

The number of entries in firstVertexIdList and secondVertexIdList fields has to be the same. The corresponding vertex ids should be in the same sequence in both fields.

During animation, when the decoder displaces vertices from the original surfaces based on the vectors specified by **BodyDefTable** nodes, it can use the **BodySegmentConnectionHint** node to connect the two surfaces. If two corresponding vertices are not displaced with the same amount due to different **BodyDefTable** displacement values or due to numerical error, then the decoder can take the average displacement of two corresponding vertices.

7.2.2.27 Box

7.2.2.27.1 Node interface

Box {
field SFVec3f **size** 2, 2, 2
}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.27.2 Functionality and semantics

The semantics of the **Box** node are specified in ISO/IEC 14772-1:1998, subclause 6.7.

7.2.2.28 Circle

7.2.2.28.1 Node interface

Circle {
exposedField SFFloat **radius** 1.0
}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.28.2 Functionality and semantics

This node specifies a circle centred at (0,0) in the local coordinate system. The **radius** field specifies the radius of the circle and shall be greater than 0. The default texture mapping coordinates are defined as the four corners of the bounding box of the circle.

7.2.2.29 Clipper2D

7.2.2.29.1 Node interface

Clipper2D {			
eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFNode	**geometry**	NULL
exposedField	SFBool	**inside**	TRUE
exposedField	SFNode	**transform**	NULL
exposedField	SFBool	**XOR**	FALSE
}			

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.29.2 Functionality and semantics

The **Clipper2D** node is a 2D grouping node that defines a free-form 2D rendering area for its children nodes. If another **Clipper2D** node is found in its children, children of that second clipper shall be clipped/cut using the combination of both clipping geometries, as indicated by the **inside** and **XOR** fields of both clippers.

The **geometry** field specifies a 2D graphical primitive to be used as the clipper shape. All 2D graphical primitives are allowed except **Bitmap**, **PointSet2D** and **IndexedLineSet2D**. If the geometry defines an open shape (for instance, **Curve2D**), the shape shall be ignored. If the geometry is NULL, children nodes are completely drawn if the **inside** field is FALSE, otherwise children are not drawn.

The **inside** field specifies whether the node shall perform a clipping operation or a cut operation. If its value is TRUE, the inside of the clipping geometry is drawn. If it is FALSE, the outside of the clipping geometry is drawn.

The **transform** field specifies a 2D transformation node (**Transform2D** or **TransformMatrix2D**). This node shall have no child, and is used to assign a 2D transformation to the geometry of the **Clipper2D** node.

The **XOR** field specifies whether union or intersection of this clipper with its parent clipper is made using a XOR operation or not. The **XOR** field is used only if this clipper and its parent clipper have the same value for the **inside** field, otherwise it is ignored.

Example of clipper cascade:

Let's draw the following scene (pixel metrics, scene size 100x100):

```
OrderedGroup {
 children [
  Background2D {backColor 1 1 1}
  DEF Clip1 Clipper2D {
   geometry rectangle { size 75 25}
   children [
    DEF Clip2 Clipper2D {
     geometry Circle { radius 25 }
     children [
      Shape {
       appearance Appearance {
        material Material2D {
         emissiveColor 0 0 0
         filled TRUE
        }
       }
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        geometry Rectangle { size 100 100 }
       }
      ]
     }
    ]
   }
  ]
}
```

Figure 13 shows the result of the preceding scene with different inside and XOR fields for both clippers.

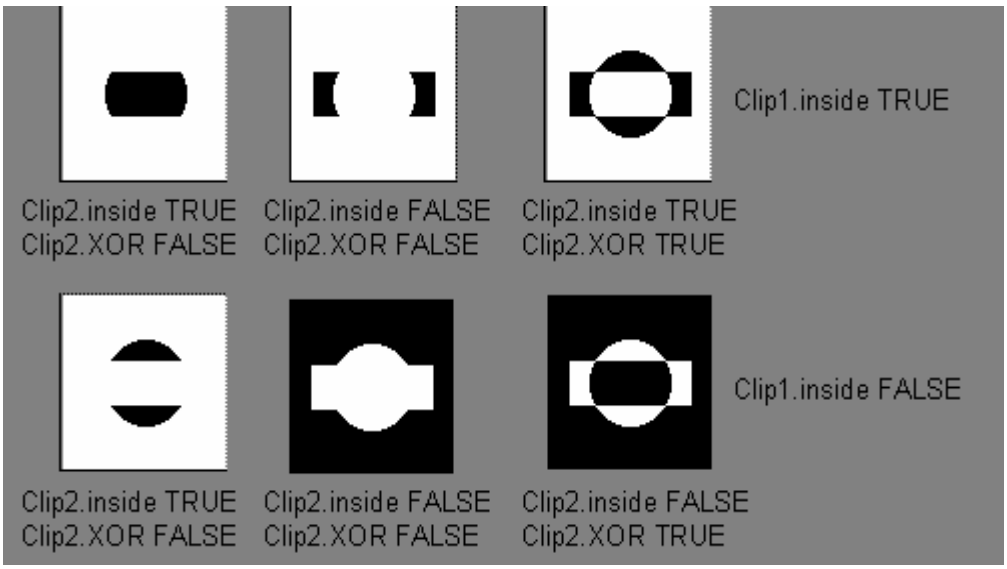


Figure 13 — Usage of the Clipper2D Node

7.2.2.30 Collision

7.2.2.30.1 Node interface

Collision {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFBool	**collide**	TRUE
field	SFNode	**proxy**	NULL
eventOut	SFTime	**collideTime**	

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.30.2 Functionality and semantics

The semantics of the Collision node are specified in ISO/IEC 14772-1:1998, subclause 6.8. ISO/IEC 14496-1 does not support the bounding box parameters (bboxCenter and bboxSize).

7.2.2.31 Color

7.2.2.31.1 Node interface

Color {

exposedField	MFColor	**color**	[]

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.31.2 Functionality and semantics

The semantics of the **Color** node are specified in ISO/IEC 14772-1:1998, subclause 6.9.

7.2.2.32 ColorInterpolator

7.2.2.32.1 Node interface

ColorInterpolator {

eventIn	SFFloat	**set_fraction**	
exposedField	MFFloat	**key**	[]
exposedField	MFColor	**keyValue**	[]
eventOut	SFColor	**value_changed**	

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.32.2 Functionality and semantics

The semantics of the **ColorInterpolator** node are specified in ISO/IEC 14772-1:1998, subclause 6.10.

7.2.2.33 ColorTransform

7.2.2.33.1 Node interface

ColorTransform **{**

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFFloat	**mrr**	1
exposedField	SFFloat	**mrg**	0
exposedField	SFFloat	**mrb**	0
exposedField	SFFloat	**mra**	0
exposedField	SFFloat	**tr**	0
exposedField	SFFloat	**mgr**	0
exposedField	SFFloat	**mgg**	1
exposedField	SFFloat	**mgb**	0
exposedField	SFFloat	**mga**	0
exposedField	SFFloat	**tg**	0
exposedField	SFFloat	**mbr**	0
exposedField	SFFloat	**mbg**	0
exposedField	SFFloat	**mbb**	1
exposedField	SFFloat	**mba**	0
exposedField	SFFloat	**tb**	0
exposedField	SFFloat	**mar**	0
exposedField	SFFloat	**mag**	0
exposedField	SFFloat	**mab**	0
exposedField	SFFloat	**maa**	1
exposedField	SFFloat	**ta**	0

}

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.33.2 Functionality and semantics

The **ColorTransform** node is a grouping node that defines a color space transformation for any Color field of its children in the RGBA space. See ISO/IEC 14772-1:1998 for a description of the **children**, **addChildren**, and **removeChildren** fields and eventIns.

The **m*** and **t*** fields define a color transformation in RGBA intensity space (each component varying between 0.0 and 1.0) based on the following transformation matrix:

$$T = \begin{pmatrix} m_{rr} & m_{rg} & m_{rb} & m_{ra} & t_r \\ m_{gr} & m_{gg} & m_{gb} & m_{ga} & t_g \\ m_{br} & m_{bg} & m_{bb} & m_{ba} & t_b \\ m_{ar} & m_{ag} & m_{ab} & m_{aa} & t_a \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Given an RGBA color C and a **ColorTransform** node, C is transformed into point C' the transformation whose matrix is T.

$$C' = T \times C$$

Colors defined with an SFColor are converted to RGBA by using the related transparency (see **Material**, **Material2D** and **XLineProperties**) information, with A = 1.0 – transparency, or the related opacity information (see **RadialGradient**, **LinearGradient**) with A = opacity.

7.2.2.34 CompositeTexture2D

7.2.2.34.1 Node interface

CompositeTexture2D {			
eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFInt32	**pixelWidth**	-1
exposedField	SFInt32	**pixelHeight**	-1
exposedField	SFNode	**background**	NULL
exposedField	SFNode	**viewport**	NULL
field	SFInt32	**repeatSandT**	3
}			

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.34.2 Functionality and semantics

The **CompositeTexture2D** node represents a texture that is composed of a 2D scene, which may be mapped onto another object.

This node may only be used as the texture field of an **Appearance** node. All behaviors and user interaction are enabled when using a **CompositeTexture2D**.

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.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The **children** field contains a list of 2D children nodes that define the 2D scene that is to form the texture map.

The **pixelWidth** and **pixelHeight** fields specify the ideal size in pixels of this map. The default values result in an undefined size being used. This is a hint for the content creator to define the quality of the texture mapping.

The semantics of the **background** and **viewport** fields are identical to the semantics of the **Layer2D** (see 7.2.2.72) fields of the same name.

The meaning of the field **repeatSandT** is the meaning of the combined **repeatS** and **repeatT** of the **ImageTexture** node. The value 0 is equivalent to **repeatS** = false, **repeatT** = false. The value 1 is equivalent to **repeatS** = true, **repeatT** = false. The value 2 is equivalent to **repeatS** = false, **repeatT** = true. The value 3 is equivalent to **repeatS** = true, **repeatT** = true.

Value (-1, -1) for the couple of fields (**pixelWidth**, **pixelHeight**) should not be used by authors since it does not give any hint on what should be the size of the off-screen surface allocated for the texture. Although these are exposedFields they should not be dynamically modified since they represent the physical size of the off-screen surface used for compositing.

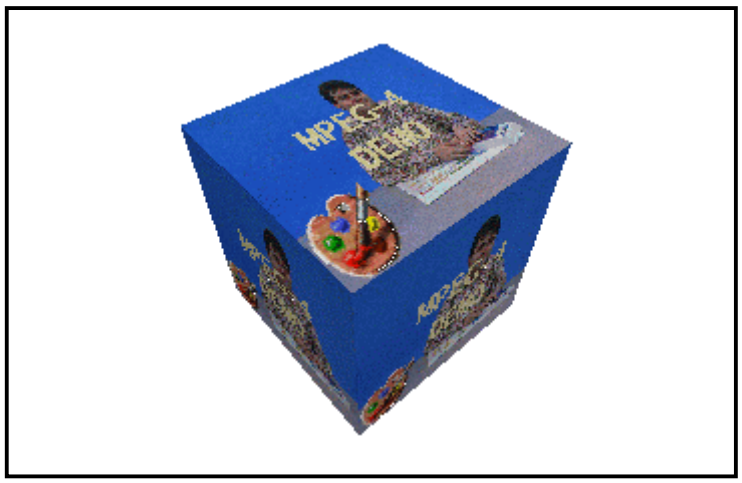


Figure 14 — A CompositeTexture2D example. The 2D scene is projcted onto the 3D cube.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2005 – All rights reserved
Not for Resale

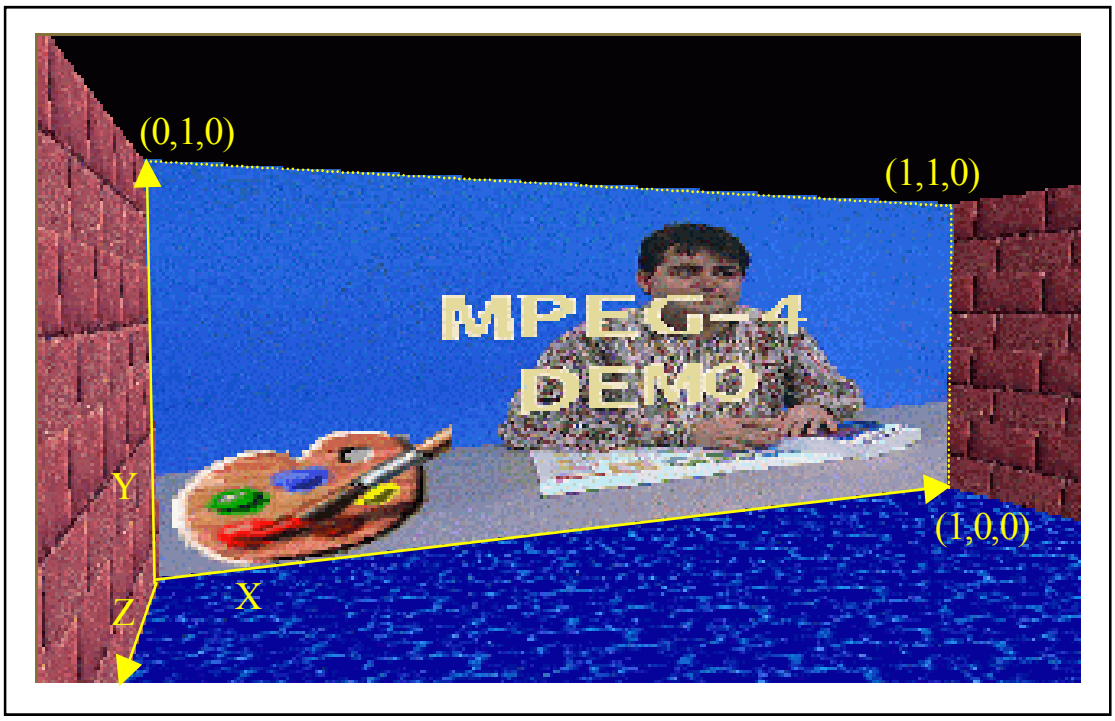


Figure 15 — A CompositeTexture2D example.

Here the 2D scene as defined in Figure 14 composed of an image, a logo, and a text, is textured on a rectangle n in the local X,Y plane of the back wall. A similar effect may be obtained by simply placing the 2D objects in the (3D) **Transform**. However, **CompositeTexture2D** and **CompositeTexture3D** shall be used when maping onto non-flat geometries.

7.2.2.35 CompositeTexture3D

7.2.2.35.1 Node interface

CompositeTexture3D {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFInt32	**pixelWidth**	-1
exposedField	SFInt32	**pixelHeight**	-1
exposedField	SFNode	**background**	
exposedField	SFNode	**fog**	
exposedField	SFNode	**navigationInfo**	
exposedField	SFNode	**viewpoint**	
field	SFBool	**repeatS**	TRUE
field	SFBool	**repeatT**	TRUE
eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFInt32	**pixelWidth**	-1
exposedField	SFInt32	**pixelHeight**	-1
exposedField	SFNode	**background**	
exposedField	SFNode	**fog**	
exposedField	SFNode	**navigationInfo**	
exposedField	SFNode	**viewpoint**	
field	SFInt32	**repeatSandT**	3

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.35.2 Functionality and semantics

The **CompositeTexture3D** node represents a texture mapped onto a 3D object that is composed of a 3D scene.

Behaviors and user interaction are enabled when using a **CompositeTexture3D**. However, the standard user navigation on the textured scene is disabled. Instead, sensors contained in the scene which forms the **CompositeTexture3D** may be used to define behaviours. This node may only be used as a **texture** field of an **Appearance** node.

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 is the list of 3D children nodes that define the 3D scene that forms the texture map.

The **pixelWidth** and **pixelHeight** fields specify the ideal size in pixels of this map. The default values result in an undefined size being used. This is a hint for the content creator to define the quality of the texture mapping.

The **background**, **fog**, **navigationInfo** and **viewpoint** fields represent the current values of the bindable children nodes used in the 3D scene. This node may only be used as the **texture** field of an **Appearance** node. All behaviors and user interaction are enabled when using a **CompositeTexture2D**.

The semantics of the field **repeatSandT** is the same as those of the same field of the **CompositeTexture2D** node.

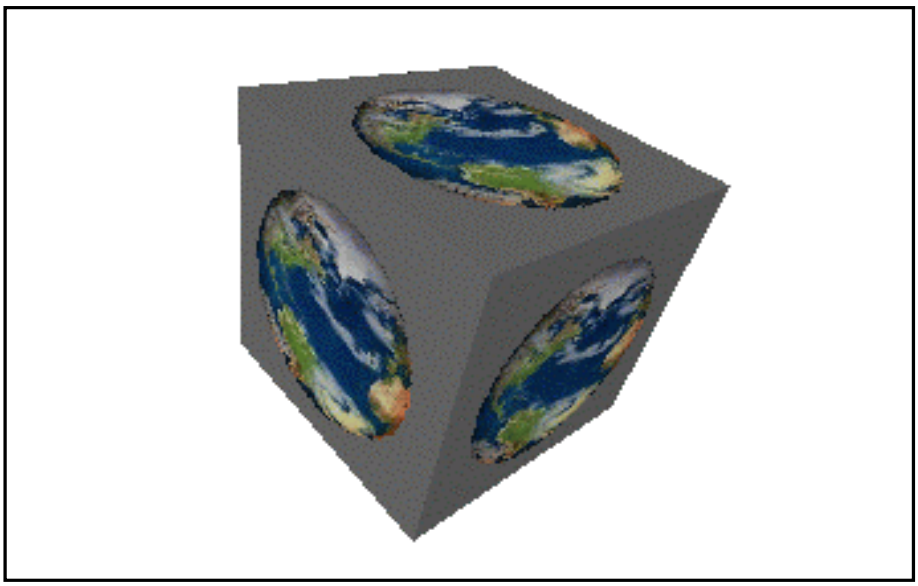


Figure 16 — CompositeTexture3D example. The 3D view of the earth is projected onto the 3D cube

7.2.2.36 Conditional

7.2.2.36.1 Node interface

```
Conditional {
    eventIn         SFBool      activate
    eventIn         SFBool      reverseActivate
    exposedField    SFString    buffer              ""
    eventOut        SFBool      isActive
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.36.2 Functionality and semantics

The **Conditional** node interprets a buffered bit string of BIFS-Commands when it is activated. This allows events to trigger node updates, deletions, and other modifications to the scene. The buffered bit string is interpreted as if it had just been received.

Upon reception of either an SFBool event of value TRUE on the **activate** eventIn, or an SFBool event of value FALSE on the **reverseActivate** eventIn, the contents of the buffer field shall be interpreted as a BIFS `CommandFrame` (see 8.6.2). These updates are not time-stamped; they are executed at the time of the event, assuming a zero-decoding time.

The **isActive** eventOut field sends out a TRUE event just before the BIFS command(s) in the buffer are executed. The command(s) are then executed, and then **isActive** sends out a FALSE event. Since the complete execution of the BIFS command(s) in the buffer happens within one time stamp, implementations shall ensure that the **isActive** TRUE event is propagated before the FALSE event is sent out.

NOTE — The propagation of the **isActive** TRUE and FALSE events allows the creation of an execution chain of **Conditional** nodes by routing the **isActive** event of a **Conditional** to the **reverseActivate** field of the next one in the chain.

EXAMPLE — A typical use of this node is for the implementation of the action of a button. The button geometry is enclosed in a grouping node which also contains a **TouchSensor** node. The **isActive** eventOut of the **TouchSensor** is routed to the activate eventIn of **Conditional** C1 and to the **reverseActivate** eventIn of **Conditional** C2; C1 then implements the “mouse-down” action and C2 implements the “mouse-up” action.

7.2.2.37 Cone

7.2.2.37.1 Node interface

```
Cone {
   field       SFFloat     bottomRadius      1.0
   field       SFFloat     height            2.0
   field       SFBool      side              TRUE
   field       SFBool      bottom            TRUE
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.37.2 Functionality and semantics

The semantics of the **Cone** node are specified in ISO/IEC 14772-1:1998, subclause 6.11.

7.2.2.38 Coordinate

7.2.2.38.1 Node interface

```
Coordinate {
   exposedField   MFVec3f     point             []
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.38.2 Functionality and semantics

The semantics of the **Coordinate** node are specified in ISO/IEC 14772-1:1998, subclause 6.12.

7.2.2.39 Coordinate2D

7.2.2.39.1 Node interface

```
Coordinate2D {
   exposedField   MFVec2f     point             []
```

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.39.2 Functionality and semantics

This node defines a set of 2D coordinates to be used in the **coord** field of geometry nodes.

The **point** field contains a list of points in the 2D coordinate space (see 7.1.1.2.2).

7.2.2.40 CoordinateInterpolator

7.2.2.40.1 Node interface

```
CoordinateInterpolator {
    eventIn         SFFloat     set_fraction
    exposedField    MFFloat     key                 []
    exposedField    MFVec3f     keyValue            []
    eventOut        MFVec3f     value_changed
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.40.2 Functionality and semantics

The semantics of the **CoordinateInterpolator** node are specified in ISO/IEC 14772-1:1998, subclause 6.13.

7.2.2.41 CoordinateInterpolator2D

7.2.2.41.1 Node interface

```
CoordinateInterpolator2D {
    eventIn         SFFloat     set_fraction
    exposedField    MFFloat     key                 []
    exposedField    MFVec2f     keyValue            []
    eventOut        MFVec2f     value_changed
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.41.2 Functionality and semantics

CoordinateInterpolator2D is the 2D equivalent of **CoordinateInterpolator** (see 7.2.2.40).

7.2.2.42 CoordinateInterpolator4D

7.2.2.42.1 Node interface

```
CoordinateInterpolator4D {
    eventIn         SFFloat     set_fraction
    exposedField    MFFloat     key                 []
    exposedField    MFVec4f     keyValue            []
    eventOut        MFVec4f     value_changed
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

7.2.2.42.2 Functionality and semantics

As **CoordinateInterpolator**, this node linearly interpolates 4-dimensional values.

7.2.2.43 Curve2D

7.2.2.43.1 Node interface

Curve2D {			
exposedField	SFNode	**point**	NULL
exposedField	SFFloat	**fineness**	0.5
exposedField	MFInt32	**type**	[]
}			

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.43.2 Functionality and semantics

This node is used to describe the Bezier approximation of a polygon in the scene at an arbitrary level of precision. It behaves as other “lines”, which means it is sensitive to modifications of line width and “dotted-ness”, and can be filled or not.

The given parameters are a control polygon and a parameter setting the quality of approximation of the curve. Internally, another polygon of fineness points is computed on the basis of the control polygon. The coordinates of that internal polygon are given by the following formula:

$$x[j] = \sum_{i=0}^{n} xc[i] \times \frac{(n-1)!}{i!(n-1-i)!} \times \left(\frac{j}{f}\right)^{i} \times \left(1-\frac{j}{f}\right)^{n-1-i},$$

where *x[j]* is the jth x coordinate of the internal polygon, *n* is the number of points in the control polygon, *xc[i]* is the ith x coordinate of the control polygon and *f* is short for the above fineness parameter. A similar formula yields the y coordinates.

The **point** field shall list the vertices of the control polygon.

The **fineness** parameter is an SFFloat value that indicates how finely to tessellate the Bezier curves. A value of 1 means that the curve shall be fine enough that no edges are visible. A value of 0 indicates that a straight line shall be drawn between the two points of the curve. The default value of 0.5 gives an intermediate level of smoothness. The amount of tessellation may be adjusted according to scale of the shape, making it possible to avoid visible edges appearing when the shape is zoomed. When the field **type** is specified, the above functionality is extended as follows: the curve is now defined piecewise either with the above equation or as straight segments or as non-segments, depending on the values in **type**. The **point** field is now taken to contain all key-points (points where the curve passes) and control-points (points defining the aspect of the curve around them). The values in the **type** field define the semantics of the elements of **point**.

The **point** field contains a **Coordinate2D** field with the list of points. If the **type** field is non-empty, then it shall contain tokens indicating how the point list is to be interpreted, according to the following algorithm (expressed in pseudo-code):

```
SFInt32 i     = 0;
SFInt32 j     = 0;
SFVec2f cur   = point[i++];
SFVec2f first = cur;
SFVec2f curctl;

while (i < point.length)
   SFInt32 t = 0;
   if (type.length > j) t = type[j++];

   switch(t) {
      case 0: // move, use 1 point
         if (is_filled) draw_line(cur, point[i]);
         cur = point[i];
         i++;
         break;

      case 1: // line, use 1 point
```

Copyright International Organization for Standardization © ISO/IEC 2005 All rights reserved
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
            draw_line(cur, point[i]);
            cur = point[i];
            i++;
            break;

         case 2: // bezier curve, use 3 points
            draw_curve(cur, point[i], point[i+1], point[i+2]);
            cur = point[i+2];
            curctl =point[i+1];
            i += 3;
            break;

         case 3: // tangent curve, use 2 points
            SFVec2f tanctl;
            tanctl.x = 2*cur.x - curctl.x;
            tanctl.y = 2*cur.y - curctl.y;
            draw_curve(cur, tanctl, point[i], point[i+1]);
            cur = point[i+1];
            curctl = point[i];
            i += 2;
            break;
      }
   }
   if (is_filled) draw_line(cur, first);
```

In the above pseudo-code, `draw_line(a,b)` draws a line from a to b and `draw_curve(a,b,c,d)` draws a Bezier curve from a to d, using b as the control point for a and c as the control point for d. Note that, because of the move command (`type` = 0) multiple disjoint segments are possible. In the case of a filled shape, each segment is closed by drawing a straight line from the last point in the segment to the first. Shapes are filled using the odd-even winding fill rule. If one segment is contained within another, the inside of the inner shape is not filled, allowing shapes with holes.

The first coordinate pair in **point** is the starting point of the curve. The first value in **type** describes the treatment to be applied to the subsequent coordinate pairs. At any time, a value in **type** describes the characteristics of the next curve segment. If **P is** the starting point or the last point of the previous segment of the curve; **N** the ending point of the current curve segment; $\mathbf{C_1}$ the control point on the side of **P** and $\mathbf{C_2}$ the control point on the side of **N.**

The permitted values of **type** are:

0 = MoveTo: One coordinate pair in the **point** list is consumed, defining **N**. **P** ends the curve. The curve shall start again at **N**. Sequences of two or more MoveTos shall not occur. MoveTo shall not occur as the first element in **type.**

1 = LineTo: One coordinate pair in the **point** list is consumed, defining **N**. A straight line is drawn from **P** to **N**.

2 = CurveTo: Three coordinate pairs in the **point** list are consumed, defining $\mathbf{C_1}$, $\mathbf{C_2}$ and **N** respectively. The first coordinate pair specifies the control point the start of this curve segment ($\mathbf{C_1}$), the second specifies the control point for end of the curve segment ($\mathbf{C_2}$) and the third specifies the ending point of the curve segment (**N**).

3 = NextCurveto: Two coordinate pairs in the **point** list are consumed, defining $\mathbf{C_2}$ and **N** in this order. The first coordinate pair specifies the control point for the end of the curve segment ($\mathbf{C_2}$), and the second specifies the ending point of the curve segment (**N**). The control point $\mathbf{C_1}$ for the start of the curve segment is derived from the previous control point. If the previous segment was formed with CurveTo or NextCurveTo, the start control point $\mathbf{C_1}$ is symmetrical to the end control point $\mathbf{C'_2}$ of the previous curve segment with respect to point **P**. This control type shall not occur immediately following a MoveTo or LineTo.

The formula for obtaining the coordinates of $\mathbf{C_1}$ in the case of a NextCurveTo is:

$$\mathbf{C_{1x}} = 2.\mathbf{P_x} - \mathbf{C'_{2x}} \quad \text{and} \quad \mathbf{C_{1y}} = 2.\mathbf{P_y} - \mathbf{C'_{2y}}$$

The first point in **point**, as the first point in the curve, is implicitly a MoveTo.

For CurveTo and NextCurveTo, the piece of curve is constructed using the above formula as applied to a polygon constructed from four points, that is the starting point **P**, the first control point $\mathbf{C_1}$, the second control point $\mathbf{C_2}$ and the end point **N**, which is the next point in the point list.

The curve shall be continuous except at points tagged with MoveTo. The tangent of the curve is only continuous at points tagged with NextCurveTo, or at points where the previous second control point $\mathbf{C'_2}$, the key point **P** and the next first control point $\mathbf{C_1}$ are aligned.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If there are more values in **point** than specified by **type**, then the unused points shall describe a curve as if no **type** was defined.

An unfilled curve shall not be lit, texture mapped, nor collided with during collision detection. A filled curve that is texture mapped shall used texture mapping coordinates as defined by the four corners of the bounding box of the internal polygon.

EXAMPLE —

```
geometry Curve2D {
   point Coordinate2D {
         points [ 0 0 0 100 200 100 200 200 210 200 220 200 ]
      }
      type [ 2 0 1 ]
}
```

The first segment of curve starts at 0,0 goes to 200,200 and control points are 0,100 and 200,100. The Bezier curve drawn is the one with the polygon [0 0 0 100 200 100 200 200] (represented in dotted gray) when types=null, with the same fineness. When types is specified, the fineness parameter is applied to each curve segment. Then we have a "move to", from 200,200 to 210,200. Then we have a "line to", from 210,200 to 220,200 (small segment in upper right corner).

In Figure 17, the curve is drawn in wide black, and the control polygon is drawn in dotted gray. The curve has two connex components.

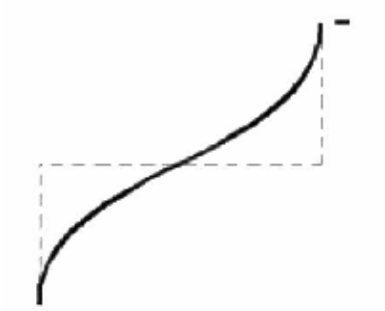


Figure 17 — Curve node example

7.2.2.44 Cylinder

7.2.2.44.1 Node interface

Cylinder {			
field	SFBool	**bottom**	TRUE
field	SFFloat	**height**	2.0
field	SFFloat	**radius**	1.0
field	SFBool	**side**	TRUE
field	SFBool	**top**	TRUE
}			

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.44.2 Functionality and semantics

The semantics of the **Cylinder** node are specified in ISO/IEC 14772-1:1998, subclause 6.14.

7.2.2.45 CylinderSensor

7.2.2.45.1 Node interface

CylinderSensor {			
exposedField	SFBool	**autoOffset**	TRUE
exposedField	SFFloat	**diskAngle**	0.262

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

exposedField	SFBool	**enabled**	TRUE
exposedField	SFFloat	**maxAngle**	-1.0
exposedField	SFFloat	**minAngle**	0.0
exposedField	SFFloat	**offset**	0.0
eventOut	SFBool	**isActive**	
eventOut	SFRotation	**rotation_changed**	
eventOut	SFVec3f	**trackPoint_changed**	

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.45.2 Functionality and semantics

The semantics of the **CylinderSensor** node are specified in ISO/IEC 14772-1:1998, subclause 6.15.

7.2.2.46 DirectiveSound

7.2.2.46.1 Node interface

DirectiveSound {

field	MFFloat	**angles**	0
field	MFFloat	**directivity**	1
field	MFFloat	**frequency**	[]
field	SFFloat	**speedOfSound**	340
field	SFFloat	**distance**	1000
field	SFBool	**useAirabs**	FALSE
exposedField	SFVec3f	**direction**	0, 0, 1
exposedField	SFFloat	**intensity**	1
exposedField	SFVec3f	**location**	0, 0, 0
exposedField	SFNode	**source**	NULL
exposedField	SFNode	**perceptualParameters**	NULL
exposedField	SFBool	**roomEffect**	FALSE
exposedField	SFBool	**spatialize**	TRUE

}

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.46.2 Functionality and semantics

The purpose of the **DirectiveSound** node is to obtain sound source directivity which is characteristic to the sound source present in a 3-D scene. It is also needed for rendering of the acoustic response of the virtual environment. The modeling of sound propagation from the source to the listening point includes distance dependent attenuation, propagation delay between the source and the listener, and modeling of sound reflections, transmission through objects, and reverberation. Two different rendering schemes are applied to **DirectiveSound** depending on the value of **perceptualParameters** field. If this field is NULL, the *physical* approach is applied, and if it contains a **PerceptualParameters** node, the *perceptual* approach is applied (see 7.1.1.2.13.4).

DirectiveSound is rendered in a specified area in a 3-D scene. The **distance** field specifies the radius of a spherical region around the source where the sound is audible to the user. Additionally, in the physical approach, a 3-D rectangular region specified in the **AcousticScene** nodes specify areas in the scene where the sound is audible when the **DirectiveSound** and the **Viewpoint** or **ListeningPoint** are both inside that area.

The direction dependent sound radiation properties of the sound source is defined in the **directivity** field of the node for an arbitrary number of angles given in the **angles** field with respect to the main direction axis (defined in the **direction** field) to the back of the sound source.

The **angles** field specifies the angles between the direction vector of the source and the vector between the sound source location and the listener (**Viewpoint** or **ListeningPoint**) in radians, at which the directivity parameters apply.

The semantics of the **directivity** field is defined in two different ways depending on the value of the **frequency** field, which in one case is an empty vector [], and in the other case is a MFField containing a set of frequencies at which digital filter magnitude response gains are valid. Both ways are allowed in the physical approach, but in the perceptual approach only one is allowed.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

frequency field defines the frequencies at which the directivity gains are valid (similarly as **refFrequency** and **transFrequency** for **reffunc** and **transfunc** in **AcousticMaterial**, see 7.2.2.1).

There are two different ways of defining directivity for a sound source. If the **frequency** field is [], the parameters in the **directivity** field are considered as a set of digital filter coefficients, and if this field is different from [] its semantics are as explained above.

In the *physical approach* both ways of defining directivity are possible. If **frequency** is equal to [], the directivity can be defined as a single scale factor associated to each given azimuth angle, or as frequency modifying digital filter parameters. In the latter case, the general form for the field is:

[$b_{\alpha 0,0}$, $b_{\alpha 0,1}$, $b_{\alpha 0,2}$, …, $b_{\alpha 0,M}$, $a_{\alpha 0,1}$, $a_{\alpha 0,2}$,.., $a_{\alpha 0,M}$, $b_{\alpha 1,0}$, $b_{\alpha 1,1}$, $b_{\alpha 1,2}$, …, $b_{\alpha 1,M}$, $a_{\alpha 1,1}$, $a_{\alpha 1,2}$,.., $a_{\alpha 1,M}$,..]

where $\alpha 0$ is the first specified angle in the **angles** field, $\alpha 1$ is the second angle etc., and M is the order of the digital filter. If the directivity is specified as gains, the form of the field is:

[$b_{\alpha 0}$, $b_{\alpha 1}$, …, $b_{\alpha K}$,]

where K is the number of specified angles.

These coefficients represent a digital filter, whose system function $H(z)$ is represented in the z-domain as a division of the z-transform of the output sequence $Y(z)$ with the z-transform of the input sequence $X(z)$:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{M} a_k z^{-k}}.$$

The distinction between the coefficients of different filters is obtained by dividing the length of the directivity field by the amount of specified angles (length of the **angles** field):

$$n = \frac{length(\mathbf{directivity})}{length(\mathbf{angles})} = 2 \cdot M + 1 = m_b + m_a, ,$$

where n is the number of coefficients in each filter, and M is the order of the filter, m_b is the number of b coefficients, and m_a is the number of a coefficients Thus, the number of a coefficients is

$$m_a = \frac{n-1}{2},$$

and the number of b coefficients is

$$m_b = \frac{n-1}{2} + 1.$$

If the first angle $\alpha 0 > 0$, the directivity at angles $0 < \alpha < \alpha 0$ is the same as at $\alpha 0$, and if the last specified angle αM is smaller than π, the directivity at angles $\alpha M < \alpha < \pi$ is the same as at αM.

The second way of defining directivity is to give a set of gains in **directivity** field at frequencies defined in **frequency** field. This scheme can be used both in perceptual and in the physical approaches.

The source directivity is defined as gain factors at specified frequencies for a set of reference angles (specified in the **angles** field). In this approach, the general form for the **directivity** field is then:

[$gain^0_0$, $gain^0_1$,...$gain^0_{nf-1}$,$gain^1_0$, $gain^1_1$,...$gain^1_{nf-1}$..., $gain^{na-1}_0$, $gain^{na-1}_1$,...$gain^{na-1}_{nf-1}$].

Where, nf is the number of reference frequencies, and $freq_j$ is the jth reference frequency

$gain^i_j$ is the gain for the ith reference angle and the jth reference frequency, and na is the length of the **angles** field.

The form of the **frequency** field is then:

[$freq_0$, $freq_1$, ...$freq_{nf-1}$]

The number of reference angles is the same as the length of the **angles** field, and the number of reference frequencies is the same as the length of **frequency** field. An example of **directivity** is given below:

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[0.9,0.85,0.7,0.6,0.55,

0.85,0.75,0.6,0.5,0.4,

0.8,0.65,0.5,0.4,0.3,

0.5,0.45,0.3,0.2,0.1]

and an example of **frequency** in this case is:

[250, 500, 1000, 2000, 4000]

Axisymmetry is assumed, so only angles from 0 to π radians are needed to fully define frequency-dependent directivity.

If not specified in the node, the default filtering at 0 rad is the same as for the first specified angle (α_0). If not specified in the node, the default gain at π rad is **gain** $^{\mathbf{na-1}}_{\mathbf{j}}$ for the j_{th} frequency.

If not specified in the node, the default gain at 0 Hz is **gain** $^{\mathbf{i}}_{\mathbf{0}}$ for the i_{th} angle.

By default, the gain for frequencies above $f_{nf-1}$is **gain** $^{\mathbf{i}}_{\mathbf{nf-1}}$ for the i_{th} angle.

The directivity filtering is defined by these gains at the specified frequencies.

In both physical and the perceptual approaches the output of directivity filtering between the specified angles should perform an interpolated result of the magnitude responses of the specified directivities. This can be a result of, e.g., crossfading between different filter outputs, or suitable interpolation of coefficients of the filters.

The **direction** field specifies the direction the **DirectiveSound** node is facing. This field is used in the directivity computation of the sound source, i.e., it defines the direction of the angle of 0 (rad) in the directivity field.

The **intensity** field specifies the gain the original sound stream is multiplied with.

The **speedOfSound** field is used to enable control of the pre-delay added to the sound depending on the distance between the source and the listener. With other values of **speedOfSound** than 0 the delay is computed as:

$$d = \frac{dist}{speedOfSound},$$

where *dist* is the current distance between the source and the listener in meters, and *speedOfSound* is the value of **speedOfSound** field in meters.

speedOfSound field also defines the delay of the reflections off acoustic surfaces in the physical approach, since they are computed according to the corresponding *image source* locations and the speed of sound. These acoustic surfaces are polygons defined in **IndexedFaceSet** nodes that have **AcousticMaterial** associated to them as their appearance (see 7.2.2.1). This field also controls the Doppler effect that is caused by the changing distance between the listening point and the listener. Thus the smaller the value of **speedOfSound** is, the stronger the Doppler effect is (pitch shift caused by the changing distance between the source and the listener). The changing delay caused by a varying distance between the source (direct sound or image source corresponding to a reflection) and the listener should always be interpolated to avoid artifacts such as clicks in the delayed sound.

The default value of **speedOfSound** is 0. With this value, and **roomEffect** = FALSE, no delay of sound propagation between the direct sound and the listener is rendered (except when there are physically rendered early reflections, see next paragraph). This enables a **DirectiveSound** node to be spatialized in a 3-D space so that the direction and attenuation of the sound are perceived according to the sound source location relative to the listener, but neither Doppler effect nor delay is implemented.

If the sound is rendered according to the physical approach, and the source and the listening point are located within an **AcousticScene** audibility region, and there are **IndexedFaceSet** surfaces with acoustic reflectivity, associated to that **AcousticScene**, and the value of **roomEffect** is TRUE, the Doppler effect and the delays of the direct sound and physical early reflections are computed according to the speed of sound in the air (340 m/s), even if this field is set to 0.

The **distance** field specifies the distance dependent attenuation of the sound. Its definition depends on the type of rendering :

Physical approach: Within **distance** meters from the source the sound is multiplied by the value of the **intensity** field before any spatial processing (directivity filtering, spatialization, or room effect). At a distance in meters given by the **distance** field, the sound has attenuated 60dB from the value within the 1-meter distance. Outside this distance from the sound source the sound is not audible. The attenuation function varies linearly on a dB vs. logarithm of distance scale between the source and the given cutoff distance such:

Attdb(d) = 0 , 0<= d <=1

Attdb(d) = 60 log2(d) / log2(distance) ,1<= d <=distance

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The radiation pattern defined by the **directivity** field will thus give the overall directivity, which will be uniformly attenuated as a function of distance. If, however, the **distance** field is set to 0, no distance dependent attenuation is applied.

Perceptual approach: Within **distance** meters from the source, the sound is multiplied by the value of the **intensity** field before any spatial processing (directivity filtering, spatialization, or room effect). Outside this distance from the sound source, the sound is not audible. Between 0 and **distance**, the distance dependent attenuation is performed according to 7.2.2.93.2.2 by modifying the source presence Es. If, however, the **distance** field is set to 0, no distance dependent attenuation shall be applied.

Field **useAirabs** specifies whether the distance dependent air absorption filtering is applied to the direct sound. ISO 9613-1:1993 specifies equations for air absorption curves in different humidity and temperature conditions, and the frequency modification of the distance dependent air absorption filtering should follow one of these curves at maximum accuracy possible.

location field specifies the 3-D location of the sound source in the local coordinate system of the **DirectiveSound.**

source field allows the connection of an audio source containing the sound.

The **spatialize** field has the same semantics concerning the direct sound, as in the **Sound** node, i.e., if this flag is set to TRUE, the sound stream attached to this node should be processed so that appears to come from the direction of sound source with respect to the current direction of the viewpoint. In the case of **DirectiveSound** (physical approach), this flag is also applied to the reflections caused by acoustic surfaces (specified by **IndexedFaceSet**s and **AcousticMaterial**s). When **spatialize =** TRUE, also the directions of the reflections are rendered. If the value of this flag is FALSE, the sound routed through **DirectiveSound** node, or its reflections are not spatialized according to their 3-D direction of arrival at the listener.

Field **roomEffect** is used for enabling and disabling environmental spatialization of audio. This field specifies whether the environmental response (*physical* case: reflections, reverberation, sound transmission filtering when propagating through surfaces; *perceptual* case: reverberation according to the **PerceptualParameters** node) is applied to this sound node. When this flag is TRUE the **DirectiveSound** source is spatialized according to the reflections and reverberation in the virtual environment. If, like mentioned above, also the **spatialize** flag is TRUE, the directions of the physical reflections are also rendered, and if **spatialize** is FALSE (but **roomEffect** is TRUE), a monophonic room acoustic effect is produced.

7.2.2.47 DiscSensor

7.2.2.47.1 Node interface

DiscSensor {

exposedField	SFBool	**autoOffset**	TRUE
exposedField	SFBool	**enabled**	TRUE
exposedField	SFFloat	**maxAngle**	-1.0
exposedField	SFFloat	**minAngle**	0.0
exposedField	SFFloat	**offset**	0.0
EventOut	SFBool	**isActive**	
EventOut	SFFloat	**rotation_changed**	
EventOut	SFVec2f	**trackPoint_changed**	

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.47.2 Functionality and semantics

This sensor enables the rotation of an object in the 2D plane around an axis specified in the local coordinate system. The semantics are as similar to those for **CylinderSensor**, but restricted to a 2D case.

7.2.2.48 DirectionalLight

7.2.2.48.1 Node interface

DirectionalLight {

exposedField	SFFloat	**ambientIntensity**	0.0
exposedField	SFColor	**color**	1, 1, 1
exposedField	SFVec3f	**direction**	0, 0, -1

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

exposedField	SFFloat	**intensity**	1.0
exposedField	SFBool	**on**	TRUE

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.48.2 Functionality and semantics

The semantics of the **DirectionalLight** node are specified in ISO/IEC 14772-1:1998, subclause 6.16.

7.2.2.49 ElevationGrid

7.2.2.49.1 Node interface

ElevationGrid {

EventIn	MFFloat	**set_height**	
exposedField	SFNode	**color**	NULL
exposedField	SFNode	**normal**	NULL
exposedField	SFNode	**texCoord**	NULL
Field	MFFloat	**height**	[]
Field	SFBool	**ccw**	TRUE
Field	SFBool	**colorPerVertex**	TRUE
Field	SFFloat	**creaseAngle**	0.0
Field	SFBool	**normalPerVertex**	TRUE
Field	SFBool	**solid**	TRUE
Field	SFInt32	**xDimension**	0
Field	SFFloat	**xSpacing**	1.0
Field	SFInt32	**zDimension**	0
Field	SFFloat	**zSpacing**	1.0

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.49.2 Functionality and semantics

The semantics of the **ElevationGrid** node are specified in ISO/IEC 14772-1:1998, subclause 6.17.

7.2.2.50 Ellipse

7.2.2.50.1 Node interface

Ellipse {

exposedField	SFVec2f	**radius**	1 1

}

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.50.2 Functionality and semantics

An ellipse is a planar geometry node and is defined by two-radius extent in its local coordinate system: **radius**[0] along x-direction and **radius**[1] along y-direction. By default, the ellipse is a unit circle (rx = ry = 1).

7.2.2.51 Expression

7.2.2.51.1 Node interface

Expression {

Field	SFInt32	**expression_select1**	0
Field	SFInt32	**expression_intensity1**	0

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

Field	SFInt32	**expression_select2**	0
Field	SFInt32	**expression_intensity2**	0
Field	SFBool	**init_face**	FALSE
Field	SFBool	**expression_def**	FALSE

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.51.2 Functionality and semantics

The **Expression** node is used to define the expression of the face as a combination of two expressions from the standard set of expressions defined ISO/IEC 14496-2, Annex C, Table C-3.

The **expression_select1** and **expression_select2** fields specify the expression types. The **expression_intensity1** and **expression_intensity2** fields specify the corresponding expression intensities.

If **init_face** is set, a neutral face may be modified before applying FAPs 1 and 3-68.

If **expression_def** is set, current FAPs are used to define an expression and store it.

7.2.2.52 Extrusion

7.2.2.52.1 Node interface

Extrusion {

EventIn	MFVec2f	**set_crossSection**	
EventIn	MFRotation	**set_orientation**	
EventIn	MFVec2f	**set_scale**	
EventIn	MFVec3f	**set_spine**	
Field	SFBool	**beginCap**	TRUE
Field	SFBool	**ccw**	TRUE
Field	SFBool	**convex**	TRUE
Field	SFFloat	**creaseAngle**	0.0
Field	MFVec2f	**crossSection**	1, 1, 1, -1, -1, -1, -1, 1, 1, 1
Field	SFBool	**endCap**	TRUE
Field	MFRotation	**orientation**	0, 0, 1, 0
Field	MFVec2f	**scale**	1, 1
Field	SFBool	**solid**	TRUE
Field	MFVec3f	**spine**	0, 0, 0, 0, 1, 0

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.52.2 Functionality and semantics

The semantics of the **Extrusion** node are specified in ISO/IEC 14772-1:1998, subclause 6.18.

7.2.2.53 Face

7.2.2.53.1 Node interface

Face {

exposedField	SFNode	**fit**	NULL
exposedField	SFNode	**fdp**	NULL
exposedField	SFNode	**fap**	NULL
exposedField	SFNode	**ttsSource**	NULL
exposedField	MFNode	**renderedFace**	NULL

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.53.2 Functionality and semantics

The **Face** node is used to define and animate a face in the scene. In order to animate the face with a facial animation stream, ut us necessary to link the **Face** node to a BIFS-Anim stream. The node shall be assigned a `nodeID`, through the DEF mechanism. Then, as for any BIFS-Anim stream, an animation mask is sent in the object descriptor of the BIFS-Anim stream (`specificInfo` field). The animation mask points to the **Face** node using its `nodeID`. The terminal shall then connect the facial animation decoder to the appropriate **Face** node.

The **FAP** field shall contain a **FAP** node, describing the facial animation parameters (FAPs). Each **Face** node shall contain a non-NULL **FAP** field.

The **FDP** field, which defines the particular look of a face by means of downloading the position of face definition points or an entire model, is optional. If the **FDP** field is not specified, the default face model of the terminal shall be used.

The **FIT** field, when specified, allows a set of FAPs to be defined in terms of another set of FAPs. When this field is non-NULL, the terminal shall use **FIT** to compute the maximal set of FAPs before using the FAPs to compute the mesh.

The **ttsSource** field shall only be non-NULL if the facial animation is to determine the facial animation parameters from an audio TTS source (see ISO/IEC 14496-3, subpart 6). In this case the **ttsSource** field shall contain an **AudioSource** node and the face shall be animated using the phonemes and bookmarks received from the TTS. See also subclause 7.7.

renderedFace is the scene graph of the face after it is rendered (all FAP's applied).

7.2.2.54 FaceDefMesh

7.2.2.54.1 Node interface

FaceDefMesh {

Field	SFNode	**faceSceneGraphNode**	NULL
Field	MFInt32	**intervalBorders**	[]
Field	MFInt32	**coordIndex**	[]
Field	MFVec3f	**displacements**	[]

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.54.2 Functionality and semantics

The **FaceDefMesh** node allows for the deformation of an **IndexedFaceSet** as a function of the amplitude of a FAP as specified in the related **FaceDefTable** node. The **FaceDefMesh** node defines the piece-wise linear motion trajectories for vertices of the **faceSceneGraphNode** field, which shall contain an **IndexedFaceSet** node. This **IndexedFaceSet** node belongs to the scenegraph of the **faceSceneGraph** field of the **FDP** node.

The **intervalBorders** field specifies interval borders for the piece-wise linear approximation in increasing order. Exactly one interval border shall have the value 0.

The **coordIndex** field shall contain a list of indices into the **Coordinate** node of the **IndexedFaceSet** node specified by the **faceSceneGraphNode** field.

For each vertex indexed in the **coordIndex** field, displacement vectors are given in the **displacements** field for the intervals defined in the **intervalBorders** field. There must be exactly (num(**intervalBorders**)-1)*num(**coordIndex**) values in this field.

In most cases, the animation generated by a FAP cannot be specified by updating a **Transform** node. Thus, a deformation of an **IndexedFaceSet** node needs to be performed. In this case, the **FaceDefTables** shall define which **IndexedFaceSets** are affected by a given FAP and how the **coord** fields of these nodes are updated. This is done by means of tables.

If a FAP affects an **IndexedFaceSet**, the **FaceDefMesh** shall specify a table of the following format for this **IndexedFaceSet**:

Table 17 — Vertex displacements

Vertex no.	1st Interval [I1, I2]	2nd Interval [I2, I3]	…
Index 1	Displacement D11	Displacement D12	…
Index 2	Displacement D21	Displacement D22	…

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Vertex no.	1st Interval [I1, I2]	2nd Interval [I2, I3]	…
…	…	…	…

Exactly one interval border I_k must have the value 0:

$$[I_1, I_2], [I_2, I_3], \ldots[I_{k-1}, 0], [0, I_{k+1}], [I_{k+1}, I_{k+2}], \ldots[I_{max-1}, I_{max}]$$

During animation, when the terminal receives a FAP, which affects one or more **IndexedFaceSets** of the face model, it shall piece-wise linearly approximate the motion trajectory of each vertex of the affected **IndexedFaceSets** by using the appropriate table.

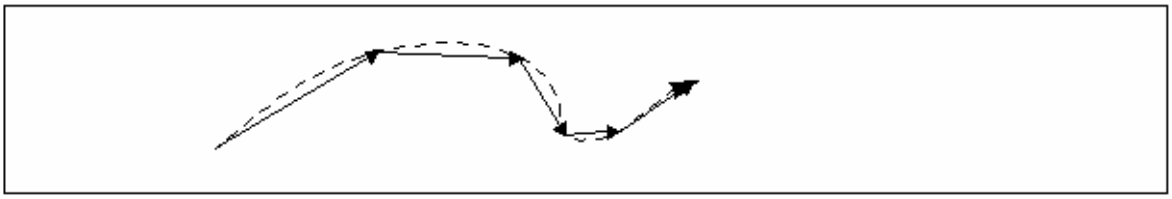


Figure 18 — An arbitrary motion trajectory is approximated as a piece-wise linear one.

If $\boldsymbol{P}_m$ is the position of the mth vertex in the **IndexedFaceSet** in neutral state (FAP = 0), $\boldsymbol{P'}_m$ the position of the same vertex after animation with the given FAP and $\boldsymbol{D_{mk}}$ the 3D displacement in the kth interval, the following algorithm shall be applied to determine the new position $\boldsymbol{P'}_m$.

Determine, in which of the intervals listed in the table the received FAP is lying.

If the received FAP is lying in the jth interval [Ij, Ij+1] and 0=Ik ≤ Ij, the new vertex position P'm of the mth vertex of the **IndexedFaceSet** is given by:

$$P'_m = FAPU * ((I_{k+1}-0) * D_{m,k} + (I_{k+2}-I_{k+1}) * D_{m,\,k+1} + \ldots + (I_j - I_{j-1}) * D_{m,\,j-1} + (FAP-I_j) * D_{m,\,j}) + P_m. \quad \text{(Eq. 1)}$$

If FAP > I_{max}, then P'_m is calculated by using equation Eq. 1 and setting the index j = max.

If the received FAP is lying in the jth interval [I_j, I_{j+1}] and $I_{j+1} \le I_k$=0, the new vertex position P'_m is given by:

$$P'_m = FAPU * ((I_{j+1} - FAP) * D_{m,\,j} + (I_{j+2} - I_{j+1}) * D_{m,\,j+1} + \ldots + (I_{k-1} - I_{k-2}) * D_{m,\,k-2} + (0 - I_{k-1}) * D_{m,\,k-1}) + P_m \quad \text{(Eq. 2)}$$

If FAP < I_1, then P'_m is calculated by using equation Eq. 1 and setting the index j+1 = 1.

If for a given FAP and **IndexedFaceSet** the table contains only one interval, the motion is strictly linear:

P'm = FAPU * FAP * Dm1 + Pm.

EXAMPLE —

```
FaceDefMesh {
   objectDescriptorID UpperLip
   intervalBorders [ -1000, 0, 500, 1000 ]
   coordIndex [ 50, 51]
   displacements [1 0 0,  0.9 0 0, 1.5 0 4, 0.8 0 0, 0.7 0 0, 2 0 0 ]
}
```

This **FaceDefMesh** defines the animation of the mesh "UpperLip". For the piecewise-linear motion function three intervals are defined: [-1000, 0], [0, 500] and [500, 1000]. Displacements are given for the vertices with the indices 50 and 51. The displacements for the vertex 50 are: (1 0 0), (0.9 0 0) and (1.5 0 4), the displacements for vertex 51 are (0.8 0 0), (0.7 0 0) and (2 0 0). Given a FAPValue of 600, the resulting displacement for vertex 50 would be:

displacement(vertex 50) = $500*(0.9\ 0\ 0)^T + 100 * (1.5\ 0\ 4)^T = (600\ 0\ 400)^T$.

If the FAPValue is outside the given intervals, the boundary intervals are extended to +I or -I, as appropriate.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.55 FaceDefTables

7.2.2.55.1 Node interface

FaceDefTables {

Field	SFInt32	**fapID**	0
Field	SFInt32	**highLevelSelect**	0
exposedField	MFNode	**faceDefMesh**	[]
exposedField	MFNode	**faceDefTransform**	[]

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.55.2 Functionality and semantics

The **FaceDefTables** node defines the behavior of a facial animation parameter FAP on a downloaded face model in **faceSceneGraph** by specifying the displacement vectors for moved vertices inside **IndexedFaceSet** objects as a function of the FAP **fapID** and/or specifying the value of a field of a **Transform** node as a function of FAP **fapID**.

The **FaceDefTables** node is transmitted directly after the BIFS bitstream of the **FDP** node. The **FaceDefTables** lists all FAPs that animate the face model. The FAPs animate the downloaded face model by updating the **Transform** or **IndexedFaceSet** nodes of the scene graph in **faceSceneGraph**. For each listed FAP, the **FaceDefTables** node describes which nodes are animated by this FAP and how they are animated. All FAPs that occur in the bitstream have to be specified in the **FaceDefTables** node. The animation generated by a FAP can be specified either by updating a **Transform** node (using a **FaceDefTransform**), or as a deformation of an **IndexedFaceSet** (using a **FaceDefMesh**).

The FAPUs shall be calculated by the terminal using the feature points that shall be specified in the FDP. The FAPUs are needed in order to animate the downloaded face model.

7.2.2.55.3 Semantics

The **fapID** field specifies the FAP, for which the animation behavior is defined in the **faceDefMesh** and **faceDefTransform** fields.

If **fapID** has value 1 or 2, the **highLevelSelect** field specifies the type of viseme or expression. In other cases this field has no meaning and shall be ignored.

The **faceDefMesh** field shall contain a **FaceDefMesh** node.

The **faceDefTransform** field shall contain a **FaceDefTransform** node.

7.2.2.56 FaceDefTransform

7.2.2.56.1 Node interface

FaceDefTransform {

Field	SFNode	**faceSceneGraphNode**	NULL
Field	SFInt32	**fieldId**	1
Field	SFRotation	**rotationDef**	0, 0, 1, 0
Field	SFVec3f	**scaleDef**	1, 1, 1
Field	SFVec3f	**translationDef**	0, 0, 0

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.56.2 Functionality and semantics

The **FaceDefTransform** node defines which field (**rotation**, **scale** or **translation**) of a **Transform** node (**faceSceneGraphNode)** of **faceSceneGraph** (defined in an **FDP** node) is updated by a facial animation parameter, and how the field is updated. If the face is in its neutral position, the **faceSceneGraphNode** has its **translation**, **scale**, and **rotation** fields set to the neutral values $(0,0,0)^T$, $(1,1,1)^T$, (0,0,1,0), respectively.

The **faceSceneGraphNode** field specifies the **Transform** node for which the animation is defined. The node shall be part of **faceScenegraph** as defined in the **FDP** node.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The **fieldId** field specifies which field in the **Transform** node, specified by the **faceSceneGraphNode** field, is updated by the FAP during animation. Possible fields are **translation**, **rotation**, **scale**.

If **fieldID**==1, **rotation** shall be updated using **rotationDef** and `FAPValue`.

If **fieldID**==2, **scale** shall be updated using **scaleDef** and `FAPValue`.

If **fieldID**==3, **translation** shall be updated using **translationDef** and `FAPValue`.

The **rotationDef** field is of type SFRotation. With **rotationDef**=(r_x,r_y,r_z,θ), the new value of the **rotation** field of the **Transform** node **faceSceneGraphNode** is:

rotation: **=**(r_x,r_y,r_z,θ*`FAPValue`*AU) [AU is defined in ISO/IEC 14496-2]

The **scaleDef** field is of type SFVec3f. The new value of the **scale** field of the **Transform** node **faceSceneGraphNode** is:

scale:= `FAPValue`***scaleDef**

The **translationDef** field is of type SFVec3f. The new value of the **translation** field of the **Transform** node **faceSceneGraphNode** is:

translation:= `FAPValue`***translationDef**

7.2.2.57 FAP

7.2.2.57.1 Node interface

FAP {

exposedField	SFNode	**viseme**	NULL
exposedField	SFNode	**expression**	NULL
exposedField	SFInt32	**open_jaw**	+I
exposedField	SFInt32	**lower_t_midlip**	+I
exposedField	SFInt32	**raise_b_midlip**	+I
exposedField	SFInt32	**stretch_l_corner**	+I
exposedField	SFInt32	**stretch_r_corner**	+I
exposedField	SFInt32	**lower_t_lip_lm**	+I
exposedField	SFInt32	**lower_t_lip_rm**	+I
exposedField	SFInt32	**lower_b_lip_lm**	+I
exposedField	SFInt32	**lower_b_lip_rm**	+I
exposedField	SFInt32	**raise_l_cornerlip**	+I
exposedField	SFInt32	**raise_r_cornerlip**	+I
exposedField	SFInt32	**thrust_jaw**	+I
exposedField	SFInt32	**shift_jaw**	+I
exposedField	SFInt32	**push_b_lip**	+I
exposedField	SFInt32	**push_t_lip**	+I
exposedField	SFInt32	**depress_chin**	+I
exposedField	SFInt32	**close_t_l_eyelid**	+I
exposedField	SFInt32	**close_t_r_eyelid**	+I
exposedField	SFInt32	**close_b_l_eyelid**	+I
exposedField	SFInt32	**close_b_r_eyelid**	+I
exposedField	SFInt32	**yaw_l_eyeball**	+I
exposedField	SFInt32	**yaw_r_eyeball**	+I
exposedField	SFInt32	**pitch_l_eyeball**	+I
exposedField	SFInt32	**pitch_r_eyeball**	+I
exposedField	SFInt32	**thrust_l_eyeball**	+I
exposedField	SFInt32	**thrust_r_eyeball**	+I
exposedField	SFInt32	**dilate_l_pupil**	+I
exposedField	SFInt32	**dilate_r_pupil**	+I
exposedField	SFInt32	**raise_l_i_eyebrow**	+I
exposedField	SFInt32	**raise_r_i_eyebrow**	+I
exposedField	SFInt32	**raise_l_m_eyebrow**	+I
exposedField	SFInt32	**raise_r_m_eyebrow**	+I
exposedField	SFInt32	**raise_l_o_eyebrow**	+I

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

exposedField	SFInt32	**raise_r_o_eyebrow**	+I
exposedField	SFInt32	**squeeze_l_eyebrow**	+I
exposedField	SFInt32	**squeeze_r_eyebrow**	+I
exposedField	SFInt32	**puff_l_cheek**	+I
exposedField	SFInt32	**puff_r_cheek**	+I
exposedField	SFInt32	**lift_l_cheek**	+I
exposedField	SFInt32	**lift_r_cheek**	+I
exposedField	SFInt32	**shift_tongue_tip**	+I
exposedField	SFInt32	**raise_tongue_tip**	+I
exposedField	SFInt32	**thrust_tongue_tip**	+I
exposedField	SFInt32	**raise_tongue**	+I
exposedField	SFInt32	**tongue_roll**	+I
exposedField	SFInt32	**head_pitch**	+I
exposedField	SFInt32	**head_yaw**	+I
exposedField	SFInt32	**head_roll**	+I
exposedField	SFInt32	**lower_t_midlip_o**	+I
exposedField	SFInt32	**raise_b_midlip_o**	+I
exposedField	SFInt32	**stretch_l_cornerlip**	+I
exposedField	SFInt32	**stretch_r_cornerlip_o**	+I
exposedField	SFInt32	**lower_t_lip_lm_o**	+I
exposedField	SFInt32	**lower_t_lip_rm_o**	+I
exposedField	SFInt32	**raise_b_lip_lm_o**	+I
exposedField	SFInt32	**raise_b_lip_rm_o**	+I
exposedField	SFInt32	**raise_l_cornerlip_o**	+I
exposedField	SFInt32	**raise_r_cornerlip_o**	+I
exposedField	SFInt32	**stretch_l_nose**	+I
exposedField	SFInt32	**stretch_r_nose**	+I
exposedField	SFInt32	**raise_nose**	+I
exposedField	SFInt32	**bend_nose**	+I
exposedField	SFInt32	**raise_l_ear**	+I
exposedField	SFInt32	**raise_r_ear**	+I
exposedField	SFInt32	**pull_l_ear**	+I
exposedField	SFInt32	**pull_r_ear**	+I

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.57.2 Functionality and semantics

This node defines the current look of the face by means of expressions and FAPs and gives a hint to TTS controlled systems on which viseme to use. For a definition of the facial animation parameters see ISO/IEC 14496-2, Annex C.

The **viseme** field shall contain a **Viseme** node.

The **expression** field shall contain an **Expression** node.

The semantics for the remaining fields are described in the ISO/IEC 14496-2, Annex C and in particular in Table C-1.

A FAP of value +I shall be interpreted as indicating that the particular FAP is uninitialized.

7.2.2.58 FDP

7.2.2.58.1 Node interface

FDP {

exposedField	SFNode	**featurePointsCoord**	NULL
exposedField	SFNode	**textureCoords**	NULL
exposedField	SFBool	**useOrthoTexture**	FALSE
exposedField	MFNode	**faceDefTables**	[]
exposedField	MFNode	**faceSceneGraph**	[]

}

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.58.2 Functionality and semantics

The **FDP** node defines the face model to be used at the terminal. Two options are supported:

1. If **faceDefTables** is NULL, calibration information is downloaded, so that the proprietary face of the terminal can be calibrated using facial feature points and, optionally, the texture information. In this case, the **featurePointsCoord** field shall be set. **featurePointsCoord** contains the coordinates of facial feature points, as defined in ISO/IEC 14496-2, Annex C, Figure C-1, corresponding to a neutral face. If a coordinate of a feature point is set to +I, the coordinates of this feature point shall be ignored. The **textureCoord** field, if set, is used to map a texture on the model calibrated by the feature points. The **textureCoord** points correspond to the feature points. Tthat is, each defined feature point shall have corresponding texture coordinates. In this case, the **faceSceneGraph** shall contain exactly one texture image, and any geometry it might contain shall be ignored. The terminal shall interpret the feature points, texture coordinates, and the **faceSceneGraph** in the following way:

   - Feature points of the terminal's face model shall be moved to the coordinates of the feature points supplied in **featurePointsCoord**, unless a feature point is to be ignored, as explained above.

   - If **textureCoord** is set, the texture supplied in the **faceSceneGraph** shall be mapped onto the terminal's default face model. The texture coordinates are derived from the texture coordinates of the feature points supplied in **textureCoords**. The **useOrthoTexture** field provides a hint to the decoding terminal that, when FALSE, indicates that the texture image is best obtained by cylindrical projection of the face. If **useOrthoTexture** is TRUE, the texture image is best obtained by orthographic projection of the face.

2. A face model as described in the **faceSceneGraph** is decoded. This face model replaces the terminal's default face model in the terminal. The **faceSceneGraph** shall contain the face in its neutral position (all FAPs = 0). If desired, the **faceSceneGraph** shall contain the texture maps of the face. The functions defining the way in which the **faceSceneGraph** shall be modified, as a function of the FAPs, shall also be decoded. This information is described by **faceDefTables** that define how the **faceSceneGraph** is to be modified as a function of each FAP. By means of **faceDefTables**, **IndexedFaceSets** and **Transform** nodes of the **faceSceneGraph** can be animated. Since the amplitude of FAPs is defined in units that are dependent on the size of the face model, the **featurePointsCoord** field defines the position of facial features on the surface of the face described by **faceSceneGraph**. From the location of these feature points, the terminal computes the units of the FAPs. Generally, only two node types in the scene graph of a decoded face model are affected by FAPs: **IndexedFaceSet** and **Transform** nodes. If a FAP causes a deformation of an object (e.g. lip stretching), then the coordinate positions in the affected **IndexedFaceSets** shall be updated. If a FAP causes a movement which can be described with a **Transform** node (e.g. FAP 23, yaw_l_eyeball), then the appropriate fields in this **Transform** node shall be updated. It shall be assumed that this **Transform** node has its **rotation**, **scale**, and **translation** fields set to neutral values if the face is in its neutral position. A unique `nodeId` shall be assigned via the DEF statement to all **IndexedFaceSet** and **Transform** nodes which are affected by FAPs so that they can be accessed unambiguously during animation.

The **featurePointsCoord** field shall contain a **Coordinate** node that specifies feature points for the calibration of the terminal's default face. The coordinates are specified in the **point** field of the **Coordinate** node in the prescribed order, that a feature point with a lower label number is listed before a feature point with a higher label naumber.

EXAMPLE — Feature point 3.14 before feature point 4.1

The **textureCoords** field shall contain a **Coordinate** node that specifies texture coordinates for the feature points. The coordinates are listed in the **point** field in the **Coordinate** node in the prescribed order, that a feature point with a lower label is listed before a feature point with a higher label.

The **useOrthoTexture** field may contain a hint to the terminal as to the type of texture image, in order to allow better interpolation of texture coordinates for the vertices that are not feature points. If **useOrthoTexture** is FALSE, the terminal may assume that the texture image was obtained by cylindrical projection of the face. If **useOrthoTexture** is 1, the terminal may assume that the texture image was obtained by orthographic projection of the face.

The **faceDefTables** field shall contain **FaceDefTables** nodes. The behavior of FAPs is defined in this field for the face in **faceSceneGraph**.

The **faceSceneGraph** field shall contain a **Group** node. In the case of option 1 (above), this may be used to contain a texture image as described above. In the case of option 2, this shall be the grouping node for the face model rendered in the compositor and shall contain the face model. In this case, the effect of facial animation parameters is defined in the **faceDefTables** field.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.59 FIT

7.2.2.59.1 Node interface

```
FIT {
    exposedField    MFInt32     FAPs                []
    exposedField    MFInt32     Graph               []
    exposedField    MFInt32     numeratorTerms      []
    exposedField    MFInt32     denominatorTerms    []
    exposedField    MFInt32     numeratorExp        []
    exposedField    MFInt32     denominatorExp      []
    exposedField    MFInt32     numeratorImpulse    []
    exposedField    MFFloat     numeratorCoefs      []
    exposedField    MFFloat     denominatorCoefs    []
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.59.2 Functionality and semantics

The **FIT** node allows a smaller set of FAPs to be sent during a facial animation. This small set can then be used to determine the values of other FAPs, using a rational polynomial mapping between parameters. In a **FIT** node, rational polynomials are used to specify interpolation functions.

EXAMPLE — The top inner lip FAPs can be sent and then used to determine the top outer lip FAPs. Another example is that only viseme and/or expression FAPs are sent to drive the face. In this case, low-level FAPs are interpolated from these two high-level FAPs.

To make the scheme general, sets of FAPs are specified, along with a FAP interpolation graph (FIG) between the sets that specifies which sets are used to determine which other sets. The FIG is a graph with directed links. Each node contains a set of FAPs. Each link from a parent node to a child node indicates that the FAPs in the child node can be interpolated from the parent node. **Expression** (FAP#1) or **Viseme** (FAP #2) and their fields shall not be interpolated from other FAPs.

In a FIG, a FAP may appear in several nodes, and a node may have multiple parents. For a node that has multiple parent nodes, the parent nodes are ordered as 1st parent node, 2nd parent node, etc. During the interpolation process, if this child node needs to be interpolated, it is first interpolated from 1st parent node if all FAPs in that parent node are available. Otherwise, it is interpolated from 2nd parent node, and so on.

An example of FIG is shown in Figure 19. Each node has a `nodeID`. The numerical label on each incoming link indicates the order of these links.

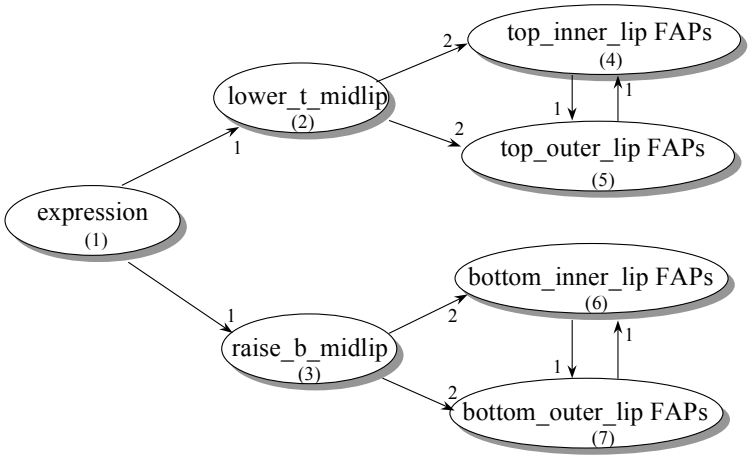


Figure 19 — A FIG example

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The interpolation process based on the FAP interpolation graph is described using pseudo-C code as follows:

```
do {
   interpolation_count = 0;
   for (all Node_i) {   // from Node_1 to Node_N
      for (ordered Node_i's parent Node_k) {
         if (FAPs in Node_i need interpolation and
                FAPs in Node_k have been interpolated or are available) {
            interpolate Node_i from Node_k;  //using interpolation function
                                             // table here
            interpolation_count ++;
            break;
         }
      }
   }
} while (interpolation_count != 0);
```

Each directed link in a FIG is a set of interpolation functions. Suppose F_1, F_2, …, F_n are the FAPs in a parent set and f_1, f_2, …, f_m are the FAPs in a child set.

Then, there are m interpolation functions denoted as:

$f_1 = I_1(F_1, F_2, \ldots, F_n)$

$f_2 = I_2(F_1, F_2, \ldots, F_n)$

…

$f_m = I_m(F_1, F_2, \ldots, F_n)$

Each interpolation function I_k () is in a rational polynomial form if the parent node does not contain viseme FAP or expression FAP.

$$I(F_1, F_2, ..., F_n) = \sum_{i=0}^{K-1} (c_i \prod_{j=1}^{n} F_j^{l_{ij}}) \Big/ \sum_{i=0}^{P-1} (b_i \prod_{j=1}^{n} F_j^{m_{ij}})$$

Otherwise, an impulse function is added to each numerator polynomial term to allow selection of expression or viseme.

$$I(F_1, F_2, ..., F_n) = \sum_{i=0}^{K-1} \delta(F_{s_i} - a_i)(c_i \prod_{j=1}^{n} F_j^{l_{ij}}) \Big/ \sum_{i=0}^{P-1} (b_i \prod_{j=1}^{n} F_j^{m_{ij}})$$

In both equations, K and P are the numbers of polynomial products, c_i and b_i are the coefficient of the *i*th product. l_{ij} and m_{ij} are the power of F_j in the *i*th product. An impulse function equals 1 when $F_{s_i} = a_i$, otherwise, equals 0. F_{s_i} can only be viseme_select1, viseme_select2, expression_select1, and expression_select2. a_i is an integer that ranges from 0 to 6 when F_{s_i} is expression_select1 or expression_select2, ranges 0 to 14 when F_{s_i} is viseme_select1 or viseme_select2. The encoder shall send an interpolation function table which contains K, P, a_i, s_i, c_i, b_i, l_{ij}, m_{ij} to the terminal.

To aid in the explanation below, it is assumed that there are N different sets of FAPs with index 1 to N, and that each set has n_i, i=1,..,N parameters. It is also assumed that there are L directed links in the FIG and that each link points from the FAP set with index P_i to the FAP set with index C_i, for i = 1, .. , L

The FAPs field shall contain a list of FAP-indices specifying which animation parameters form sets of FAPs. Each set of FAP indices is terminated by –1. There shall be a total of $N + n_1 + n_2 + \ldots + n_N$ numbers in this field, with N of them being –1. FAP#1 to FAP#68 are of indices 1 to 68. Fields of the **Viseme** FAP (FAP#1), namely, **viseme_select1**, **viseme_select2**, **viseme_blend**, are of indices from 69 to 71. Fields of the **Expression** FAP (FAP#2), namely, **expression_select1**, **expression_select2**, **expression_intensity1**, **expression_intensity2** are of indices from 72 to 75. When the parent node contains a **Viseme** FAP, three indices, 69, 70, 71, shall be included in the node (but not index 1). When a parent node contains an **Expression** FAP, four indices, 72,73,74,75, shall be included in the node (but not index 2).

The **Graph** field shall contain a list of pairs of integers, specifying a directed links between sets of FAPs. The integers refer to the indices of the sets specified in the **FAPs** field, and thus range from 1 to N. When more than one direct link terminates at the same set, that is, when the second value in the pair is repeated, the links have precedence determined

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

by their order in this field. This field shall have a total of 2L numbers, corresponding to the directed links between the parents and children in the FIG.

The **numeratorTerms** field shall be a list containing the number of terms in the polynomials of the numerators of the rational functions used to interpolae parameter values. Each element in the list corresponds to *K* in equation 1 above). Each link i (that is, the *i*th integer pair) in the **Graph** field must have n_{Ci} values specified, one for each child FAP. The order in the **numeratorTerms** list shall correspond to the order of the links in the **Graph** field and the order that the child FAP appears in the **FAPs** field. There shall be $n_{C1} + n_{C2} + \ldots + n_{CL}$ numbers in this field.

The **denominatorTerms** field shall contain a list of the number of terms in the polynomials of the denominator of the rational functions controlling the parameter value. Each element in the list corresponds to *P* in equation 1. Each link i (that is, the *i*th integer pair) in the **Graph** field must have n_{Ci} values specified, one for each child FAP. The order in the **denominatorTerms** list corresponds to the order of the links in the **Graph** field and the order that the child FAP appears in the **FAPs** field. There shall be $n_{C1} + n_{C2} + \ldots + n_{CL}$ numbers in this field.

The **numeratorImpulse** field shall contain a list of impulse functions in the numerator of the rational function for links with the **Viseme** or **Expression** FAP in parent node. This list corresponds to the $\delta(F_{s_i} - a_i)$. Each entry in the list is (s_i, a_i).

The **numeratorExp** field shall contain a list of exponents of the polynomial terms in the numerator of the rational function controlling the parameter value. This list corresponds to l_{ij}. For each child FAP in each link i, n_{Pi}*K values need to be specified. The order in the **numeratorExp** list shall correspond to the order of the links in the **Graph** field and the order that the child FAP appears in the **FAPs** field.

NOTE — K may be different for each child FAP.

The **denominatorExp** field shall contain a list of exponents of the polynomial terms of the denominator of the rational function controlling the parameter value. This list corresponds to m_{ij}. For each child FAP in each link i, n_{Pi}*P values need to be specified. The order in the **denominatorExp** list shall correspond to the order of the links in the **Graph** field and the order that the child FAP appears in the **FAPs** field.

NOTE — P may be different for each child FAP.

The **numeratorCoefs** field shall contain a list of coefficients of the polynomial terms of the numerator of the rational function controlling the parameter value. This list corresponds to c_i. The list shall have *K* terms for each child parameter that appears in a link in the FIG, with the order in **numeratorCoefs** corresponding to the order in **Graph** and **FAPs**.

NOTE — K is dependent on the polynomial, and is not a fixed constant.

The **denominatorCoefs** field shall contain a list of coefficients of the polynomial terms in the numerator of the rational function controlling the parameter value. This list corresponds to b_i. The list shall have *P* terms for each child parameter that appears in a link in the FIG, with the order in **denominatorCoefs** corresponding to the order in **Graph** and **FAPs**.

NOTE — P is dependent on the polynomial, and is not a fixed constant.

EXAMPLE — Suppose a FIG contains four nodes and 2 links. Node 1 contains FAP#3, FAP#3, FAP#5. Node 2 contains FAP#6, FAP#7. Node 3 contains an expression FAP, which means contains FAP#72, FAP#73, FAP#74, and FAP#75. Node 4 contains FAP#12 and FAP#17. Two links are from node 1 to node 2, and from node 3 to node 4. For the first link, the interpolation functions are

$$F_6 = (F_3 + 2F_4 + 3F_5 + 4F_3F_4^2)/(5F_5 + 6F_3F_4F_5)$$

$$F_7 = F_4.$$

For the second link, the interpolation functions are

$$F_{12} = \delta(F_{72} - 6)(0.6F_{74}) + \delta(F_{73} - 6)(0.6F_{75})$$

$$F_{17} = \delta(F_{72} - 6)(-1.5F_{74}) + \delta(F_{73} - 6)(-1.5F_{75}).$$

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The second link simply says that when the expression is surprise (FAP#72=6 or FAP#73=6), for FAP#12, the value is 0.6 times of expression intensity FAP#74 or FAP#75; for FAP#17, the value is –1.5 tims of FAP#74 or FAP#75.

After the FIT node given below, we explain each field separately.

```
FIT {
   FAPs            [ 3 4 5 -1 6 7 -1 72 73 74 75 -1 12 17 -1]
   Graph           [ 1 2 3 4]
   numeratorTerms  [ 4 1 2 2 ]
   denominatorTerms  [2 1 1 1]
   numeratorExp    [1 0 0   0 1 0   0 0 1  1 2 0  0 1 0
                      0 0 1 0  0 0 0 1  0 0 1 0  0 0 0 1 ]
   denominatorExp  [ 0 0 1  1 1 1  0 0 0
                       0 0 0 0  0 0 0 0 ]
   numeratorImpulse  [ 72 6  73 6  72 6  73 6 ]
   numeratorCoefs  [1 2 3 4   1   0.6 0.6  -1.5 -1.5 ]
   denominatorCoefs  [5 6 1 1 1 ]
}
```

`FAPs [ 3 4 5 -1 6 7 -1 72 73 74 75 -1 12 17 -1]`
Four sets of FAPs are defined, the first with FAPs number 3, 4, and 5, the second with FAPs number 6 and 7, the third with FAPs number 72, 73, 74, 75, and the fourth with FAPs number 12, 17.

`Graph  [ 1 2 3 4]`
The first set is made to be the parent of the second set, so that FAPs number 6 and 7 will be determined by FAPs 3, 4, and 5. Also, the third set is made to be the parent of the fourth set, so that FAPs number 12 and 17 will be determined by FAPs 72, 73, 74, and 75.

`numeratorTerms [ 4 1 2 2]`
The rational functions that define F6 and F7 are selected to have 4 and 1 terms in their numerator, respectively. Also, the rational functions that define F12 and F17 are selected to have 2 and 2 terms in their numerator, respectively.

`denominatorTerms [ 2 1 1 1]`
The rational functions that define F6 and F7 are selected to have 2 and 1 terms in their denominator, respectively. Also, the rational functions that define F12 and F17 are selected to both have 1 term in their denominator.

`numeratorExp [ 1 0 0  0 1 0  0 0 1  1 2 0  0 1 0  0 0 1 0  0 0 0 1  0 0 1 0  0 0 0 1]`
The numerator selected for the rational function defining F6 is F3 + 2F4 + 3 F5 + 4F3F42. There are 3 parent FAPs, and 4 terms, leading to 12 exponents for this rational function. For F7, the numerator is just F4, so there are three exponents only (one for each FAP). Values for F12 and F17 are derived in the same way.

`denominatorExp [ 0 0 1  1 1 1  0 0 0  0 0 0 0  0 0 0 0]`
The denominator selected for the rational function defining F6 is 5F5+ 6F3F4F5 , so there are 3 parent FAPs and 2 terms and hence, 6 exponents for this rational function. For F7, the denominator is just 1, so there are three exponents only (one for each FAP). Values for F12 and F17 are derived in the same way.

`numeratorImpulse [72 6  73 6  72 6  73 6]`
For the second link, all four numerator polynomial terms contain impulse function $\delta(F_{72}-6)$ or $\delta(F_{73}-6)$.

`numeratorCoefs [ 1 2 3 4  1   0.6 0.6  -1.5 -1.5]`
There is one coefficient for each term in the numerator of each rational function.

`denominatorCoefs [ 5 6   1   1 1]`
There is one coefficient for each term in the denominator of each rational function.

7.2.2.60 Fog

7.2.2.60.1 Node interface

Fog {

exposedField	SFColor	**color**	1 1 1
exposedField	SFString	**fogType**	"LINEAR"
exposedField	SFFloat	**visibilityRange**	0.0

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    eventIn      SFBool      set_bind
    eventOut     SFBool      isBound
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.60.2 Functionality and semantics

The semantics of the **Fog** node are specified in ISO/IEC 14772-1:1998, subclause 6.19.

7.2.2.61 FontStyle

7.2.2.61.1 Node interface

```
FontStyle {
    exposeField   MFString   family          ["SERIF"]
    exposeField   SFBool     horizontal      TRUE
    exposeField   MFString   justify         ["BEGIN"]
    exposeField   SFString   language        ""
    exposeField   SFBool     leftToRight     TRUE
    exposeField   SFFloat    size            1.0
    exposeField   SFFloat    spacing         1.0
    exposeField   SFString   style           "PLAIN"
    exposeField   SFBool     topToBottom     TRUE
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.61.2 Functionality and semantics

The semantics of the **FontStyle** node are specified in ISO/IEC 14772-1:1998, subclause 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.

EXAMPLE:

```
fontStyle FontStyle {
   family ["OD:104;FSID:33"]
   size 25.0
   style "BOLD"
}
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.62 Form

7.2.2.62.1 Node interface

```
Form {
    eventIn         MFNode      addChildren
    eventIn         MFNode      removeChildren
    exposedField    MFNode      children        []
    exposedField    SFVec2f     size            -1, -1
    exposedField    MFInt32     groups          []
    exposedField    MFString    constraints     []
    exposedField    MFInt32     groupsIndex     []
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.62.2 Functionality and semantics

The **Form** node specifies the placement of its children according to relative alignment and distribution constraints. Distribution spreads objects regularly, with an equal spacing between them.

The **children** field shall specify a list of nodes that are to be arranged. The children's position is implicit and order is important.

The **size** field specifies the width and height of the layout frame.

The **groups** field specifies the list of groups of objects on which the constraints can be applied. The children of the **Form** node are numbered from 1 to n, 0 being reserved for a reference to the form itself. A group is a list of child indices, terminated by a -1.

The **constraints** and the **groupsIndex** fields specify the list of constraints. One constraint is constituted by a constraint type from the **constraints** field, coupled with a set of group indices terminated by a –1 contained in the **groupsIndex** field. There shall be as many strings in **constraints** as there are –1-terminated sets in **groupsIndex**. The n-th constraint string shall be applied to the n-th set in the **groupsIndex** field. A value of 0 in the **groupsIndex** field references the form node itself, otherwise a **groupsIndex** field value is a 1-based index into the **group** field.

Constraints belong to two categories: alignment and distribution constraints.

Groups referred to in the tables below are groups whose indices appear in the list following the constraint type. When rank is mentioned, it refers to the rank in that list.

The semantics of the <s>, when present in the name of a constraint, is the following. It shall be a number, integer when the scene uses pixel metrics, and float otherwise, which specifies the space mentioned in the semantics of the constraint.

In case the form itself is specified in alignment constraint (group index 0), the form rectangle shall be used as the base of the alignment computation and other groups in the constraint list shall be aligned as specified by the constraint

Table 18 — Alignment Constraints

Alignment Constraints	Type Index	Effect
AL: Align Left edges	"AL"	The xmin of constrained groups becomes equal to the xmin of the left-most group.
AH: Align centers Horizontally	"AH"	The (xmin+xmax)/2 of constrained groups becomes equal to the (xmin+xmax)/2 of the group of constrained groups as computed before this constraint is applied.
AR: Align Right edges	"AR"	The xmax of constrained groups becomes equal to the xmax of the right-most group.
AT: Align Top edges	"AT"	The ymax of all constrained groups becomes equal to the ymax of the top-most group.
AV: Align centers Vertically	"AV"	The (ymin+ymax)/2 of constrained groups becomes equal to the (ymin+ymax)/2 of the group of constrained groups as computed before this constraint is applied.
AB: Align Bottom edges	"AB"	The ymin of constrained groups becomes equal to the ymin of the

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

		bottom-most group.
ALspace: Align Left edges by specified space	“AL <s>”	The xmin of the second and following groups become equal to the xmin of the first group plus the specified space.
ARspace: Align Right edges by specified space	“AR <s>”	The xmax of the second and following groups becomes equal to the xmax of the first group minus the specified space.
ATspace: Align Top edges by specified space	“AT <s>”	The ymax of the second and following groups becomes equal to the ymax of the first group minus the specified space.
ABspace: Align Bottom edges by specified space	“AB <s>”	The ymin of the second and following groups become equal to the ymin of the first group plus the specified space.

The purpose of distribution constraints is to specify the space between groups, by making such pairwise gaps equal either to a given value or to the effect of filling available space.

Table 19 — Distribution Constraints

Distribution Constraints	Type Index	Effect
SH: Spread Horizontally	“SH”	The differences between the xmin of each group and the xmax of the previous one all become equal. The first and the last group shall be constrained horizontally already.
SHin: Spread Horizontally in container	“SHin”	The differences between the xmin of each group and the xmax of the previous one all become equal. References are the edges of the layout.
SHspace: Spread Horizontally by specified space	“SH <s>”	The difference between the xmin of each group and the xmax of the previous one all become equal to the specified space. The first group is not moved.
SV: Spread Vertically	“SV”	The differences between the ymin of each group and the ymax of the previous one all become equal. The first and the last group shall be constrained vertically already.
SVin: Spread Vertically in container	“SVin”	The differences between the ymin of each group and the ymax of the previous one all become equal. References are the edges of the layout.
SVspace: Spread Vertically by specified space	“SV <s>”	The difference between the ymin of each group and the ymax of the previous one all become equal to the specified space. The first group is not moved.

All objects start at the center of the **Form**. The constraints are then applied in sequence.

EXAMPLE — Laying out five 2D objects.

```
Shape {
   geometry Rectangle { size 50 55 } // draw the Form's frame.
   appearance use AppRect
}

Transform2D {
   translation 10 10
      children [
         Form {
            children [
               Shape { use OBJ1 }
               Shape { use OBJ2 }
               Shape { use OBJ3 }
               Shape { use OBJ4 }
               Shape { use OBJ5 }
            ]
            size 50 55
            groups [ 1 -1 2 -1 3 -1 4 -1 5 -1 1 3 -1]
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            constraints ["SH" "SV" "AR" "AB" "AB 6"
                         "AB 7" "AL 7" "AT -2" "AR -2"]
            groupsIndex  [6 -1 1 -1 0 2 -1 0 2 -1 0 3 -1
                          0 4 -1 0 4 -1 0 5 -1 0 5 -1]
        }
      ]
    }
}
```

The above **constraints** specify the following operations:

spread group 6 (objects 1 and 3) horizontally in container (object 0)

spread group 1 (object 1) vertically in container

align the right edges of groups 0 (container) and 2 (object 2)

align the bottom edges of the container and group 2 (object 2)

align the bottom edges of the container and group 3 (object 3) with spacing of size 6

align the bottom edges of the container and group 4 (object 4) with spacing of size 7

align the left edges of the container and group 4 (object 4) with spacing of size 7

align the top edges of the container and group 5 (object 5) with spacing size of -2

align the right edges of the container and group 5 (object 5) with spacing size of -2

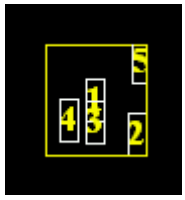


Figure 20 — Visual result of the Form node example

7.2.2.63 Group

7.2.2.63.1 Node interface

Group {

eventIn	MFNode	**addChildren**
eventIn	MFNode	**removeChildren**
exposedField	MFNode	**children** []

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.63.2 Functionality and semantics

The semantics of the **Group** node are specified in ISO/IEC 14772-1:1998, subclause 6.21. ISO/IEC 14496-1 does not support the bounding box parameters (**bboxCenter** and **bboxSize**).

Where multiple sub-graphs containing audio content (i.e. **Sound** nodes) occur as children of a **Group** node, the sounds shall be combined as described in 7.2.2.116.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.64 Hierarchical3Dmesh

7.2.2.64.1 Node Interface

Hierarchical3DMesh {

eventIn	SFInt32	**triangleBudget**	
exposedField	SFFloat	**level**	
field	MFString	**url**	[]
eventOut	SFBool	**doneLoading**	

}

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.64.2 Functionality and Semantics

The **Hierarchical3DMesh** is used to represent multi-resolution polygonal models with multiple levels of detail (LOD), smooth transition (interpolation) between consecutive levels, and hierarchical transmission through an independent elementary stream encoded with the 3D Mesh Coding tools (see ISO/IEC 14496-2:2004). The implementation of the **Hierarchical3DMesh** requires two execution threads, the *decoder* thread, and the *player* thread.

The decoder thread decodes the compressed 3D Mesh bitstream from the elementary stream specified in the **url** field, and reconstructs the LOD hierarchy and the information necessary to implement the smooth transition property in internal data structures. How the LOD hierarchy is stored in the internal data structures, and whether all or a subset of the transmitted hierarchy is stored for player interaction is implementation-dependent.

The decoder thread is started immediately after instantiation. Once this thread finishes decoding the compressed 3D Mesh bitstream, it sends a **done_loading** eventOut with the value TRUE to the player, and dies.

The **Hierarchical3DMesh** is seen by the player as a read-only **IndexedFaceSet** node. That is, the player has access to the following fields for rendering purposes, but they can neither be explicitly instantiated, nor modified by routing events into them:

field SFNode color
field SFNode coord
field SFNode normal
field SFNode texCoord
field SFBool ccw
field MFInt32 colorIndex
field SFBool colorPerVertex
field SFBool convex
field MFInt32 coordIndex
field SFFloat creaseAngle
field MFInt32 normalIndex
field SFBool normalPerVertex
field SFBool solid
field MFInt32 texCoordIndex

The player thread is responsible for switching levels of detail responding to the **set_level** and **triangleBudget** eventIn events sent by the player. It does so by modifying the fields of the **IndexedFaceSet** seen by the player from information stored in the internal data structures build by the decoder thread.

The **level** exposedField (between 0 and 1) is used to (1) set a particular fractional level, (2) query the current level, (3) as an eventOut to notify the browser when a level was actually set and which level it is.

Optionally, the player can set the level of detail by sending a **triangleBudget** eventIn to the node. The value of the **triangleBudget** eventIn represents the desired number of triangles that the player assigns to the node. The node must select a level of detail that best matches the given budget.

7.2.2.65 ImageTexture

7.2.2.65.1 Node interface

ImageTexture {

exposedField	MFString	**url**	[]

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

field	SFBool	**repeatS**	TRUE
field	SFBool	**repeatT**	TRUE

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.65.2 Functionality and semantics

The semantics of the **ImageTexture** node are specified in ISO/IEC 14772-1:1998, subclause 6.22.

The **url** field specifies the data source to be used (see 7.1.1.2.7.1).

7.2.2.66 IndexedFaceSet

7.2.2.66.1 Node interface

IndexedFaceSet {

eventIn	MFInt32	**set_colorIndex**	
eventIn	MFInt32	**set_coordIndex**	
eventIn	MFInt32	**set_normalIndex**	
eventIn	MFInt32	**set_texCoordIndex**	
exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL
exposedField	SFNode	**normal**	NULL
exposedField	SFNode	**texCoord**	NULL
field	SFBool	**ccw**	TRUE
field	MFInt32	**colorIndex**	[]
field	SFBool	**colorPerVertex**	TRUE
field	SFBool	**convex**	TRUE
field	MFInt32	**coordIndex**	[]
field	SFFloat	**creaseAngle**	0.0
field	MFInt32	**normalIndex**	[]
field	SFBool	**normalPerVertex**	TRUE
field	SFBool	**solid**	TRUE
field	MFInt32	**texCoordIndex**	[]

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.66.2 Functionality and semantics

The semantics of the **IndexedFaceSet** node are specified in ISO/IEC 14772-1:1998, subclause 6.23. Some restrictions on these semantics are described below.

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 subclause 8.5.3. 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

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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 subclause 7.8 (3D Mesh Coding in BIFS scenes).

7.2.2.67 IndexedFaceSet2D

7.2.2.67.1 Node interface

IndexedFaceSet2D {			
eventIn	MFInt32	**set_colorIndex**	
eventIn	MFInt32	**set_coordIndex**	
eventIn	MFInt32	**set_texCoordIndex**	
exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL
exposedField	SFNode	**texCoord**	NULL
field	MFInt32	**colorIndex**	[]
field	SFBool	**colorPerVertex**	TRUE
field	SFBool	**convex**	TRUE
field	MFInt32	**coordIndex**	[]
field	MFInt32	**texCoordIndex**	[]
}			

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.67.2 Functionality and semantics

The **IndexedFaceSet2D** node is the 2D equivalent of the **IndexedFaceSet** node as defined in 7.2.2.66. The **IndexedFaceSet2D** node represents a 2D shape formed by constructing 2D faces (polygons) from 2D vertices (points) specified in the **coord** field. The **coord** field contains a **Coordinate2D** node that defines the 2D vertices, referenced by the **coordIndex** field. The faces of an **IndexedFaceSet2D** node shall not overlap each other.

The detailed semantics are identical to those for the **IndexedFaceSet** node (see 7.2.2.66), restricted to the 2D case, and with the additional differences described here.

If the **texCoord** field is NULL, a default texture coordinate mapping is calculated using the local 2D coordinate system bounding box of the 2D shape, as follows. The X dimension of the bounding box defines the S coordinates, and the Y dimension defines the T coordinates. The value of the S coordinate ranges from 0 to 1, from the left end of the bounding box to the right end. The value of the T coordinate ranges from 0 to 1, from the lower end of the bounding box to the top end. Figure 21 illustrates the default texture mapping coordinates for a simple **IndexedFaceSet2D** shape consisting of a single polygonal face.

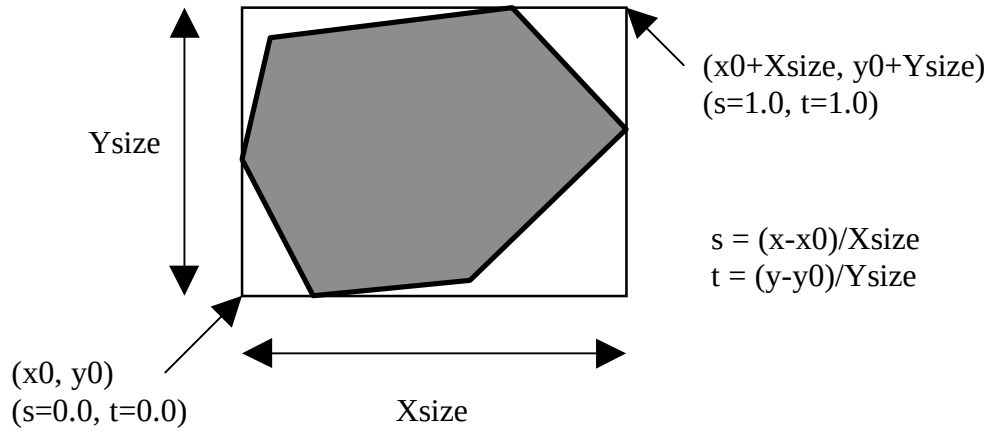


Figure 21 — IndexedFaceSet2D default texture mapping coordinates for a simple shape

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

When the Material2D indicates "filled" the faces (polygons) are drawn and each face (polygon) is filled on the insides according to the following simple inside rule:

To determine if a point is inside draw an imaginary line through the entire polygon and each time the line crosses the polygon's border increment a counter that was initialized to zero. When the count is odd the line is inside, when the count is even the line is outside.

When **color** field is non-null the color(s) are used either to fill the faces or to draw outlines of the faces depending on whether **Material2D filled** field is true or false respectively. In addition, if the **filled** field is true and the **Material2D lineProps** field is non-null then lines are drawn using the **LineProperties lineColor**.

When **color** field is null then the faces are filled and outlines are drawn using the rules listed in the **Material2D** node, see 7.2.2.80.2.

In all cases that outlines are drawn the lines are drawn using the **lineStyle** and **width** field values from the **Material2D lineProps**, whether explicitly specified, or default values when the field is null.

7.2.2.68 IndexedLineSet

7.2.2.68.1 Node interface

IndexedLineSet {

eventIn	MFInt32	**set_colorIndex**	
eventIn	MFInt32	**set_coordIndex**	
exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL
field	MFInt32	**colorIndex**	[]
field	SFBool	**colorPerVertex**	TRUE
field	MFInt32	**coordIndex**	[]

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.68.2 Functionality and semantics

The semantics of the **IndexedLineSet** node are specified in ISO/IEC 14772-1:1998, subclause 6.24.

7.2.2.69 IndexedLineSet2D

7.2.2.69.1 Node interface

IndexedLineSet2D {

eventIn	MFInt32	**set_colorIndex**	
eventIn	MFInt32	**set_coordIndex**	
exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL
field	MFInt32	**colorIndex**	[]
field	SFBool	**colorPerVertex**	TRUE
field	MFInt32	**coordIndex**	[]

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.69.2 Functionality and semantics

The **IndexedLineSet2D** node specifies a collection of lines or polygons.

The **coord** field shall list the vertices of the lines. When **coordIndex** is empty, the order of vertices shall be assumed to be sequential in the **coord** field. Otherwise, the **coordIndex** field determines the ordering of the vertices, with an index of -1 representing an end to the current polyline.

If the **color** field is not NULL, it shall contain a **Color** node, and the colors are applied to the line(s) as with the **IndexedLineSet** node (see 7.2.2.68).

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The lines shall be drawn using the **LineProperties** node (whether explicit or default) attributes of **lineStyle** and **width**. If the **IndexedLineSet2D color** field is null then the **Material2D** is used to set the color of all the lines and **emissiveColor** shall be used unless the **lineProps** field is non-null when the **LineProperties lineColor** shall be used instead.

7.2.2.70 Inline

7.2.2.70.1 Node interface

```
Inline {
   exposedField    MFString        url                     []
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.70.2 Functionality and semantics

The semantics of the **Inline** node are specified in ISO/IEC 14772-1:1998, subclause 6.25. ISO/IEC 14496-1 does not support the bounding box parameters (**bboxCenter** and **bboxSize**).

The **url** field specifies the data source to be used (see 7.1.1.2.7.1). The external source must contain a valid scene description stream.

7.2.2.71 InputSensor

7.2.2.71.1 Node interface

```
InputSensor {
      exposedField    SFBool          enabled                 TRUE
      exposedField    SFString        buffer                  ""
      exposedField    MFString        url                     ""
      eventOut        SFTime          eventTime
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.71.2 Functionality and semantics

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
]]
```

NOTE — 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 8.6.2). Allowed BIFS-Commands are the following: `FieldReplacement` (see 8.6.21), `IndexedValueReplacement` (see 8.6.22) and `NodeDeletion` with a NULL node argument (see 8.7.3.2). The **buffer** shall contain a number of BIFS-Commands that matches the number of fields in the DDF definition for the attached device. The type of the field replaced by the nth command in the **buffer** shall match the type of the nth field in the DDF definition.

The **url** field specifies the data source to be used (see 7.1.1.2.7.1). The **url** field shall point to a stream of type UserInteractionStream, which “access units” are DDFs.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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, 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 nth position indicates that the value of the DDF corresponding to the nth field of the DDF definition shall be ignored.

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

EXAMPLE — A typical use of this node is to handle the inputs of a keyboard.

7.2.2.71.3 Adding New Devices and Interoperability

In order to achieve interoperability when defining new devices, the way to use InputSensor with the new device needs to be specified. The following steps are necessary:

- define the content of the DDF definition: this sets the order and type of the data coming from the device and then mandates the content of the InputSensor buffer.
- define the deviceName string which will designate the new device.
- define the optional devSpecInfo of UIConfig.

NOTE — the bitstream syntax does not need to change.

7.2.2.71.4 Keyboard Mappings

The **KeySensor mapping** is defined as follows.

The KeySensor DDF definition is:

```
KeySensorDataFrame [[
  SFInt32 keyPressed
  SFInt32 keyReleased
  SFInt32 actionKeyPressed
  SFInt32 actionKeyReleased
  SFBool  shiftKeyChanged
  SFBool  controlKeyChanged
  SFBool  altKeyChanged
]]
```

keyPress and keyRelease events are generated as keys which produce characters are pressed and released on the keyboard. The value of these events is a string of length 1 containing the single UTF-8 character associated with the key pressed. The set of UTF-8 characters that can be generated will vary between different keyboards and different implementations.

actionKeyPress and actionKeyRelease events are generated as 'action' keys are pressed and released on the keyboard. The value of these events are:

KEY	VALUE	KEY	VALUE	KEY	VALUE
HOME	13	**END**	14	**PGUP**	15
PGDN	16	**UP**	17	**DOWN**	18
LEFT	19	**RIGHT**	20	**F1-F12**	1-12

shiftKeyChanged, controlKeyChanged, and altKeyChanged events are generated as the shift, alt and control keys on the keyboard are pressed and released. Their value is TRUE when the key is pressed and FALSE when the key is released.

The KeySensor UIConfig.devSpecInfo is empty.

The KeySensor deviceName is “KeySensor”

The **StringSensor mapping** is defined as follows.

The StringSensor DDF definition is:

```
StringSensorDataFrame [[
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    SFString enteredText
    SFString finalText
  ]]
```

The StringSensor UIConfig.devSpecInfo contains 2 UTF-8 strings: the first one is called terminationCharacter and the second one is called deletionCharacter. When no devSpecInfo is provided, the default terminationCharacter is '\r' and the default deletionCharacter is '\b'.

enteredText events are generated as keys which produce characters are pressed on the keyboard. The value of this event is the UTF-8 string entered including the latest character struck. The set of UTF-8 characters that can be generated will vary between different keyboards and different implementations. If deletionCharacter is provided, the previously entered character in the enteredText is removed. The deletionCharacter field contains a string comprised of one UTF-8 character. It may be a control character. If the deletionCharacter is the empty string, no deletion operation is provided.

The finalText event is generated whenever a sequence of keystrokes are recognized which match the keys in the terminationText string. When this recognition occurs, the enteredText is moved to the finalText and the enteredText is set to the empty string. This causes both a finalText event and an enteredText event to be generated.

The StringSensor deviceName is "StringSensor".

7.2.2.71.5 Mouse Mappings

The Mouse mapping is defined as follows.

The Mouse DDF definition is:

```
  MouseDataFrame [[
    SFVec2f position
    SFBool  leftButtonDown
    SFBool  middleButtonDown
    SFBool  rightButtonDown
    SFFloat wheel
  ]]
```

position is specified in screen coordinates, pixels or meter as specified in the BifsConfig. leftButtonDown becomes true when the left button is down, and false otherwise. Likewise for the middle and right buttons respectively. wheel values are: 0 when the wheel is inactive, +1 (resp. –1) when the wheel is moved forward (resp. backward) by one delta.

The Mouse UIConfig.devSpecInfo is empty.

The Mouse deviceName is "Mouse".

NOTE — This mouse mapping can be used with mice with 1 button, 2 buttons or 3 buttons, and possibly a wheel. DDF fields for missing buttons or wheel are simply never activated.

7.2.2.72 Layer2D

7.2.2.72.1 Node interface

Layer2D {			
eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	NULL
exposedField	SFVec2f	**size**	-1, -1
exposedField	SFNode	**background**	NULL
exposedField	SFNode	**viewport**	NULL
}			

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.72.2 Functionality and semantics

The **Layer2D** node is a transparent rendering rectangle region on the screen where a 2D scene is drawn. The rectangle always faces the viewer of the scene. **Layer2D** and **Layer3D** nodes enable the composition of multiple 2D and 3D scenes (see Figure 22).

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXAMPLE — This allows users to have 2D interfaces to a 2D scene, or 3D interfaces to a 2D scene, or to view a 3D scene from different viewpoints in the same scene.

The **addChildren** eventIn specifies a list of 2D nodes that shall be added to the **Layer2D's children** field.

The **removeChildren** eventIn specifies a list of 2D nodes that shall be removed from the **Layer2D's children** field.

The **children** field may contain any 2D children nodes that define a 2D scene. Layer nodes are considered to be 2D objects within the scene. The layering of the 2D and 3D layers is specified by any relevant transformations in the scene graph. The **Layer2D** node is composed with its center at the origin of the local coordinate system and shall not be present in 3D contexts (see 7.1.1.2.1).

The **size** parameter shall be a floating point number that expresses the width and height of the layer in the units of the local coordinate system. In case of a layer at the root of the hierarchy, the size is expressed in terms of the default 2D coordinate system (see 7.1.1.2.2). A size of -1 in either direction, means that the **Layer2D** node is not specified in size in that direction, and that the size is adjusted to the size of the parent layer, or the global rendering area dimension if the layer is on the top of the hierarchy. In the case where a 2D scene or object is shared between several **Layer2D** nodes, the behaviours are defined exactly as for objects that are multiply referenced using the DEF/USE mechanism. A sensor triggers an event whenever the sensor is triggered in any of the **Layer2D** in which it is contained. The behaviors triggered by the shared sensors as well as other behaviors that apply on objects shared between several layers apply on all layers containing these objects.

A **Layer2D** stores the stack of bindable children nodes that can affect the children scene of the layer. All relevant bindable children nodes have a corresponding exposedField in the **Layer2D** node. During presentation, these fields take the value of the currently bound bindable children node for the scene that is a child of the **Layer2D** node. Initially, the bound bindable children node is the corresponding field value of the **Layer2D** node if it is defined. If the field is undefined, the first bindable children node defined in the child scene will be bound. When the binding mechanism of the bindable children node is used (**set_bind** field set to TRUE), all the parent layers containing this node set the corresponding field to the current bound node value. It is therefore possible to share scenes across layers, and to have different bound nodes active, or to trigger a change of bindable children node for all layers containing a given bindable children node. For 2D scenes, the **background** field specifies the bound **Background2D** node. The **viewport** field is reserved for future extensions for 2D scenes.

All the 2D objects contained in a single **Layer2D** node form a single composed object. This composed object is considered by other elements of the scene to be a single object. In other words, if a **Layer2D** node, A, is the parent of two objects, B and C, layered one on top of the other, it will not be possible to insert a new object, D, between B and C unless D is added as a child of A.

Layers are transparent to user input if the background field is set to NULL. If the background field is specified, any transparent part of the background will also let user input through to lower layers.

EXAMPLE — In the following example, the same scene is used in two different **Layer2D** nodes. However, one scene is initially viewed with background b1, the other with background b2. When the user clicks on the button1 object, all layers are set with background b3.

```
OrderedGroup{
   children [
      Transform2D { # A set of transforms to translate and scale the layer
         ...
         children [
            Layer2D {
               background DEF b1 Background2D {…}
                  # It is possible to define the bindable children node directly in
                  # the corresponding field
               children [
                  DEF MYSCENE Transform2D {
                     children [
                        DEF b3 Background2D {…} # A shared background
                        DEF TS TouchSensor{}
                        DEF button1 Shape{..}   # The button 1
                                  # The objects of my scene
                     ]
                  }
               ]
            }
         ]
      }
      Transform2D {
                  # Another set of transforms to translate and scale the layer
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      children [
         Layer2D {
            children [
            DEF b2 Background2D{…} # It is possible to define the bindable
                                   # children node in the children field.
                                   # b2 is initially bound sicne it is the
                                   # first background 2D in the children
                                   # field OF the parent Layer2d
            Transform2D USE MYSCENE
            ]
         }
      ]
   }
]
}

ROUTE TS.isActive TO b3.set_bind
```

7.2.2.73 Layer3D

7.2.2.73.1 Node interface

Layer3D {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	NULL
exposedField	SFVec2f	**size**	-1, -1
exposedField	SFNode	**background**	NULL
exposedField	SFNode	**fog**	NULL
exposedField	SFNode	**navigationInfo**	NULL
exposedField	SFNode	**viewpoint**	NULL

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.73.2 Functionality and semantics

The **Layer3D** node is a transparent, rectangular rendering region where a 3D scene is drawn. The **Layer3D** node may be composed in the same manner as any other 2D node. It represents a rectangular region on the screen facing the viewer. The basic **Layer3D** semantics are identical to those for **Layer2D** (see 7.2.2.72) but with 3D (rather than 2D) children. In general, **Layer3D** nodes shall not be present in 3D co-ordinate systems. The permitted exception to this in when a **Layer3D** node is the "top" node that begins a 3D scene or context (see 7.1.1.2.1).

The following fields specify bindable children nodes for **Layer3D**:

background for **Background** and **Background2D** nodes

fog for **Fog** nodes

navigationInfo for **NavigationInfo** nodes

viewpoint for **Viewpoint** nodes

The **viewpoint** field can be used to allow the viewing of the same scene with several viewpoints.

NOTE — The rule for transparency to behaviors is also true for navigation in **Layer3D**. Authors should carefully design the various **Layer3D** nodes in a given scene to take account of navigation. Overlapping several **Layer3D** with navigation turned on may trigger strange navigation effects which are difficult to control by the user. Unless it is a feature of the content, navigation can be easily turned off using the **NavigationInfo** type field, or **Layer3D's** can be designed not to be superimposed.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

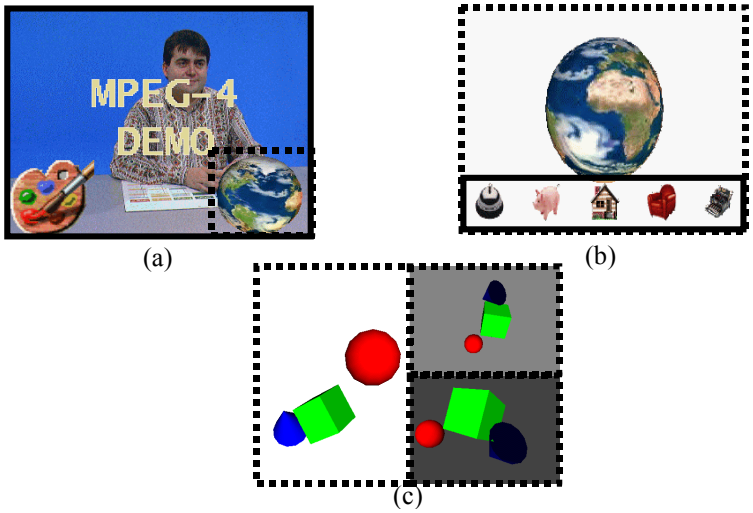


Figure 22 — Three Layer2D and Layer3D examples composed in a 2D space.

Layer2D's are indicated by a continuous line; **Layer3D's** by a dashed line. Image (a) shows a **Layer3D** containing a 3D view of the earth on top of a **Layer2D** composed of a video, a logo and a text. Image (b) shows a **Layer3D** of the earth with a **Layer2D** containing various icons on top. Image (c) shows 3 views of a 3D scene with 3 non-overlapping **Layer3D**.

7.2.2.74 Layout

7.2.2.74.1 Node interface

Layout {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFBool	**wrap**	FALSE
exposedField	SFVec2f	**size**	-1, -1
exposedField	SFBool	**horizontal**	TRUE
exposedField	MFString	**justify**	["BEGIN"]
exposedField	SFBool	**leftToRight**	TRUE
exposedField	SFBool	**topToBottom**	TRUE
exposedField	SFFloat	**spacing**	1.0
exposedField	SFBool	**smoothScroll**	FALSE
exposedField	SFBool	**loop**	FALSE
exposedField	SFBool	**scrollVertical**	TRUE
exposedField	SFFloat	**scrollRate**	0.0
exposedField	SFInt32	**scrollMode**	0

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.74.2 Functionality and semantics

The **Layout** node specifies the placement (layout) of its children in various alignment modes as specified. For text children, this is by their **fontStyle** fields, and for non-text children by the fields **horizontal**, **justify**, **leftToRight**, **topToBottom** and **spacing** present in this node. It also provides the functionality of scrolling its children horizontally or vertically.

The **children** field shall specify a list of nodes that are to be arranged. Note that the children's position is implicit and that order is important.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The **wrap** field specifies whether children are allowed to wrap to the next row (or column in vertical alignment cases) after the edge of the layout frame is reached. If **wrap** is set to TRUE, children that would be positioned across or past the frame boundary are wrapped (vertically or horizontally) to the next row or column. If **wrap** is set to FALSE, children are placed in a single row or column that is clipped if it is larger than the layout.

When **wrap** is TRUE, if text objects larger than the layout frame need to be placed, these texts shall be broken down into pieces that are smaller than the layout. The preferred places for breaking text are spaces, tabs, hyphens, carriage returns and line feeds. When there is no such character in the texts to be broken, the texts shall be broken at the last character that is entirely placed in the layout frame.

The **size** field specifies the width and height of the layout frame.

The **horizontal**, **justify**, **leftToRight**, **topToBottom** and **spacing** fields have the same meaning as in the **FontStyle** node (see 7.2.2.61).

The **scrollRate** field specifies the time needed in seconds to scroll the layout in the given direction. For example, a layout of 200x100 pixels scrolling vertically with a **scrollRate** value of 2 will translate its objects vertically of 100/2 times the simulation frame duration in seconds (eg, 1.65 pixels at 30 fps). When **scrollRate** is zero, then there is no scrolling and the remaining scroll-related fields are ignored.

The **smoothScroll** field selects between smooth and line-by-line/character-by-character scrolling of children. When TRUE, smooth scroll is applied.

The **loop** field specifies continuous looping of children when set to TRUE. When **loop** is FALSE, child nodes that have scrolled out of the scroll layout frame will be deleted. When **loop** is TRUE, then the set of children scrolls continuously, wrapping around when they have scrolled out of the layout area. If the set of children is smaller than the layout area, some empty space will be scrolled with the children. If the set of children is bigger than the layout area, then only some of the children will be displayed at any point in time. When **scrollVertical** is TRUE and **loop** is TRUE and **scrollRate** is negative (top-to-bottom scrolling), then the bottom-most object will reappear on top of the layout frame as soon as the top-most object has scrolled entirely into the layout frame.

The **scrollVertical** field specifies whether the scrolling is done vertically or horizontally. When set to TRUE, the scrolling rate shall be interpreted as a vertical scrolling rate and a positive rate shall be interpreted as scrolling towards the top. When set to FALSE, the scrolling rate shall be interpreted as a horizontal scrolling rate and a positive rate shall mean scrolling to the right.

Objects are placed one by one, in the order they are given in the children list. Text objects are placed according to the **horizontal**, **justify**, **leftToRight**, **topToBottom** and **spacing** fields of their **FontStyle** node. Other objects are placed according to the same fields of the **Layout** node. The reference point for the placement of an object is the reference point as left by the placement of the previous object in the list.

In the case of vertical alignment, objects may be placed with respect to their top, bottom, center or baseline. The baseline of non-text objects is the same as their bottom.

Spacing shall be coherent only within sequences of objects with the same orientation (same value of **horizontal** field). The notions of top edge, bottom edge, base line, vertical center, left edge, right edge, horizontal center, line height and row width shall have a single meaning over coherent sequences of objects. This means that over a sequence of objects where **horizontal** is TRUE, **topToBottom** is TRUE and **spacing** has the same value, then the vertical size of the lines is computed as follows:

maxAscent is the maximum of the ascent on all text objects.

maxDescent is the maximum of the descent on all text objects.

maxHeight is the maximum height of non-text objects.

If the minor mode in the **justify** field of the layout is FIRST (baseline alignment), then the non-text objects shall be aligned on the baseline, which means the vertical size of the line is:

$$size = max(maxAscent, maxHeight) + maxDescent$$

If the minor mode in the justify field of the layout is any other value, then the non-text objects shall be aligned with respect to the top, bottom or center, which means the size of the line is:

$$size = max(maxAscent+maxDescent, maxHeight)$$

The first line is placed with its top edge flush to the top edge of the layout; the base line is placed maxAscent units lower, and the bottom edge is placed maxDescent units lower. The center line is in the middle, between the top and bottom edges. The top edges of subsequent lines are placed at regular intervals of value spacing $\times$ size.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The other cases can be inferred from the above description. When the orientation is vertical, then the baseline, ascent and descent are not useful for the computation of the width of the rows. All objects only have a width. Column size is the maximum width over all objects.

EXAMPLE —

If **wrap** is FALSE:

a) If **horizontal** is TRUE, then objects are placed in a single line. The layout direction is given by the **leftToRight** field. Horizontal alignment in the row is done according to the first argument in **justify** (major mode = flush left, flush right, centered), and vertical alignment is done according to the second argument in **justify** (minor mode = flush top, flush bottom, flush baseline, centered). The **topToBottom** field is meaningless in this configuration.

b) If ***horizontal*** is FALSE, then objects are placed in a single column. The layout direction is given by the ***topToBottom*** field. Vertical alignment in the column is done according to the first argument in ***justify*** (major mode), and horizontal alignment is done according to the second argument in justify (minor mode).

If **wrap** is TRUE:

a) If ***horizontal*** is TRUE, then objects are placed in multiple lines. The layout direction is given by the ***leftToRight*** field. The wrapping direction is given by the ***topToBottom*** field. Horizontal alignment in the lines is done according to the first argument in ***justify*** (major mode), and vertical alignment is done according to the second argument in ***justify*** (minor mode).

b) If ***horizontal*** is FALSE, then objects are placed in multiple column. The layout direction is given by the ***topToBottom*** field. The wrapping direction is given by the ***leftToRight*** field. Vertical alignment in the columns is done according to the first argument in ***justify*** (major mode), and horizontal alignment is done according to the second argument in ***justify*** (minor mode).

If **scrollRate** is zero, then the **Layout** is static and positions change only when children are modified.

If **scrollRate** is non-zero, then the position of the children is updated according to the values of **scrollVertical**, **scrollRate**, **smoothScroll** and **loop**.

If **scrollVertical** is TRUE, then if ***scrollRate*** is positive, then the scrolling direction is left-to-right, and vice-versa.

If **scrollVertical** is FALSE, then if ***scrollRate*** is positive, then the scrolling direction is bottom-to-top, and vice-versa.

The **SCROLLMODE** field specifies the scrolling mode. The allowed values are –1, 0 and +1 and there meaning is respectively scroll-in, scroll-in-out, scroll-out.

Given that all the objects to be laid out are positioned, the bounding box (BB) of these objects is computed, and the **SCROLLMODE** field values are interpreted as follows:

scrollMode	**scrollVertical** = FALSE		**scrollVertical** = TRUE	
	scrollRate < 0	**scrollRate** > 0	**scrollRate < 0**	**scrollRate** > 0
-1 (scroll-in) Objects are initially translated so that :	left edge of BB is aligned with right edge of the layout frame (LF)	right edge of BB is aligned with left edge of LF	bottom edge of BB is aligned with top edge	top edge of BB is aligned with bottom edge
1 (scroll-out): Objects are scrolled until:	right edge of BB is aligned with left edge of LF	left edge of BB is aligned with right edge of LF	top edge of BB is aligned with bottom edge	bottom edge of BB is aligned with top edge

Value 0 of the **scrollMode** field corresponds to the combination of both scroll-in and scroll-out modes.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.75 LineProperties

7.2.2.75.1 Node interface

```
LineProperties {
    exposedField    SFColor     lineColor       0, 0, 0
    exposedField    SFInt32     lineStyle       0
    exposedField    SFFloat     width           1.0
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.75.2 Functionality and semantics

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:

Table 20 — lineStyle description

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 distiguishable 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. The join style is described in Figure 23.

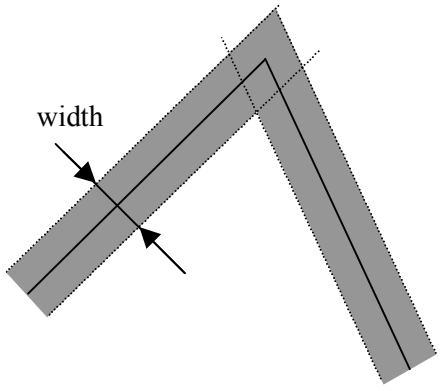


Figure 23 — Cap and join style for LineProperties

7.2.2.76 LinearGradient

7.2.2.76.1 Node interface

```
LinearGradient {
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

exposedField	SFNode	**transform**	NULL
exposedField	SFVec2f	**startPoint**	0 0
exposedField	SFVec2f	**endPoint**	1 0
exposedField	SFInt32	**spreadMethod**	0
exposedField	MFFloat	**key**	[]
exposedField	MFColor	**keyValue**	[]
exposedField	MFFloat	**opacity**	[1]

}

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.76.2 Functionality and semantics

The **LinearGradient** node is a texture ode 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 must 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 specifed meaning all color values have the same opacity, else an opacity must be specified for each color value.

EXAMPLE

```
Shape {
    geometry Rectangle { size 3 1 }
    appearance Appearance {
        material GradientLinear {
            key[ 0 1 ]
            keyValue [ 0 1 0, 1 1 0 ]
        }
}
```



7.2.2.77 ListeningPoint

7.2.2.77.1 Node interface

ListeningPoint {

eventIn	SFBool	**set_bind**

©ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

exposedField	SFBool	**jump**	TRUE
exposedField	SFRotation	**orientation**	0, 0, 1, 0
exposedField	SFVec3f	**position**	0, 0, 10
field	SFString	**description**	""
eventOut	SFTime	**bindTime**	
eventOut	SFBool	**isBound**	

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.77.2 Functionality and semantics

The **ListeningPoint** node specifies the reference position and orientation for spatial audio presentation. If there is no **ListeningPoint** given in a scene, the apparent listener position is slaved to the active **ViewPoint**.

The semantics are identical to those of the **Viewpoint** node (see 7.2.2.136).

7.2.2.78 LOD

7.2.2.78.1 Node interface

LOD {

exposedField	MFNode	**level**	[]
field	SFVec3f	**center**	0, 0, 0
field	MFFloat	**range**	[]

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.78.2 Functionality and semantics

The semantics of the **LOD** node are specified in ISO/IEC 14772-1:1998, subclause 6.26.

7.2.2.79 Material

7.2.2.79.1 Node interface

Material {

exposedField	SFFloat	**ambientIntensity**	0.2
exposedField	SFColor	**diffuseColor**	0.8, 0.8, 0.8
exposedField	SFColor	**emissiveColor**	0, 0, 0
exposedField	SFFloat	**shininess**	0.2
exposedField	SFColor	**specularColor**	0, 0, 0
exposedField	SFFloat	**transparency**	0.0

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.79.2 Functionality and semantics

The semantics of the **Material** node are specified in ISO/IEC 14772-1:1998, subclause 6.27.

7.2.2.80 Material2D

7.2.2.80.1 Node interface

Material2D {

exposedField	SFColor	**emissiveColor**	0.8, 0.8, 0.8
exposedField	SFBool	**filled**	FALSE
exposedField	SFNode	**lineProps**	NULL

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

exposedField	SFFloat	**transparency**	0.0

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.80.2 Functionality and semantics

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 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 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.

7.2.2.81 MaterialKey

7.2.2.81.1 Node interface

MaterialKey {

exposedField	SFBool	**isKeyed**	TRUE
exposedField	SFBool	**isRGB**	TRUE
exposedField	SFColor	**keyColor**	0, 0, 0
exposedField	SFFloat	**lowThreshold**	0
exposedField	SFFloat	**highThreshold**	0
exposedField	SFFloat	**transparency**	0

}

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.81.2 Functionality and semantics

The **MaterialKey** node can be used in the material field of the **Appearance** node, which only appears in the **appearance** field of a **Shape** node. It can be used when the texture of the **Shape** node is defined by either an image (**ImageTexture** or **PixelTexture**) or a video sequence (**MovieTexture**). Its functionality is similar to the **Material2D** node, but is specific to the BitMap geometry, so it does not include the line properties functionality. It generates a shape mask, based on a color and the threshold values defined in the node. It also defines a transparency value, which will behave identically to the transparency values in both **Material** and **Material2D**, except that it applies only to the visible part of the shape.

The fields of the **MaterialKey** node are defined as follows:

The **isKeyed** field specifies whether the keying functionality is enabled or disabled.

The **isRGB** field allows the content author to choose which color space they wish to define the keying in, either RGB or YUV.

The **keyColor** field specifies the reference color used for keying of shape.

The **lowThreshold** field defines the magnitude of the variance from the exact key value for which the pixel will be considered completely transparent.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The **highThreshold** field defines the magnitude of the variance from the exact key value for which the pixel will be considered opaque (visable).

The **transparency** field defines the level of transparency assigned to the opaque or visable region of the shape.

An example implementation of **MaterialKey** is given in subclause 7.5.

7.2.2.82 MatteTexture

7.2.2.82.1 Node interface

MatteTexture {

Field	SFNode	**surfaceA**	NULL
Field	SFNode	**surfaceB**	NULL
Field	SFNode	**alphaSurface**	NULL
exposedField	SFString	**operation**	""
Field	SFBool	**overwrite**	FALSE
exposedField	SFFloat	**fraction**	0
exposedField	MFFloat	**parameter**	0

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.82.2 Functionality and semantics

The **MatteTexture** node uses image compositing operations to combine the image data from two surfaces onto a third surface. The result of the compositing operation is computed at the resolution of **surfaceB**. If the size of **surfaceA** differs from that of **surfaceB**, the image data on **surfaceA** is zoomed up or down before performing the operation.

The compositing operations that are defined are capable of being hardware accelerated using low-cost, widely available graphics accelerators.

The **surfaceA, surfaceB** and **alphaSurface** fields specify the three surfaces that provide the input image data for the compositing operation. Not all three surfaces have to specified. In particular, there are unary, binary, and ternary operations. Each of these fields can contain any MPEG-4 texture node. These include **CompositeTexture2D**, **CompositeTexture3D**, **PixelTexture**, **MovieTexture**, **ImageTexture** and **MatteTexture**

The operation field specifies what compositing function to perform on the input surfaces.

The **parameter** and **fraction** fields provides one or more floating point parameters that can alter the effect of the compositing function. The specific interpretation of the parameter values depends upon which operation is specified.

The **overwrite** field indicates whether the **MatteTexture** node should allocate a new surface for storing the result of the compositing operation (**overwrite** = FALSE) or whether the data stored on surfaceB should be overwritten with the results of the compositing operation (**overwrite** = TRUE).

Note:Authors should only set overwrite to TRUE when they are certain that overwriting the contents of surfaceB will not have any adverse side-effects.

The possible values for **operation** are:

Unary Operations operate on the texture in the surfaceB field:

"INVERT" replaces the value C in each channel of with 1-C. The **parameter** field is used to specify whether or not channels containing alpha are inverted. If **parameter** is 0, then alpha channels are not inverted. If **parameter** is 1, then alpha channels are inverted.

"OFFSET" shifts the image DX pixels to the right and DY pixels to the top. Negative DX and DY values shift the image left and down, respectively. The DX and DY value are taken as the first two values in the parameter field. The color of pixels that are exposed by the OFFSET operation is set to black with an alpha value of 1.

"SCALE" scales each channel independently by multiplying the color in channel i by the value in **parameter**[i]. Pixel color and alpha values are clamped to the range 0 to 1."BIAS" modifies the color in channel i by adding to it the value in **parameter**[i]. Pixel color and alpha values are clamped to the range 0 to 1.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

“BLUR” performs a Gaussian blur operation on the image. The Gaussian blur kernel is an approximation of the normalized convolution:

$H(x) = \exp(-x^2/ (2s^2)) / \sqrt{2* \pi * s^2}$

Where ‘s’ is the standard deviation.

The value of stdDeviation is specified in the **parameter** field and can be either one or two numbers. If two numbers are provided, the first number represents a standard deviation value along the x-axis of the surface and the second value represents a standard deviation along the y-axis. If one number is provided, then that value is used for both x and y. Even if only one value is provided for stdDeviation, this can be implemented as a separable convolution.

NOTE — For larger values of 's' (s >= 2.0), an approximation may be used: Three successive box-blurs build a piece-wise quadratic convolution kernel, which approximates the Gaussian kernel to within roughly 3%.

let d = floor(s * 3*sqrt(2*pi)/4 + 0.5)

... if d is odd, use three box-blurs of size 'd', centered on the output pixel.

... if d is even, two box-blurs of size 'd' (the first one centered one pixel to the left, the second one centered one pixel to the right of the output pixel) and one box blur of size 'd+1' centered on the output pixel.

“COLOR_MATRIX” multiplies the RGBA value of each pixel by a matrix:

```
|R'|   |a00 a01 a02 a03|   |R|
|G'|   |a10 a11 a12 a13| * |G|
|B'| = |a20 a21 a22 a23|   |B|
|A'|   |a30 a31 a32 a33|   |A|
```

This matrix can be used for many purposes, including swapping channels and performing color space conversions. The matrix values are given in row order in the **parameter** field.

As an example, the following matrix swaps the red and blue channels:

```
| 0 0 1 0 |
| 0 1 0 0 |
| 1 0 0 0 |
| 0 0 0 1 |
```

The following matrix converts luminance to alpha:

```
|   0       0       0       0 |
|   0       0       0   0   |
|   0       0       0   0 |
| 0.299  0.587   0.114   0 |
```

Binary Operations operate on the textures in the surfaceB and either the **surfaceA or alphaSurface fields**:

"REPLACE_ALPHA" combines the RGB channels of surfaceB with the alpha channel from **alphaSurface**. If **alphaSurface** has 1 component (grayscale intensity only), that component is used as the alpha values. If **alphaSurface** has 2 or 4 components (grayscale intensity+alpha or RGBA), the alpha channel is used to provide the alpha values. If **alphaSurface** has 3 components (RGB), the operation is undefined. This operation can be used to provide static or dynamic alpha masks for static or dynamic imagery. For example, a texture node could render an animating James Bond character against a transparent background. The alpha from this image could then be used as a mask shape for a video clip.

"MULTIPLY_ALPHA" behaves just like REPLACE_ALPHA, except the alpha values from **alphaSurface** are multiplied with the alpha values from **surfaceB**.

"CROSS_FADE" fades between two surfaces using the value in the fraction field to control the percentage of each surface that is visible. This operation can dynamically fade between two static or dynamic images. By animating the fraction field value from 0 to 1, the imagery on **surfaceA** fades into that of **surfaceB**.

"BLEND" combines the image data from **surfaceA** and **surfaceB** using the alpha channel from **surfaceB** to control the blending percentage. This operation allows the alpha channel of **surfaceB** to control the blending of the two images. By animating the alpha channel of **surfaceB** by rendering a texture node or playing a **MovieTexture**, you can produce a complex traveling matte effect. If R1, G1, B1, and A1 represent the red, green, blue, and alpha values of a pixel of **surfaceA** and R2, G2, B2, and A2 represent the red, green, blue, and alpha values of the corresponding pixel of **surfaceB**, then the resulting values of the red, green, blue, and alpha components of that pixel are:

red = R1 * (1 - A2) + R2 * A2

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

green = G1 * (1 - A2) + G2 * A2

blue = B1 * (1 - A2) + B2 * A2

alpha = 1

"ADD", and "SUBTRACT" add or subtract the color channels of **surfaceA** and **surfaceB**. The alpha of the result equals the alpha of **surfaceB**.

“A” is the identity operator for **surfaceA**. In other words, the resulting image contains the contents of **surfaceA**. If overwrite is TRUE, then the contents of **surfaceB** are overwritten with the contents of **surfaceA**.

“B” is the identity operator for **surfaceB**. In other words, the resulting image contains the contents of **surfaceB**.

Ternary Operations operate on the textures in the surfaceA, **surfaceB**, **and alphaSurface fields**:

"REVEAL" is similar to CROSS_FADE except that the **fraction** value does not directly specify the percentage of **surface1** and **surface2** to use in the result. Instead, the **fraction** value specifies a threshold level for a third surface (i.e. the **alphaSurface**). In regions of the **alphaSurface** where the alpha values are less than the threshold, the resulting pixels come from **surface1**. In regions of the **alphaSurface** where the alpha values are greater than the threshold, the resulting pixels come from **surface2**.

So far, this describes a hard-edged transition region between **surface1** and **surface2**. In other words, each pixel of the result comes directly from either **surface1** or **surface2**. Introducing a softness value (which is specified using the first value of the **parameter** field), allows a range of alpha values surrounding the threshold value to be specified where the result is a linear blend of **surface1** and **surface2**.

For example, if softness (soft) = 0.1, and threshold (thresh) = 0.5, then for alpha values less than or equal to (thresh - soft) = 0.4, the result would be **surface1**. For alpha values greater than or equal to (thresh + soft) = 0.6, then result would be **surface2**. For alpha values between 0.4 and 0.6, then result would be a linear combination of **surface1** and **surface2**:

1 - (1/(2*soft)) * (alpha + soft - thresh)) * surface1 + (1/(2*soft)) * (alpha + soft - thresh) * surface2

Example :

The following example shows how the node can be used to mix three surfaces.

The following scene uses a “REVEAL” operation to mix two images using an alphaSurface. The figures below show the alphaSurface used (Figure 24) and a snap shot of the operation for a value of TransitionEffect.fraction between 0 and 1 (Figure 25).

```
# content fragment showing an image processing transition effect using
# MatteSurface and its REVEAL operator
#
# MovieA transitions to reveal MovieB on the same in-scene texture,
# as TransitionEffect.fraction is animated from 0.0 to 1.0

DEF VideoScreen Transform {
  children Shape {
    appearance  Appearance {
        texture DEF TransitionEffect MatteTexture  {
           surfaceA DEF MovieA MovieTexture{
              url "A.roll.mpg"
           }
           surfaceB DEF MovieB MovieTexture {
              url "B.roll.mpg"
           }
           alphaSurface ImageTexture {
              url "revealDiamondArt.png"
           }
           parameter 0.063
           operation "REVEAL"
        }
      }
    }
    geometry IndexedFaceSet {
      coord  Coordinate {
        point[ -10 -7.5 0,
           -10 7.5 0,
           10 7.5 0,
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            10 -7.5 0 ]
          }
          coordIndex [ 0, 1, 2, 3, -1 ]
          solid FALSE
      }
    }
}
```

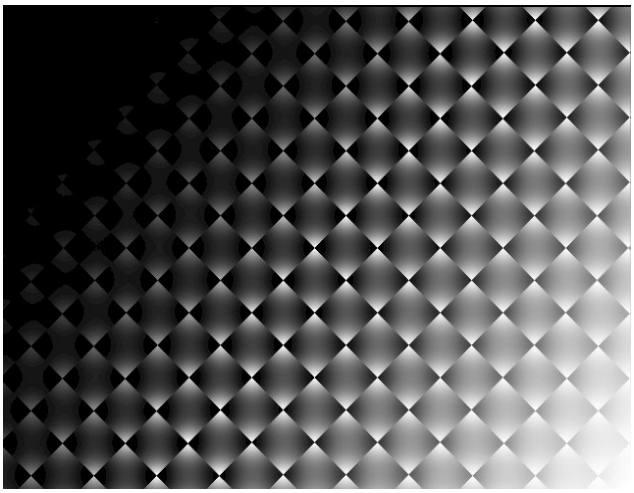


Figure 24 — An alphaSurface (revealDiamondArt.png) used in a "REVEAL" operation to mix two videos.



Figure 25 — An image resulting from a "REVEAL" operation on A.roll.mpg and B.roll.mpg using the alphaSurface in Figure 24.

Repeat pattern (**repeatS** and **repeatT**) of a **MatteTexture** node is given by the repeat pattern of the **surfaceB** texture of that node.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.83 MediaBuffer

7.2.2.83.1 Node interface

MediaBuffer {

exposedField	SFFloat	**bufferSize**	0.0
exposedField	MFString	**url**	[ ]
exposedField	SFTime	**mediaStartTime**	-1
exposedField	SFTime	**mediaStopTime**	+I
EventOut	SFBool	**isBuffered**	
exposedField	SFBool	**enabled**	TRUE

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.83.2 Functionality and semantics

The **MediaBuffer** node allows storage of media streams in local buffers created specifically for playback. This allows, for instance, storage of clips for interactive playback or looping.

Storage of a stream object in the **MediaBuffer** shall occur only if the stream is active (see **MediaControl** section).

The time interval of the stream object with media time between **mediaStartTime** and **mediaStopTime** shall be stored. If these values are changed as a stream object is being buffered, the result is undefined.

The **mediaStartTime** or **mediaStopTime** fields have special values; see the **MediaControl** section.

The **url** field refers to the stream objects that are to be stored; there shall be one buffer for each stream object.

The **bufferSize** field signals how many seconds of media shall be stored locally. If **bufferSize** = –1.0 the whole range of **mediaStartTime** to **mediaStopTime** shall be stored. If this range is unbounded because the duration of the stream object is not known, no buffering shall occur.

Note – The physical buffer sizes can be computed from stream parameters and either the **bufferSize** value or **mediaStartTime** and **mediaStopTime**.

The **isBuffered** event sends a TRUE value when all of the streams in the **url** have been completely buffered.

When the **enabled** field is set to TRUE, the buffers shall be allocated. When the **enabled** field is set to FALSE the buffers may be freed and no buffering shall take place. If a media buffer has insufficient space to add more media samples, the earliest added media samples are discarded and replaced with the most recently received media samples.

When buffering of a stream object is started, all previous buffer contents shall be discarded.

Playback of a stream object shall occur through the MediaBuffer under the following conditions:

A media node referring to the same stream object referenced in the url field of the MediaBuffer becomes active, and

the requested playback time interval of that stream object is completely available in the MediaBuffer.

Note – Play back of buffered stream object may be controlled by a MediaControl node.

7.2.2.84 MediaControl

7.2.2.84.1 Node interface

MediaControl {

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

exposedField	MFString	**url**	“”
exposedField	SFTime	**mediaStartTime**	-1
exposedField	SFTime	**mediaStopTime**	+I
exposedField	SFFloat	**mediaSpeed**	1.0
exposedField	SFBool	**loop**	FALSE
exposedField	SFBool	**preRoll**	TRUE
exposedField	SFBool	**mute**	FALSE
exposedField	SFBool	**enabled**	TRUE
EventOut	SFBool	**isPreRolled**	

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.84.2 Functionality and semantics

The **MediaControl** node controls the play back and, hence, delivery of a media stream referenced by a media node. The **MediaControl** node allows selection of a time interval within one or more stream objects for play back, modification of the playback direction and speed, as well as pre-rolling and muting of the stream.

A media node may be used with or without an associated **MediaControl** node. A media node for which no **MediaControl** node is present shall behave as if a **MediaControl** node for that media stream were present in the scene, with default values set.

The **url** field contains a reference to one or more stream objects (“OD:n#segment” or “OD:n”), called the controlled stream objects, all of which must belong to the same media stream. This media stream is called the controlled stream. When any media node referring to a media stream in its **url** field is active, the associated media stream is said to be active.

Note – This means that the controlled stream becomes active exactly when some media node pointing to it becomes active. The controlled stream becomes inactive, when all media nodes referring to it become inactive.

When a controlled media stream becomes active, the associated controlled stream objects in the **url** field of the MediaControl node shall be played sequentially.

The **mediaStartTime** and **mediaStopTime** fields define the time interval, in media time, of each controlled stream object to be played back.

If media time of the media stream is undefined, selection of a time interval of the controlled stream object for play back is not supported. In that case the **mediaStartTime** and **mediaStopTime** fields shall be ignored.

The following values have special meaning for **mediaStartTime** and **mediaStopTime**:

0 indicates the beginning of the controlled stream object

-1 indicates the media time of the controlled stream object when the associated media node becomes active.

+I, or any value greater than the duration of the controlled stream object indicates its end.

Semantics of **mediaStartTime** and **mediaStopTime** depend on the delivery scenario.

Semantics in case of delivery scenarios that permit seeking:

Play back of the controlled stream object shall start at **mediaStartTime** of the first controlled media object when the controlled stream becomes active.

When the controlled stream becomes inactive and then active again, then if **mediaStartTime** is –1 the stream starts playing from the point where it was last stopped. Otherwise the first controlled stream object in the **url** field restarts playing from **mediaStartTime**.

If the **loop** field is TRUE, all the controlled stream objects are played in a loop, each in the range **mediaStartTime** to **mediaStopTime** while the controlled stream is active. If **mediaStartTime** is –1, each stream object will start from the beginning.

In all delivery scenarios, play back of the controlled stream object shall occur only in the range defined by **mediaStartTime** and **mediaStopTime**. Outside this range the play back shall be muted. The **loop** field shall be ignored in delivery scenarios that do not permit seeking.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The **mediaSpeed** is a requested multiplication factor to the normal speed of each controlled stream object. Negative values for **mediaSpeed** request that the controlled stream object plays backward from **mediaStartTime** to **mediaStopTime**. When this field is zero, the controlled stream shall be paused.

NOTE — All streams, independent of speed, are only played in the range defined by **mediaStartTime** and **mediaStopTime**. When **mediaSpeed** < 0, the stream object can only be played if the server reassigns time stamps to be increasing from **mediaStopTime** to **mediaStartTime**.

If **mediaSpeed** > 0 (forward play back) and **mediaStopTime** < **mediaStartTime,** then the controlled stream object will play until the end.

If **mediaSpeed** < 0 (backward play back) and **mediaStopTime** > **mediaStartTime,** then the controlled stream object will play to the beginning.

In these equations, the special value –1 is substituted by the actual value of media time that it represents.

There is no requirement that a delivery service supports specific ranges of **mediaSpeed** other than **mediaSpeed** = 1. Media content shall comply with maximum and average bit rates specified for the stream, irrespective of the value of the **mediaSpeed** field.

If the **preRoll** field is set to TRUE the controlled stream should be pre-rolled in order to be ready to start instantly when the controlled stream becomes active. All streams that are associated to the same object time base as the stream that is pre-rolled should also be pre-rolled.

If the delivery scenario does not permit seeking, **preRoll** = TRUE means that the controlled stream object should be delivered and recently received access units should be stored in the decoding buffer in order to enable instantaneous play back when the media node becomes active.

Note – Play back of stream objects in media nodes that are not controlled by **MediaControl** or where **preRoll** is FALSE may suffer an unspecified startup delay if play back is requested by an unpredictable action (e.g. user interaction, script).

The **isPreRolled** event sends a TRUE value when the controlled stream object has completed pre-rolling.

If the **mute** field is set to TRUE, the stream objects in the **url** field are not rendered when they are played. However, their media clock is not stopped. For visual streams, whether natural video or synthetic such as animation streams or **Inline** nodes, **mute** means that the visual texture remains unchanged; for audio streams, the audio is not played.

If the **enabled** field is set to TRUE the **MediaControl** node controls the stream object it refers to. More than one **MediaControl** node may be used to control a stream object within the same stream. At most one of these **MediaControl** nodes shall be enabled at any time.

If one of these **MediaControl** nodes becomes enabled, the **enabled** field of all other **MediaControl** nodes that refer to the same stream shall automatically be set to FALSE.

If the **enabled** field is set to FALSE the **MediaControl** node shall cease to control the play back and muting of the controlled stream object, however, **preRoll** shall still be evaluated. If the controlled stream object is playing when **enabled** is set to FALSE and no other **MediaControl** node takes control of the stream, the stream object shall continue playing as if it were still controlled by the disabled **MediaControl** node.

Only one **MediaControl** node shall refer to any of the set of media streams that are associated to a single object time base.

Note – **MediaControl** affects the OTB of the controlled stream and therefore affects all the streams that are associated to the same OTB. Therefore changing play position, speed or direction of one stream will correspondingly affect all the active streams that are associated to the same OTB.

EXAMPLE — The following scene shows how to control a video from a MediaControl the Script also receives the current video time and can trigger events according to the video time: In the example, when the TouchSensor is touched, the video will play the section ranging between 10 to 20 seconds from the start of the sequence.

```
 [....]
Shape {
        texture DEF M MovieTexture { url "od:5"}
        geometry BitMap{}
   }
   DEF MS MediaSensor {
      url "od:5"
   }
   MediaControl {
      url "od:5"
      mediaStartTime 10.0
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      mediaStopTime  20.0
   }
   DEF T TouchSensor {}
   DEF S Script {
        eventIn SFTime videoTime
        ...
   }

ROUTE MS.mediaCurrentTime TO S.videoTime
ROUTE T.touchTime TO M.startTime
```

The following table lists values for various node fields in order to play, pause, and stop streams:

Action	Node Values
Play stream from beginning to end.	mediaStartTime = 0 mediaStopTime = +I
Play stream backwards from end to beginning	mediaStartTime = +I mediaStopTime = 0 mediaSpeed = -1.0
Pause stream	stopTime = NOW in the controlled media node and mediaStartTime = -1 Or mediaSpeed = 0
Play stream from current position to end	mediaStartTime = -1 mediaStopTime = +I
Play stream backwards from current position to beginning	mediaStartTime = -1 mediaStopTime = 0 mediaSpeed = -1.0
Play stream from 10 seconds past start to 20 seconds past start at half speed	mediaStartTime = 10 mediaStopTime = 20 mediaSpeed = 0.5
Play stream backwards from 20 seconds past start to 10 seconds past start at double speed	mediaStartTime = 20 mediaStopTime = 10 mediaSpeed = -2.0
Play stream forwards from current position to 10 seconds past start	mediaStartTime = -1 mediaStopTime = 10

7.2.2.85 MediaSensor

7.2.2.85.1 Node interface

MediaSensor **{**

exposedField	MFString	**url**	[ ]
eventOut	SFTime	**mediaCurrentTime**	
eventOut	SFTime	**streamObjectStartTime**	
eventOut	SFTime	**mediaDuration**	
eventOut	SFBool	**isActive**	
eventOut	MFString	**info**	

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.85.2 Functionality and semantics

The **MediaSensor** node monitors the availability and presentation status of one or more stream objects.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The **url** field identifies a list of stream objects monitored by the **MediaSensor** node. All the stream objects in the **url** field shall belong to the same media stream.

A stream object is considered to be available when any of its composition units is available in the composition buffer and is due for composition at that time. A stream object is considered to be no longer available when it is paused or stopped. A stream object is considered to "become available" when it "is available" for the first time. When there are several monitored stream objects available at the same time, the fields in the **MediaSensor** convey information about the stream object that became available last. If the stream that last became available becomes inactive, the **MediaSensor** node shall convey information about the first active stream in its **url** field.

The **isActive** event sends a TRUE value each time one of the monitored stream objects referred by the **url** field becomes available, and a FALSE value when all of them become not available.

Whenever a new composition unit is due for composition, a **mediaCurrentTime** event is sent and indicates the media time of that composition unit within the stream object.

The **streamObjectStartTime** event conveys the start of the stream object within a stream, relative to media time zero of the whole stream.

The **mediaDuration** event conveys the duration of the stream object in seconds. It is set to –1 if this duration is unknown.

The **info** event conveys information about the stream object that is currently monitored. Its first element identifies the stream object using the same syntax as in the **url** field.

The **streamObjectStartTime, mediaDuration** and **info** events are triggered when any stream object in the **url** field becomes available.

7.2.2.86 MovieTexture

7.2.2.86.1 Node interface

MovieTexture {

exposedField	SFBool	**loop**	FALSE
exposedField	SFFloat	**speed**	1.0
exposedField	SFTime	**startTime**	0
exposedField	SFTime	**stopTime**	0
exposedField	MFString	**url**	[]
field	SFBool	**repeatS**	TRUE
field	SFBool	**repeatT**	TRUE
eventOut	SFTime	**duration_changed**	
eventOut	SFBool	**isActive**	

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.86.2 Functionality and semantics

The **loop**, **startTime**, and **stopTime** exposedFields and the **isActive** eventOut, and their effects on the **MovieTexture** node, are described in 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):

fmod (now - **startTime**, duration/**speed**)

If **speed** is negative, then the frame or VOP to display is the frame or VOP at movie time:

duration + fmod(now - **startTime**, duration/**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

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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.

The **url** field specifies the data source to be used (see 7.1.1.2.7.1).

7.2.2.87 NavigationInfo

7.2.2.87.1 Node interface

```
NavigationInfo {
    eventIn         SFBool      set_bind
    exposedField    MFFloat     avatarSize      [0.25, 1.6, 0.75]
    exposedField    SFBool      headlight       TRUE
    exposedField    SFFloat     speed           1.0
    exposedField    MFString    type            ["WALK", "ANY"]
    exposedField    SFFloat     visibilityLimit 0.0
    eventOut        SFBool      isBound
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.87.2 Functionality and semantics

The semantics of **NavigationInfo** are specified in ISO/IEC 14772-1:1998, subclause 6.29.

7.2.2.88 Normal

7.2.2.88.1 Node interface

```
Normal {
    exposedField    MFVec3f     vector          []
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.88.2 Functionality and semantics

The semantics of the **Normal** node are specified in ISO/IEC 14772-1:1998, subclause 6.30.

7.2.2.89 NormalInterpolator

7.2.2.89.1 Node interface

```
NormalInterpolator {
    eventIn         SFFloat     set_fraction
    exposedField    MFFloat     key             []
    exposedField    MFVec3f     keyValue        []
    eventOut        MFVec3f     value_changed
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.89.2 Functionality and semantics

The semantics of the **NormalInterpolator** node are specified in ISO/IEC 14772-1:1998, subclause 6.31.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.90 OrderedGroup

7.2.2.90.1 Node interface

```
OrderedGroup {
    eventIn         MFNode      addChildren
    eventIn         MFNode      removeChildren
    exposedField    MFNode      children            []
    exposedField    MFFloat     order               []
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.90.2 Functionality and semantics

The **OrderedGroup** node controls the visual layering order of its children. When used as a child of a **Layer2D** node, it allows the control of which shapes obscure others. When used as a child of a **Layer3D** node, it allows content creators to specify the rendering order of elements of the scene that have identical z values. This allows conflicts between coplanar or close polygons to be resolved.

The **addChildren** eventIn specifies a list of objects that shall be added to the **OrderedGroup** node.

The **removeChildren** eventIn specifies a list of objects that shall be removed from the **OrderedGroup** node.

The **children** field is the current list of objects contained in the **OrderedGroup** node.

When the **order** field is empty (the default) children are layered in order, first child to last child, with the last child being rendered last. If the **order** field contains values, one value is assigned to each child. Entries in the **order** field array match the child in the corresponding element of the **children** field array. The child with the lowest order value is rendered before all others. The remaining children are rendered in increasing order. The child corresponding to the highest **order** value is rendered last. If there are more children than entries in the **order** field, those children that do not have a drawing order are drawn in the order in which they appear in the **children** field, but after the ones that have an entry in the **order** field.

If there are more order entries than children, the excess order entries are ignored.

Since 2D shapes have no z value, this is the sole determinant of the visual ordering of the shapes. However, when the **OrderedGroup** node is used with 3D shapes, its ordering mechanism shall be used in place of the natural z order of the shapes themselves. The resultant image shall show the shape with the highest **order** value on top, regardless of its z value. However, the resultant z-buffer contains a z value corresponding to the shape closest to the viewer at that pixel. The **order** shall be used to specify which geometry should be drawn first, to avoid conflicts between coplanar or close polygons.

NOTE — Content authors must use this functionality carefully since, depending on the **Viewpoint**, 3D shapes behind a given object in the natural z order may appear in front of this object.

7.2.2.91 OrientationInterpolator

7.2.2.91.1 Node interface

```
OrientationInterpolator {
    eventIn         SFFloat     set_fraction
    exposedField    MFFloat     key                 []
    exposedField    MFRotation  keyValue            []
    eventOut        SFRotation  value_changed
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.91.2 Functionality and semantics

The semantics of the **OrientationInterpolator** node are specified in ISO/IEC 14772-1:1998, subclause 6.32.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.92 PathLayout

7.2.2.92.1 Node Interface

PathLayout {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFNode	**geometry**	NULL
exposedField	MFInt32	**alignment**	[0, 0]
exposedField	SFFloat	**pathOffset**	0
exposedField	SFFloat	**spacing**	1.0
exposedField	SFBool	**reverseLayout**	FALSE
exposedField	SFInt32	**wrapMode**	0
exposedField	SFBool	**splitText**	TRUE

}

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.92.2 Functionality and Semantics

The **PathLayout** node is a grouping node used to place its 2D children along a given 2D path and possibly move them along that path. See ISO/IEC 14772-1:1998 for a description of the **children**, **addChildren**, and **removeChildren** fields and eventIns.

The **geometry** field contains a 2D geometry node defining the path. The following nodes are allowed in that field: IndexedFaceSet2D, IndexedLineSet2D, Curve2D and XCurve2D. The path is oriented from the first point to the last point. The length of the path is the sum of all the length of its sub-paths (set of connected curves or lines).

pathOffset describes the offset along the path to place the first object. Value 0 corresponds to the beginning of the path and value 1 to the end of the path. Negative values or values greater than 1 are handled according to the **wrapMode** field.

The **reverseLayout** field specifies whether the children are placed following the orientation of the path (FALSE) or the opposite orientation (TRUE).

The **alignment** field describes horizontal and vertical alignment in that order.

An object is placed as follows :

The tangent to the path at the current position is computed, chosen with the same orientation as the path;

The object is rotated so that the X-axis of its local coordinate system is parallel to the tangent and oriented in the same direction as the tangent;

Alignment is applied (see below).

The current position along the path is incremented by the current increment.

The initial position on the path is the value of the **pathOffset** field multiplied by the length of the path. The current increment depends on the current object, the next object to be placed and the alignment constraints.

Text nodes are considered as graphical objects if they are not direct children of the **PathLayout** node and therefore obey to the alignment constraints as specified by the **alignment** field. Otherwise, the **fontStyle** field of the **Text** node is used.

For graphical objects, alignment is applied as follows:

alignment[0]	Meaning	Increment	
		ReverseLayout TRUE	**ReverseLayout** FALSE
-1	left edge of the object is aligned with the current position	*spacing* × w_i	–*spacing* × w_{i+1}
0	middle of the object is aligned with the current position	*spacing* × $(w_i+w_{i+1})/2$	–*spacing* × $(w_i+w_{i+1})/2$
1	right edge of the object is aligned with the current position	*spacing* × w_{i+1}	–*spacing* × w_i

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

where w_i is the width of the current object to place and w_{i+1} is the width of the next object to place. If there is no further object, the increment is meaningless.

alignment[1]	Meaning
-1	top edge is aligned with the tangent
0	center is placed on the tangent
1	bottom edge is aligned with the tangent

For **Text** nodes that are direct children of the **PathLayout** node, their placement depends on the value of the **splitText** field.

If **splitText** is FALSE, the text is placed according to the **fontStyle** field of the **Text** node, with the origin of the local coordinate system being the current position on the path, and the X-axis of that system rotated so that it is parallel to the tangent and oriented in the same direction.

If **splitText** is TRUE, each character of the text is placed separately as if it was a single **Text** node with the same **fontStyle** field.

The **wrapMode** field indicates action to take when the current position is less than 0 or greater that the length of the path. The following values are defined:

wrapMode	Meaning
0	The current object is not rendered, but the current position is updated as specified above.
1	The current position is increased or decreased by a integer number of times the path length so that it is positive and less than the length of the path.
2	The path is virtually extended by a tangent line at the first and last point and the current position refers to that virtual path.

7.2.2.93 PerceptualParameters

7.2.2.93.1 Node interface

```
PerceptualParameters {
    exposedField    SFFloat     sourcePresence          1.0
    exposedField    SFFloat     sourceWarmth            1.0
    exposedField    SFFloat     sourceBrilliance        1.0
    exposedField    SFFloat     roomPresence            0.0
    exposedField    SFFloat     runningReverberance     1.0
    exposedField    SFFloat     envelopment             0.0
    exposedField    SFTime      lateReverberance        1.0
    exposedField    SFFloat     heavyness               1.0
    exposedField    SFFloat     liveness                1.0
    exposedField    MFFloat     omniDirectivity         1.0
    exposedField    MFFloat     directFilterGains       1.0, 1.0, 1.0
    exposedField    MFFloat     inputFilterGains        1.0, 1.0, 1.0
    exposedField    SFFloat     refDistance             1.0
    exposedField    SFFloat     freqLow                 250.0
    exposedField    SFFloat     freqHigh                4000.0
    exposedField    SFTime      timeLimit1              0.02
    exposedField    SFTime      timeLimit2              0.04
    exposedField    SFTime      timeLimit3              0.1
    exposedField    SFFloat     modalDensity            0.8
}
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.93.2 Functionality and Semantics

PerceptualParameters is a node that contains information about the perceptual properties of **DirectiveSound** objects when the perceptual rendering is desired. It contains a set of nine perceptual parameters that characterizes, for a given reference distance **refDistance** and for non-directive sounds, the acoustic to be rendered in the virtual scene. In addition it allows for physical-like effects such as transmission through a wall from another room (with **inputFilterGains)** or occlusion/diffraction of the direct path by an obstacle (with **directFilterGains)**. The directivity properties of the sound source is defined in the **directivity** fields in the **DirectiveSound** node level and in **omniDirectivity** field of this node for an arbitrary amount of azimuth angles from the front (defined in the **direction** field at the **DirectiveSound** node level) to the back of the sound source.

Generic reverberation response model:

The perceptual model is based on a temporal division of the reverberation response into four sections (see Figure 26):

direct sound (R_0)

directional early reflections (R_1)

diffuse early reflections (R_2)

diffuse late reverberation (R_3)

These four sections are separated by temporal limits (denoted l_0, l_1, l_2, l_3), and characterized by their energies in 3 frequency bands (low, mid, high). These frequency bands are separated by two cross-over frequencies denoted f_{low} and f_{high}.

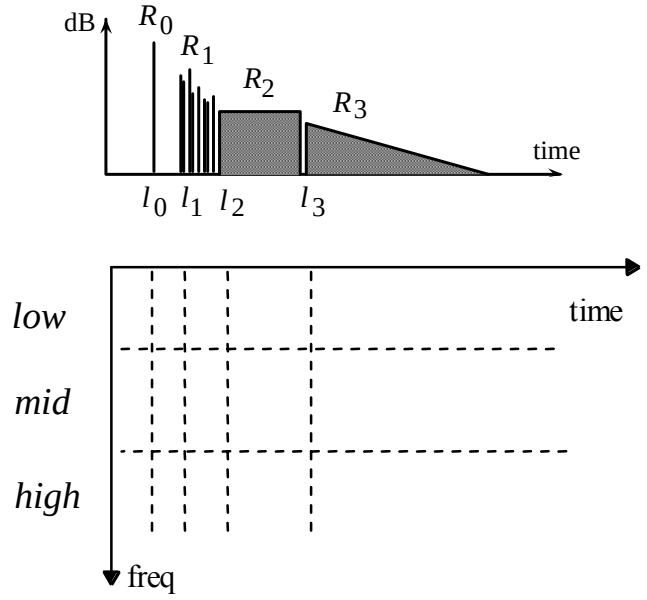


Figure 26 — Generic reverberation response model. R0 represents the direct sound, R1 the directional early relfections, R2 the diffuse reflections, ad R3 the exponentially decaying, diffuse late reverberation.

Based on the above model, the reverberation response is completely characterized by the following set of parameters:

energies R_0, R_1, R_2, R_3 (low, mid, high)

decay time Rt (low, mid, high)

temporal parameters l_0, l_1, l_2, l_3 + modal density

frequencies f_{low}, f_{high}

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The modal density is defined as the number of modes per Hz. This parameter is useful for the design and control of artificial reverberation algorithms based on recursive (IIR) digital filter structures.

High-level (perceptual) parameters:

In the perceptual acoustics rendering nine orthogonal perceptual parameters that directly relate to the audible sensations, are used to define the acoustic response for each sound source. A measurable acoustical criterion is defined for each perceptual parameter. These acoustical criteria represent an attempt to provide an exhaustive characterization of room acoustical quality in concert halls, opera houses and auditoria, by use of a minimal set of independent parameters. They can be expressed from energetic measures (low-level parameters) derived from a decomposition of the impulse response in three frequency bands and four temporal sections (see Figure 26), assuming time limits l_1, l_2, l_3 respectively equal to 20, 40, 100 ms relative to the time of arrival of the direct sound l_0, and with a dependence on the directional distribution of early reflections The formulas that define the mapping from perceptual parameters (and their corresponding acoustical criteria) to the low-level parameters are given in subclause 7.2.2.93.2.1.

The first nine fields of the **PerceptualParameters** node can be divided in three groups:

Three perceptual parameters describe effects which are characteristic of the room (the corresponding objective criteria are indicated in parentheses):

lateReverberance (late decay time, denoted Rt)

heaviness and **liveness** (relative decay time at low and high frequencies, denoted Drtl and Drth)

The six other perceptual parameters describe effects that can depend of the position of the source in a given room. The first three are perceived as characteristics of the source, while the remaining three are perceptually associated with the room:

sourcePresence (Es, affecting the “early energy”, i.e. the energy of the direct sound and early room effect)

brilliance and **warmth** (Desl, Desh, relative early energy at low and high frequencies)

roomPresence (Rev, affects mostly the energy of the late room effect)

runningReverberance (relative early decay time, denoted Edt)

envelopment (Rdl, energy of early room effect relative to direct sound)

A variation of **sourcePresence** creates a convincing effect of proximity or remoteness of the sound source. The term "reverberance" refers to the sensation that sounds are prolonged by the room reverberation. **lateReverberance** differs from **runningReverberance** by the fact that it is essentially perceived during interruptions of the sound radiated by the source, for example when the source falls silent. **runningReverberance**, on the contrary, remains perceived during continuous music.

omniDirectivity is the diffuse-field spectrum for the source. This will be called the omnidirectional directivity because it defines the directivity of an “equivalent omnidirectional source” (equivalent with regards to the reverberation, but not the direct path). It could be derived from the **directivity** field as defined in the **DirectiveSound** node by averaging the radiated power over all directions around the source. However, it is simpler and more reliable to transmit it separately in the bitstream. The same approach as for **directivity** is considered except that it doesn’t depend on the angles.

The general form for the **omniDirectivity** field is:

[nf, $freq_0$,$freq_1$,....$freq_{nf-1}$, $gain_0$, $gain_1$,....$gain_{nf-1}$].

Where,

nf is the number of reference frequencies

$freq_j$ is the jth reference frequency

$gain_j$ is the linear gain for the jth reference frequency.

An example of **omniDirectivity** is given below:

[5, 250, 500, 1000, 2000, 4000,

0.9, 0.85, 0.7, 0.6, 0.55]

If not specified in the node, the default gains at Hz is **gain** $_0$.

By default, the gain for frequencies above $f_{nf-1}$is **gain** $_{nf-1}$.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

directFilterGains specifies a filter applied to the direct path only (amplitude gains in the three frequency bands defined by the crossover frequencies **freqLow**, **freqHigh**).

inputFilterGains specifies a filter applied to the source signal similarly as **directFilterGains** for the direct path.

refDistance is a reference distance at which the above set of perceptual parameters is defined (in meters). If the distance in the scene is different from this value, it is used for calculating a new value for the **sourcePresence**. **RefDistance** shall be strictly positive.

The generic room response that is modeled in the perceptual approach is characterized in the frequency domain by two frequency limits, **freqLow** and **freqHigh** (see Figure 26). This generic room response is also characterized in the temporal domain by four time limits and by the modal density of the late reverb. The **PerceptualParameters** node contains **timeLimit1, timeLimit2, timeLimit3** which are the temporal limits l_1, l_2, l_3 (relative to l_0) and **modalDensity** (in seconds).

7.2.2.93.2.1 Mapping from high-level to low-level parameters

In order to use a reverberator to process the sound sources, it is necessary to convert from perceptual parameters to energetic parameters.

When an acoustical or perceptual criterion is updated at the higher level, the necessary modifications in the low-level energetic description of the room response can be readily computed via a nonlinear matrix inversion procedure (this is explained below). When the signal processing model is scaled down in order to reduce the computational cost, this is reflected in the behaviour of the perceptual control interface. For instance, if the *reverb* block is shared between several sources, the late decay time settings are constrained to be identical for these sources. If the *cluster* block is suppressed, the running reverberance and the room envelopment are no longer independently controllable.

When computing the energetic parameters of the room response, the perceptual parameters are denoted as follows :

Table 21 — Perceptual parameters

Perceptual parameter field	*Notation*	*Min*	*Max*
sourcePresence	Es	0.0	1.0
SourceWarmth	Desl	0.1	10.0
sourceBrilliance	Desh	0.1	10.0
roomPresence	Rev	0.0	1.0
runningReverberance	Edt_{rel}	0.0	1.0
envelopment	Rdl_{rel}	0.0	1.0
lateReverberance	Rt (s)	0.1	1000.0
heaviness	Drtl	0.1	10.0
liveness	Drth	0.1	1.0

Es and Rev are absolute energies, and Rdl_{rel} is a relative energy value. Desl, Desh, Drtl, Drth are multiplicative factors. Rt (reverberation time) is expressed in seconds, and Edt_{rel} is a relative early decay time value.

With the above notations, the energetic factors are calculated as follows:

$C = pow(10, -1.2 / Rt)$

if $Rev/Es =< 2*(1+C)/(1-C)$

R3$=(-C + sqrt[C^2+0.5*Rev/Es*(1-C)^2]) *4*Es/(1-C)^2$

else

R3$= Rev + 2*Es$

if $(2*Es/R3 =< 30.622)$

$Edt_{min}= 0.4 + Rt * [1 - 0.667*log10(1 + 2*Es/R3)]$

else

$Edt_{min}= 0.6 / log10(1 + 2*Es/R3)$

if $(Es/R3 =< 30.622)$

$Edt_{max}= 0.4 + Rt * [1 - 0.667*log10(1 + Es/R3)]$

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

else

Edt_{max}= 0.6 / log10(1 + Es/R3))

The early decay time in seconds is calculated as:

$Edt = Edt_{min} + (Edt_{max} - Edt_{min}) * Edt_{rel}$

If Edt > 0.4

R2 = -Es + R3 [pow(10, 1.5 * (1 + (0.4-Edt)/Rt)) -1]

else

R2 = -Es + R3 [pow(10, 0.6 / Edt) -1]

Rdl_{min} = 0.05*R2 /Es

Rdl_{max} = 0.27 + 0.05*R2 /Es

The absolute envelopment is computed as:

$Rdl = Rdl_{min} + (Rdl_{max} - Rdl_{min}) * Rdl_{rel}$

R1= (Es*Rdl - 0.05*R2) / 0.3

$\mathbf{R1_{low}}$= R1*Desl

$\mathbf{R1_{high}}$= R1*Desh

R0= Es - R1

$\mathbf{R0_{low}}$= R0*Desl

$\mathbf{R0_{high}}$= R0*Desh

$\mathbf{Rt_{low}}$ = Drtl * Rt

$\mathbf{Rt_{high}}$ = Drth * Rt

NOTE - All the values are energies expressed in the linear domain.

7.2.2.93.2.2 Mapping from positional to perceptual parameters

If the source is not placed at the reference distance **refDistance** for which the perceptual "preset" is defined, the following correction is applied (when **distance** field of the parent **DirectiveSound** node is different from 0):

10*Log10(Es) = 10*log10(Es) – 60*log2(*d* / **refDistance**) / log2(**distance**) ,

where *d* is the actual distance between the source and the listening point. In order to avoid saturation when *d* is small, Es shall be clipped to the value it takes when *d* = 1m.

For an example implementation of perceptual approach, see 7.6.

7.2.2.94 PixelTexture

7.2.2.94.1 Node interface

PixelTexture {

exposedField	SFImage	**image**	0 0 0
field	SFBool	**repeatS**	TRUE
field	SFBool	**repeatT**	TRUE

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.94.2 Functionality and semantics

The semantics of the **PixelTexture** node are specified in ISO/IEC 14772-1:1998, subclause 6.33.

7.2.2.95 PlaneSensor

7.2.2.95.1 Node interface

```
PlaneSensor {
    exposedField    SFBool      autoOffset              TRUE
    exposedField    SFBool      enabled                 TRUE
    exposedField    SFVec2f     maxPosition             -1 -1
    exposedField    SFVec2f     minPosition             0 0
    exposedField    SFVecf3f    offset                  0 0 0
    eventOut        SFBool      isActive
    eventOut        SFVec3f     trackPoint_changed
    eventOut        SFVec3f     translation_changed
}
```

7.2.2.95.2 Fnctionality and semantics

The semantics of the **PlaneSensor** node are specified in ISO/IEC 14772-1:1998, subclause 6.34.

7.2.2.96 PlaneSensor2D

7.2.2.96.1 Node interface

```
PlaneSensor2D {
    exposedField    SFBool      autoOffset              TRUE
    exposedField    SFBool      enabled                 TRUE
    exposedField    SFVec2f     maxPosition             0, 0
    exposedField    SFVec2f     minPosition             0, 0
    exposedField    SFVec2f     offset                  0, 0
    eventOut        SFBool      isActive
    eventOut        SFVec2f     trackPoint_changed
    eventOut        SFVec2f     translation_changed
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.96.2 Functionality and semantics

This sensor detects pointer device dragging and enables the dragging of objects on the 2D rendering plane.

The semantics of **PlaneSensor2D** are a restricted case for 2D of the semantics for the **PlaneSensor** node (see 7.2.2.95).

7.2.2.97 PointLight

7.2.2.97.1 Node interface

```
PointLight {
    exposedField    SFFloat     ambientIntensity        0.0
    exposedField    SFVec3f     attenuation             1, 0, 0
    exposedField    SFColor     color                   1, 1, 1
    exposedField    SFFloat     intensity               1.0
    exposedField    SFVec3f     location                0, 0, 0
    exposedField    SFBool      on                      TRUE
    exposedField    SFFloat     radius                  100.0
}
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.97.2 Functionality and semantics

The semantics of the **PointLight** node are specified in ISO/IEC 14772-1:1998, subclause 6.35.

7.2.2.98 PointSet

7.2.2.98.1 Node interface

PointSet {

exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.98.2 Functionality and semantics

The semantics of the **PointSet** node are specified in ISO/IEC 14772-1:1998, subclause 6.36.

7.2.2.99 PointSet2D

7.2.2.99.1 Node interface

PointSet2D {

exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.99.2 Functionality and semantics

This is a 2D equivalent of the **PointSet** node (see 7.2.2.98), with semantics that are the 2D restriction of that node.

7.2.2.100 PositionAnimator

7.2.2.100.1 Node interface

PositionAnimator {

eventIn	SFFloat	**set_fraction**	
exposedField	SFVec2f	**fromTo**	0 1
exposedField	MFFloat	**key**	[]
exposedField	SFInt32	**keyType**	0
exposedField	MFVec2f	**keySpline**	[0 0, 1 1]
exposedField	MFVec3f	**keyValue**	[]
exposedField	MFRotation	**keyOrientation**	[]
exposedField	MFFloat	**weight**	[]
exposedField	SFInt32	**keyValueType**	0
exposedField	SFVec3f	**offset**	0 0 0
eventOut	SFVec3f	**value_changed**	
eventOut	SFRotation	**rotation_changed**	
eventOut	SFVec3f	**endValue**	

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.100.2 Functionality and semantics

The model of the node follows a content creator way of designing animation. Animator nodes specify the path as well as timelines. The Animator nodes also have flags for cumulative path when it is repeated as well as normal vector to the path for specifying the orientation at the beginning of the animation.

The timeline is specified by **key**, **keyType**, and **keySpline** fields. The path is specified with **keyValue**, **keyValueType**, and **keyValueSpline** fields. The orientation at each key position is specified in **keyOrientation**.

Timeline specification

fromTo specifies when the Animator is active. By default, an animator is active for the whole duration of the timeline: from t=0 to t=1. **fromTo** can be used to produce animations with interruptions (Figure 27).

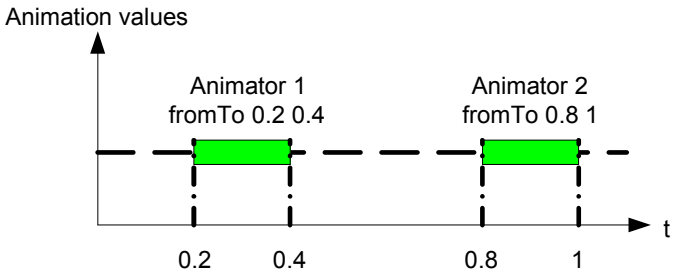


Figure 27 — fromTo field usage. The same timer can be used to trigger two Animator nodes starting and ending at different times.

keyType defines the timeline and five types of predefined timelines are possible (Figure 28). By default, if **key** is empty, linear timeline model is assumed (**keyType** 2).

For **keyType** 0 and **key** empty cases, the behavior is similar to VRML/BIFS interpolators.

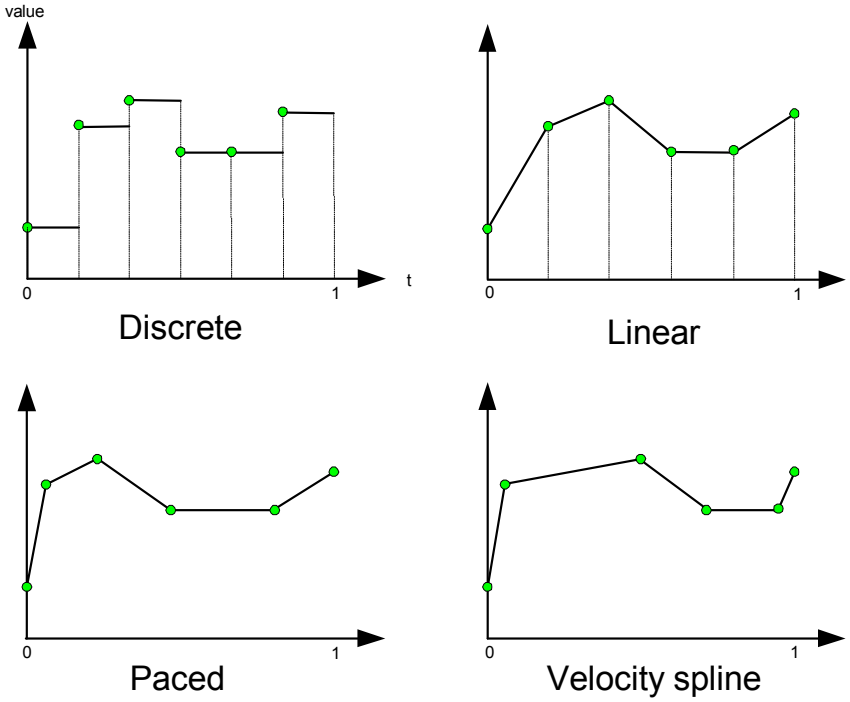


Figure 28 – Predefined types of timelines: discrete, linear, paced (constant speed), and velocity spline. The first type (keyType 0) is not shown and is user-specified. Note how the animation paths are identical but the timelines so different.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Let's denote

- $t \in [0,1]$ the time fraction received in **set_fraction** field,
- *i* is the curve segment defined by two **keyValues** *i* and *i+1*,
- $t_i = key[i], \quad i = 0,\ldots,n-1.$
- $\mathbf{v}_i = keyValue[i], \quad i = 0,\ldots,n-1.$
- $\mathbf{q}_i = keyOrientation[i], \quad i = 0,\ldots,n-1.$
- t_i and t_{i+1} the time corresponding to v_i and v_i ,
- *n* is the number of **keyValue**s.

0	*User defined*, like interpolators. The time is specified for each keyValue. If there are *n* keyValues, there must be exactly *n* keys.
1	*Discrete*. The timeline is divided into *n* equal intervals. The value remains constant on interval *i* to $\mathbf{v}_i$: $i = \lfloor t * n \rfloor$ $f(t) = t_i = i/n$ $\mathbf{v}(t) = \mathbf{v}_i \quad t_i \le t < t_{i+1}$
2	*Linear*. The timeline is divided into *(n-1)* equal intervals. $i = \lfloor t * (n-1) \rfloor$ $t_i = \frac{\lfloor t * (n-1) \rfloor}{n-1}$ $f(t) = t$ $\mathbf{v}(t) = \mathbf{C}_i(t)$ $\mathbf{q}(t) = \mathbf{Q}_i(t)$ For a line segment $\mathbf{C}_i$, the position is $\mathbf{v}(t) = \mathbf{v}_i + \frac{t - t_i}{t_{i+1} - t_i}(\mathbf{v}_{i+1} - \mathbf{v}_i)$.
3	*Paced*. The speed remains constant over the animation path. A reparametrization by arclength (the distance traveled) is needed. The arclength is defined as $s(t) = \int_0^t \lvert \mathbf{C}'(u) \rvert du$ where $\mathbf{C}'(u)$ is the first order derivative of the animation path $\mathbf{C}(u)$ by the animation parameter *u*, In general, $\lvert \mathbf{C}'(u) \rvert$ is not integrable (especially for splines) and it is left to the implementation to decide which quadrature formula to use. Whatever the approximation used, the speed should remain constant, the exact position of keyframes is not mandatory.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

	For piecewise linear paths, the arclength is: $$d_i = \sum_{j=0}^{i-1} \left| \mathbf{P}_{j+1} - \mathbf{P}_j \right| \quad \text{for piecewise linear paths},$$ Let d_i be the distance traveled up to $\mathbf{v}_i$, d_n the total length of the animation path, and $t_i = \frac{d_i}{d_n}$ the time spent to travel d_i, then $$\mathbf{v}(t) = \mathbf{v}_i + \frac{t - t_i}{t_{i+1} - t_i}(\mathbf{v}_{i+1} - \mathbf{v}_i)$$ For orientation, $\frac{t - t_i}{t_{i+1} - t_i}$ is used to interpolate between $\mathbf{q}_i$ and $\mathbf{q}_{i+1}$.
4	*Spline*. Defines a cubic Bézier velocity curve. Such a curve is defined by 4 points $[\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3]$: end points are $\mathbf{P}_0 = (0,0)$ and $\mathbf{P}_3 = (1,1)$ and the two intermediate points $\mathbf{P}_1, \mathbf{P}_2$ are defined in **keySpline**, see Figure 29. The equation of the velocity curve is: $$\mathbf{f}(u) = \mathbf{U}\mathbf{M}_\mathbf{B}\mathbf{P}_c = [u^3 \quad u^2 \quad u \quad 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \\ 1 \end{bmatrix}$$

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

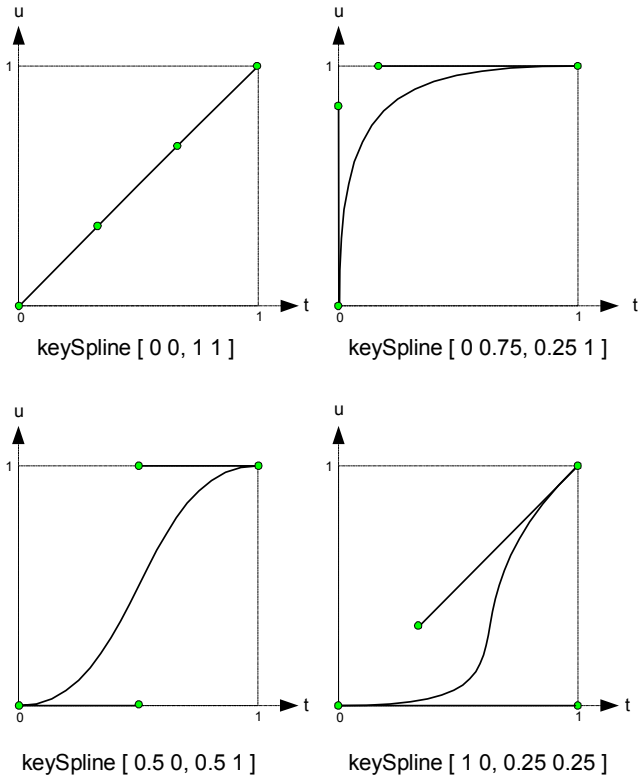


Figure 29 – Timeline control using cubic Bézier spline.

Algorithmically, $C_x(u) = t$ is first solved for the parameter *u* using a simple bisection algorithm, which is appropriate since the function is strictly monotonically increasing. $t' = C_y(u)$ is determined for *u*. *t'* is then used to determine the path segment it belongs to determine the position value $\mathbf{v}(t')$ and orientation $\mathbf{q}(t')$

$$
\begin{aligned}
i &= \lfloor t' * (n-1) \rfloor \\
\mathbf{v}(t') &= \mathbf{C}_i(t') \\
\mathbf{q}(t') &= \mathbf{Q}_i(t')
\end{aligned}
$$

key must contain the keyframes for each **keyValue** and **keyOrientation**.

Path specification

keyValueType specifies the type of curve characteristics. Apart from 1 (line), a curve is represented as a NURBS curve (type 3) and types 1 (quadratic Bézier) and 2 (cubic Bézier) are special cases. For NURBS and interpolating curves, the number of points *N* used must follow the type i.e. like [… *3 N* …]. Type 0 allows cusps in the animation path; this enables jumping from one path to another without in the same Animator node.

Table 22 — This table gives for each keyValueType, the number of keyValue that must be in the keyValue field.

0	Line	1
1	Quadratic Bézier	3
2	Cubic Bézier	4
3	NURBS curve	<N>

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4	Interpolating cubic spline curve	<N>

Notes:

For type 0, multiple line segments can be defined: the polyline passes though each point in **keyValue** field (like any interpolator). For other types, only one type of curve is possible (for example, if a path needs to have two cubic Bézier curves, they can be represented by a single NURBS curve and continuity constraints resolved in the knot vector).

For other type, an Animator node specifies only one type of curve characteristic. In general, any continuous path can be represented by a NURBS. Also, using cumulating (**offset** and **endValue** fields, see below), two Animator can be chained.

keyValue specifies the points $\{\mathbf{P}_i\}$ the animation path should pass through (line and interpolating types) or the control points for curves (quadratic and cubic Bézier, and NURBS).

weight specifies the weights $\{w_i\}$ for each keyValue. If **weight** is empty then $w_i = 1 \quad \forall i$ is assumed. Else, there must be as many weights as control points.

Note that type 1 and 2 are special cases of NURBS curves. They are separate for convenience and better compression due to their important use in animation:

For quadratic Bézier curves (**keyValueType** 2), $p = 2$ and $U = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}$. 3 control points (from **keyValue**) are needed.

For cubic Bézier curves (**keyValueType** 3), $p = 3$ and $U = \{0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1\}$. 4 control points (from **keyValue**) are needed.

Non-rational B-spline curves are obtained for NURBS curve (**keyValueType** 4) with $w_i = 1 \quad \forall i$ and a knot vector.

For NURBS curves (**keyValueType** 3), **key** contains the knot vector $\{u_i\}$. While this specification allows unclamped knot-vectors (first and last knot not repeated *p* times), this would produce curves not passing through end points and hence produce cusps in the animation path.

An interpolating cubic spline curves (**keyType** 4) passes through all <N> points specified just after the type in **keyValueType** field. Chord length parametrization with averaging knots shall be used:

$$d = \sum_{i=1}^{n-1} |\mathbf{C}_k - \mathbf{C}_{k-1}|$$

$$u_0 = \cdots = u_p = 0 \quad u_{m-p-1} = \cdots = u_{m-1} = 1$$

$$u_{j+p} = \frac{1}{p} \sum_{i=j}^{j+p-1} \overline{u}_i \quad j = 1, \ldots, n-p$$

with

$$\overline{u}_0 = 0 \quad \overline{u}_{n-1} = 1$$

$$\overline{u}_k = u_{k-1} + \frac{|\mathbf{C}_k - \mathbf{C}_{k-1}|}{d} \quad k = 1, \ldots, n-2$$

With this method the knots reflect the distribution of the $\overline{u}_k$. Furthermore, this results in a system of equation totally positive and banded with a bandwidth less than p, which can be solved by Gaussian elimination without pivoting. Note that the resulting animation would be similar to a paced timeline if linear timeline is used with this type of path. A cubic spline was chosen but other degrees are possible. However, choosing a cubic spline provides C^2 continuity, which is desirable in animation.

keyOrientation specifies the orientation $\{\mathbf{q}_i\}$ at each key value as a set of SFRotation (axis, angle). The algorithm in subclause A.2 is used except for **keyType** 0 (discrete) where no interpolation is done.

Cumulating specification

offset specifies an offset in position to be added to **value_changed** (Figure 30). By default no offset is added (**offset** is (0,0,0)). To cumulate two animations, animation 1 routes its **endValue** to animation 2's **offset**. If animation 2 starts at (0,0,0), the two animations will follow one after the other seamlessly.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

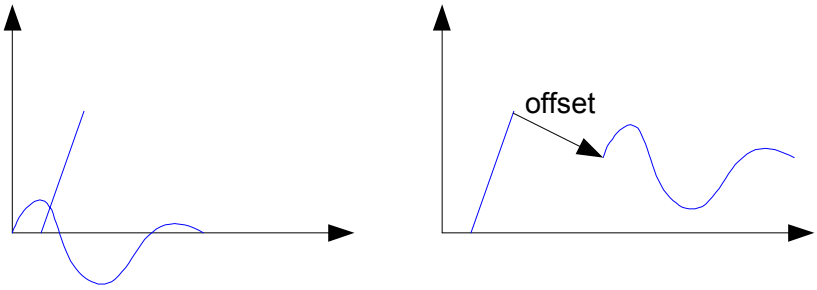


Figure 30 – Left: two animations without offset. Right: two animations chained by an offset.

Outputs

value_changed, **orientation_changed**, and **endValue** events are output.:

value_changed is $\mathbf{v}(t)$ of Eq. 2.

orientation_changed is the frame orientation at $\mathbf{q}(t)$ of Eq. 2.

endValue is emitted when the end of the animation is reached (i.e. **fromTo**[1]).

7.2.2.101 PositionAnimator2D

7.2.2.101.1 Node interface

PositionAnimator2D {

eventIn	SFFloat	**set_fraction**	
exposedField	SFVec2f	**fromTo**	0 1
exposedField	MFFloat	**key**	[]
exposedField	SFInt32	**keyType**	0
exposedField	MFVec2f	**keySpline**	[0 0, 1 1]
exposedField	MFVec2f	**keyValue**	[]
exposedField	SFInt32	**keyOrientation**	0
exposedField	MFFloat	**weight**	[]
exposedField	SFInt32	**keyValueType**	0
exposedField	SFVec2f	**offset**	0 0 0
eventOut	SFVec2f	**value_changed**	
eventOut	SFFloat	**rotation_changed**	
eventOut	SFVec2f	**endValue**	

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.101.2 Functionality and semantics

The semantic is the same as for **PositionAnimator** except that input and output values are 2D positions and **keyOrientation** has the following semantic.

If **keyOrientation** is 0, no angle is generated.

If **keyOrientation** is 1, the object is oriented normal to the curve in the direction of the curve parameter. The angle is output in **rotation_changed**.

If **keyOrientation** is 2, the object is oriented normal to the curve in the opposite direction of the curve parameter. The angle is output in **rotation_changed**.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.102 PositionInterpolator

7.2.2.102.1 Node interface

```
PositionInterpolator {
    eventIn         SFFloat     set_fraction
    exposedField    MFFloat     key                 []
    exposedField    MFVec3f     keyValue            []
    eventOut        SFVec3f     value_changed
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.102.2 Functionality and semantics

The semantics of the **PositionInterpolator** node are specified in ISO/IEC 14772-1:1998, subclause 6.37.

7.2.2.103 PositionInterpolator2D

7.2.2.103.1 Node interface

```
PositionInterpolator2D {
    eventIn         SFFloat     set_fraction
    exposedField    MFFloat     key                 []
    exposedField    MFVec2f     keyValue            []
    eventOut        SFVec2f     value_changed
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.103.2 Functionality and semantics

This is a 2D equivalent of the **PositionInterpolator** node (see 7.2.2.100) with semantics that are the 2D restriction of that node.

7.2.2.104 PositionInterpolator4D

7.2.2.104.1 Node interface

```
PositionInterpolator4D {
    eventIn         SFFloat     set_fraction
    exposedField    MFFloat     key                 []
    exposedField    MFVec4f     keyValue            []
    eventOut        SFVec4f     value_changed
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.104.2 Functionality and semantics

As **PositionInterpolator**, this node linearly interpolates 4-dimensional values.

7.2.2.105 ProceduralTexture

7.2.2.105.1 Node interface

```
ProceduralTexture {
    exposedField    SFInt32     type                0
    exposedField    SFInt32     width               7
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

exposedField	SFInt32	**height**	7
exposedField	SFInt32	**cellWidth**	4
exposedField	SFInt32	**cellHeight**	4
exposedField	SFInt32	**roughness**	0
exposedField	SFFloat	**distortion**	0
exposedField	SFInt32	**seed**	129093
exposedField	MFColor	**color**	0.3 0.698 1, 0.8 0.8 0.8, 1 1 1, 0 0 0]
exposedField	MFVec2f	**xWarpmap**	[]
exposedField	MFVec2f	**yWarpmap**	[]
exposedField	SFBool	**xSmooth**	FALSE
exposedField	SFBool	**ySmooth**	FALSE
exposedField	MFVec2f	**aWarpmap**	[0 0, 1 1]
exposedField	MFVec2f	**bWarpmap**	[0 0, 1 1]
exposedField	SFBool	**aSmooth**	FALSE
exposedField	SFBool	**bSmooth**	FALSE
exposedField	MFFloat	**aWeights**	[0,0,0,0,0,0,0,0,0,0,0,1,0,0,0]
exposedField	MFFloat	**bWeights**	[0,0,0,0,0,1,0,0,0,0,0,0,0,0,0]
eventOut	SFImage	**image_changed**	

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.105.2 Functionality and semantics

The processes underlying the creation of procedural textures include the generation of a fractal 'plasma' field, subdivision of the texture into cells, spatial distortion of the texture, selection of colors to apply to the texture, and control of how the colors vary within a cell. The five cell types supported are shown in Figure 31.

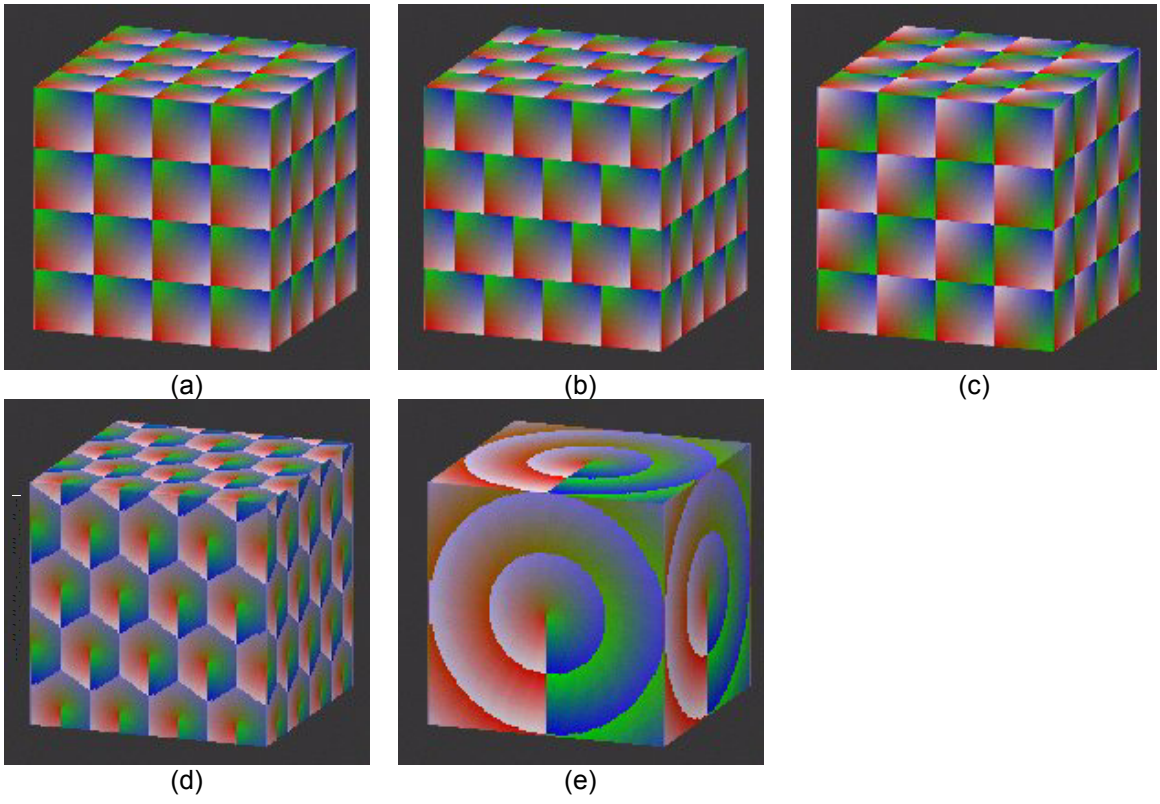


Figure 31 — The 5 cell types: (a) Rectangle, (b) Brick, (c) Weave, (d) Hexagon, (e) Ring.

The **type** field specifies the basic cell type (Figure 31), while **width** and **height** specify the overall texture dimensions and **cellWidth**, **cellHeight** the basic cell size. Note that the actual sizes in pixels are $2^{\mathbf{width}}$, $2^{\mathbf{height}}$, $2^{\mathbf{cellWidth}}$ and $2^{\mathbf{cellHeight}}$ respectively.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The plasma properties are controlled through the **roughness**, **distortion** and **seed** parameters (subclause B.2).

The **color** field describes the four colors used to generate the final texture.

The **xWarpmap**, **yWarpmap**, **xSmooth** and **ySmooth** parameters warp the x and y coordinates prior to the lookup into the plasma, while the **aWarpmap**, **bWarpmap**, **aSmooth** and **bSmooth** fields provide an extra warping stage prior to the final interpolation between the four **color** parameters. See subclause B.3 for details.

The **aWeights** and **bWeights** arrays are used to provide a weighted blend between x, y, x_{warp}, y_{warp}, x_{dist}, y_{dist}, x_{offset}, y_{offset}, r_{offset}, e_{offset}, a_{offset}, p_{center}, p_{dist}, p_{offset}, $rand_{cell}$, $rand_{dist}$ (see subclause B.2 for details). Note that for the majority of textures most of these weights will be zero.

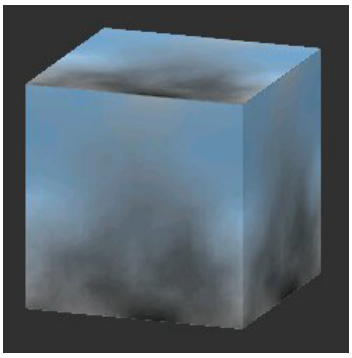


Figure 32 — Default plasma texture.

The output **image_changed** can be employed directly as a texture map. Figure 32 shows the basic texture generated by the default values.

7.2.2.105.3 Texture Generation

Underlying all procedural textures is a pseudo-random fractal ‘plasma’ (subclause B.2) which is constructed so that it can be tiled horizontally and vertically without discontinuities between one tile and its neighbor(s).

The generation of the texture is effectively a mapping from a pair of coordinates (x, y) to a color value. The stages in this procedure are as follows:

The coordinates are ‘warped’ (subclause B.3) to produce (xwarp, ywarp) using a set of user-supplied parameters. This can be used to produce large-scale variations in the texture, such as a gradual shift from bottom to top, or ripple effects.

The warped coordinates are subsequently distorted to produce (xdist, ydist). This can be used to change a regular cell pattern, such as a simple square grid, into something more ‘organic’, perhaps similar to the scales on crocodile skin. The distortion itself is generated using the plasma so the result is tileable. It also produces the appearance of stretched and compressed regions, rather than the ‘noisy’ image that would result from completely random distortion (Figure 33).

The distorted coordinates are subsequently analyzed to determine which cell they lie in (the basic cell types are shown in Figure 31). A record is made of the center of the cell (xcenter, ycenter) and the offset (xoffset, yoffset) of the distorted coordinates from the cell origin. In addition, a polar coordinate representation of the offset is calculated (roffset, aoffset), and eoffset, a variation on roffset representing distance to the nearest cell edge. Repeatable random values are derived from the cell center position, and from the distorted coordinates. Plasma values are looked up corresponding to the cell center (pcenter), the distorted coordinates (pdist), and a fixed offset from the distorted coordinates (poffset).

A weighted blend of (x, y, xwarp, ywarp, xdist, ydist, xoffset, yoffset, roffset, eoffset, aoffset, pcenter, pdist, poffset) is used to produce a pair of values which, after a further warping stage, are merged with two further weighted repeatable random values (randcell, randdist). The repeatability is achieved through cell specific seeding of a pseudo-random number generator (subclause B.4).

The resulting coordinates are used to interpolated between four colors, the result of which is the RGB pixel data for the texture at position (x, y).

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

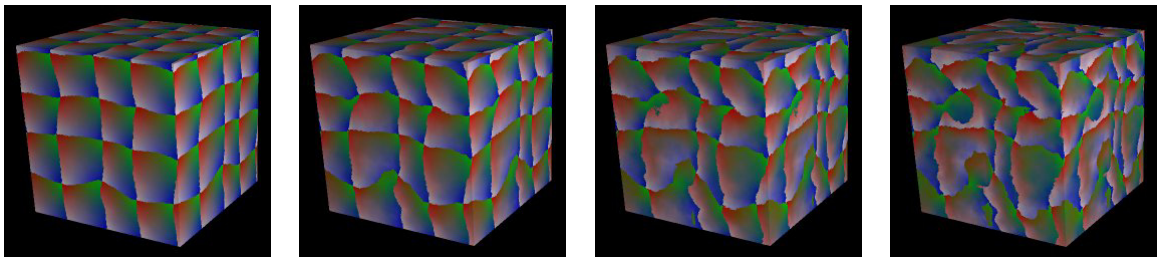


Figure 33 — The effects of distortion: (a) 25%, (b) 50%, (c) 75%, (d) 100%.

The generation of the texture (`T`) can be represented by the following pseudo-code:

```
P[] = plasma(width, height, roughness, seed)
for every pixel x, y
   x_warp = remap(x, xWarpmap[], xSmooth) * width
   y_warp = remap(y, yWarpmap[], ySmooth) * height
   x_dist = x_warp + P[(x_warp + width / 2) % width, y_warp] * width * distortion
   y_dist = y_warp + P[x_warp, (y_warp + height / 2) % height] * height * distortion
   Create vector v[] of 16 cell-related values:
      x, y
      x_warp, y_warp
      x_dist, y_dist
      x_offset, y_offset = offset of x_dist, y_dist from cell bottom-left
      r_offset = radial distance of x_dist, y_dist from cell center
      e_offset = distance towards nearest edge
      a_offset = angle of x_dist, y_dist around cell center
      p_center = P[x_dist, y_dist] // per-cell plasma value
      p_dist = P[x_center, y_center] // per-pixel plasma value
      p_offset = P[x_dist + width / 2, y_dist + height / 2]
      rand_cell = nonrandom(x_center, y_center) // per-cell random value
      rand_dist = nonrandom(x_dist, y_dist) // per-pixel random value
```

$$A = \frac{\sum_{i=0}^{15} v[i]aweights[i]}{\sum_{i=0}^{15} aweights[i]}$$

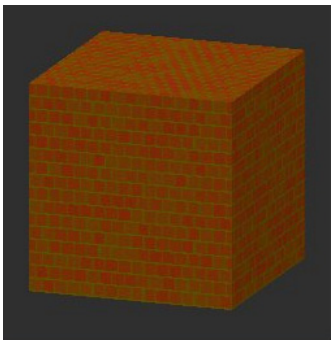
$$B = \frac{\sum_{i=0}^{15} v[i]bweights[i]}{\sum_{i=0}^{15} bweights[i]}$$

```
   A_warp = remap(A, aWarpmap[], aSmooth)
   B_warp = remap(B, bWarpmap[], bSmooth)
   c01 = color[0] + A_warp * (color[1] - color[0]);
   c23 = color[2] + A_warp * (color[3] - color[2]);
   T[x, y] = c01 + B_warp * (c23 - c01);
```

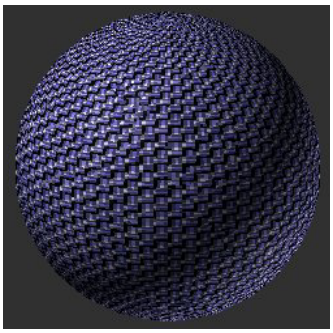
Where **width**, **height**, **roughness**, **distortion**, **seed**, **colors**, **xWarpmap**, **xSmooth**, **yWarpmap**, **ySmooth**, **aWarpmap**, **aSmooth**, **bWarpmap**, **bSmooth**, **aWeights** and **bWeights** correspond directly to the node fields while the procedures `plasma`, `remap` and `nonrandom` are defined in Annex B.2.

EXAMPLES

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved



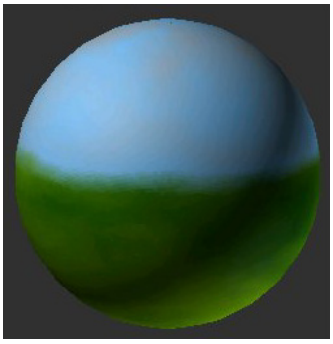
```
DEF Brickwork ProceduralTexture {
  Type 1
  CellWidth 8
  CellHeight 8
  roughness 2
  seed 63530
  color [ 0.447 0.43137 0, 0.6549 0.2 0.07843,
          0.447 0.43137 0, 0.447 0.43137 0 ]
  aWarpmap [ 0 0, 0.14 1, 0.83 1, 1 0 ]
  bWarpmap [ 0 0, 0.14 1, 0.83 1, 1 0 ]
  aWeights [ 0, 0, 0, 0, 0, 0, 0.48, 0, 0, 0, 0, 0, 0, 0,
0.40, 0.12 ]
  bWeights [ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 ]
}
```



```
DEF Fabric ProceduralTexture {
  type 2
  width 256
  height 256
  cellWidth 4
  cellHeight 4
  roughness 1
  distortion 0.05
  seed 114300
  color [ 0.898 0.89418 0.95294, 0.34118 0.29418 0.70196,
          0 0 0, 0 0 0 ]
  aWarpmap [ 0 0, 0.03 1, 0.88 1, 1 0 ]
  bWarpmap [ 0 0, 0.48 1, 1 0 ]
  aWeights [ 0, 0, 0, 0, 0, 0, 0.56, 0, 0, 0, 0, 0, 0, 0,
0.20, 0.24 ]
  bWeights [ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 ]
}
```



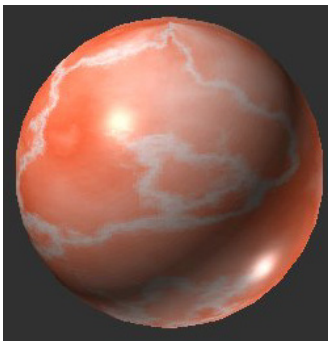
```
DEF GranitePink ProceduralTexture {
  width 256
  height 256
  cellWidth 8
  cellHeight 8
  roughness 5
  seed 36792
  color [ 0.72157 0.5647 0.5647, 0 0 0,
          0.91373 0.86275 0.86275, 0.80784 0.55294 0.51765 ]
  aWarpmap [ 0 0, 0.5 0, 0.5 1, 1 1 ]
  bWarpmap [ 0 0, 0.5 0, 0.5 1, 1 1 ]
  aWeights [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.90, 0, 0,
0.10 ]
  bWeights [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 ]
}
```



```
DEF Horizon ProceduralTexture {
  width 256
  height 256
  roughness 4
  distortion 2
  seed 108436
  color [ 0.30196 0.69804 1, 0.56078 0.56078 0.56078,
          0.05098 0.2549 0.03921, 0.4902 0.67059 0 ]
  bWarpmap [ 0 0, 0.55 0, 0.59 1, 1 1 ]
}
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



```
DEF Marble ProceduralTexture {
  width 256
  height 256
  roughness 1
  seed 22209
  color [ 0.8 0.7098 0.6902, 0.95686 0.8902 0.87451,
          0.87451 0.37255 0.23529, 0.95686 0.8902 0.87451 ]
  aWarpmap [ 0 1, 0.33 0, 1 1 ]
  bWarpmap [ 0 0, 0.55 0, 0.6 1, 0.65 0, 1 0 ]
  bWeights [ 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 ]
}
```

7.2.2.106 ProximitySensor

7.2.2.106.1 Node interface

ProximitySensor {			
exposedField	SFVec3f	**center**	0, 0, 0
exposedField	SFVec3f	**size**	0, 0, 0
exposedField	SFBool	**enabled**	TRUE
eventOut	SFBool	**isActive**	
eventOut	SFVec3f	**position_changed**	
eventOut	SFRotation	**orientation_changed**	
eventOut	SFTime	**enterTime**	
eventOut	SFTime	**exitTime**	
}			

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.106.2 Functionality and semantics

The semantics of the **ProximitySensor** node are specified in ISO/IEC 14772-1:1998, subclause 6.38.

7.2.2.107 ProximitySensor2D

7.2.2.107.1 Node interface

ProximitySensor2D {			
exposedField	SFVec2f	**center**	0, 0
exposedField	SFVec2f	**size**	0, 0
exposedField	SFBool	**enabled**	TRUE
eventOut	SFBool	**isActive**	
eventOut	SFVec2f	**position_changed**	
eventOut	SFFloat	**orientation_changed**	
eventOut	SFTime	**enterTime**	
eventOut	SFTime	**exitTime**	
}			

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.107.2 Functionality and semantics

This is the 2D equivalent of the **ProximitySensor** node (see 7.2.2.105) with semantics that are the 2D restriction of the that node.

The **orientation_changed** field is meaningless in 2D and can be ignored.

The **ProximitySensor2D** sensor generates an event when the pointing device (cursor) enters, exits, and moves within a region in space (defined by a rectangle according to **center** and **size** fields).

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.108 QuantizationParameter

7.2.2.108.1 Node interface

QuantizationParameter {

field	SFBool	**isLocal**	FALSE
field	SFBool	**position3DQuant**	FALSE
field	SFVec3f	**position3DMin**	-∞, -∞, -∞
field	SFVec3f	**position3DMax**	+∞, +∞, +∞
field	SFInt32	**position3DNbBits**	16
field	SFBool	**position2DQuant**	FALSE
field	SFVec2f	**position2DMin**	-∞, -∞
field	SFVec2f	**position2DMax**	+∞, +∞
field	SFInt32	**position2DNbBits**	16
field	SFBool	**drawOrderQuant**	FALSE
field	SFVec3f	**drawOrderMin**	-∞
field	SFVec3f	**drawOrderMax**	+∞
field	SFInt32	**drawOrderNbBits**	8
field	SFBool	**colorQuant**	TRUE
field	SFFloat	**colorMin**	0.0
field	SFFloat	**colorMax**	1.0
field	SFInt32	**colorNbBits**	8
field	SFBool	**textureCoordinateQuant**	TRUE
field	SFFloat	**textureCoordinateMin**	0.0
field	SFFloat	**textureCoordinateMax**	1.0
field	SFInt32	**textureCoordinateNbBits**	16
field	SFBool	**angleQuant**	TRUE
field	SFFloat	**angleMin**	0.0
field	SFFloat	**angleMax**	2π
field	SFInt32	**angleNbBits**	16
field	SFBool	**scaleQuant**	FALSE
field	SFFloat	**scaleMin**	0.0
field	SFFloat	**scaleMax**	+∞
field	SFInt32	**scaleNbBits**	8
field	SFBool	**keyQuant**	TRUE
field	SFFloat	**keyMin**	0.0
field	SFFloat	**keyMax**	1.0
field	SFInt32	**keyNbBits**	8
field	SFBool	**normalQuant**	TRUE
field	SFInt32	**normalNbBits**	8
field	SFBool	**sizeQuant**	FALSE
field	SFFloat	**sizeMin**	0.0
field	SFFloat	**sizeMax**	+∞
field	SFInt32	**sizeNbBits**	8
field	SFBool	**useEfficientCoding**	FALSE

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.108.2 Functionality and semantics

The **QuantizationParameter** node describes the quantization values to be applied on single fields of numerical types. For each of identified categories of fields, a minimal and maximal value is given as well as a number of bits to represent the given class of fields. Additionally, it is possible to set the **isLocal** field to apply the quantization only to the node following the **QuantizationParameter** node. The use of a node structure for declaring the quantization parameters allows the application of the DEF and USE mechanisms that enable reuse of the **QuantizationParameter** node. Also, it enables the parsing of this node in the same manner as any other scene information.

The **QuantizationParameter** node may only appear as a child of a grouping node. When a **QuantizationParameter** node appears in the scene graph, the quantization is set to TRUE, and will apply to subsequent nodes as follows:

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If the **isLocal** boolean is set to FALSE, the quantization applies to all siblings following the **QuanitzationParameter** node, and thus to all their children as well.

If the **isLocal** boolean is set to TRUE, the quantization only applies to the following sibling node in the children list of the parent node. If no sibling is following the **QuantizationParameter** node declaration, the node has no effect.

For each scene, by default, there is no quantization. A global quantizer can be defined or modified by using a GlobalQuantizationConfiguration command, see subclause 9.3.6.23. The global quantizer applies to all subsequent BIFS access units where no **QuantizationParameter** node apply. The global quantizer may be DEFed and then USEd within the scene. The global quantizer may be set to a USE of an existing **QuantizationParameter** present in the scene.

NOTE If DEFed, the global quantizer may be ROUTEd and/or modified by Script, MPEG-J and FieldReplacement commands.

Unless the **QuantizationParameter** node is present in a GlobalQuantizationConfiguration command, the quantization is applied only in the scope of a single BIFS command. That is, if a command in the same access unit, or in another access unit inserts a node in a context in which the quantization was active, no quantization will be applied, except if a new **QuantizationParameter** node is defined in this new command.

The information contained in the **QuantizationParameter** node fields applies within the context of the node scope as follows. For each category of fields, a boolean sets the quantization on or off, the minimal and maximal values are set, as well as the number of bits for the quantization. This information, combined with the node coding table, enables the relevant information to quantize the fields to be obtained. The quantization parameters are applied as explained in subclause 8.3.

If the **useEfficientCoding** boolean is set to FALSE, the encoding of floats shall be performed using 32 bits, according to IEEE Std 754-1985.

If the **useEfficientCoding** boolean is set to TRUE, the encoding of floats shall use the syntax described in subclause 8.7.19. The scope of the use of the efficient coding is the same as that of the **QuantizationParameter** node. This means that the values of the fields of the current **QuantizationParameter** node are not sent in the efficient coding mode unless the context is within the scope of a previously sent **QuantizationParameter** whose **useEfficientCoding** bit was set to true.

7.2.2.109 RadialGradient

7.2.2.109.1 Node interface

RadialGradient {

exposedField	SFNode	**transform**	NULL
exposedField	SFVec2f	**center**	0.5 0.5
exposedField	SFFloat	**radius**	0.5
exposedField	SFVec2f	**focalPoint**	0.5 0.5
exposedField	SFInt32	**spreadMethod**	0
exposedField	MFFloat	**key**	[]
exposedField	MFColor	**keyValue**	[]
exposedField	MFFloat	**opacity**	[1]

}

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.109.2 Functionality and semantics

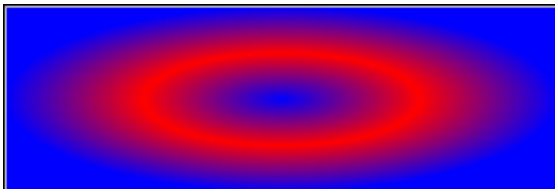
The **RadialGradient** node is a texture node that generates a texture procedurally.
The **center** and **radius** fields define the outermost circle (cx, cy, r) for the radial gradient. The gradient will be drawn such that 100% of the gradient stop is mapped to the perimeter of this outermost circle. The units for center and radius are in percentages of the bounds of the colored object. Length is x- and y-direction in the local coordinate system of the shape. By default, the center of the outermost circle is in the middle of the shape (0.5 means 50%).
focalPoint (fx, fy) represents the focal point of the gradient. The gradient will be drawn such that 0% of the gradient stop is at (fx, fy). The units are also in percentage of the bounds of the object.
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.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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.
key and **keyValue** define the ramp of colors to use on a gradient. At least two values are necessary to define a gradient. key indicates, in percentage, where the **keyValue** (a RGB color value) will be placed. key represents the percentage distance between the **focalPoint** (fx, fy) and the edge of the outermost circle (cx, cy, r). 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 must be specified for each color value.

EXAMPLE

```
Shape {
    geometry Rectangle { size 3 1 }
    appearance Appearance {
        texture RadialGradient {
            key [ 0 0.5 1 ]
            keyValue [ 0 0 1, 1 0 0, 0 0 1 ]
        }
}
```



7.2.2.110 Rectangle

7.2.2.110.1 Node interface

```
Rectangle {
    exposedField    SFVec2f        size                    2, 2
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.110.2 Functionality and semantics

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.

7.2.2.111 ScalarAnimator

7.2.2.111.1 Node Interface

```
ScalarAnimator {
    eventIn         SFFloat     set_fraction
    exposedField    SFVec2f     fromTo          0 1
    exposedField    MFFloat     key             []
    exposedField    SFInt32     keyType         0
    exposedField    MFVec2f     keySpline       [0 0, 1 1]
    exposedField    MFFloat     keyValue        []
    exposedField    MFFloat     weight          []
    exposedField    SFInt32     keyValueType    0
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    exposedField    SFFloat         offset                  0
    eventOut        SFFloat         value_changed
    eventOut        SFFloat         endValue
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.111.2 Functionality and semantics

Same semantic as for **PositionAnimator** except input and output key values are scalars (and no orientation is generated).

EXAMPLE Curve paths

This example shows how to define a cubic B-Spline.

```
DEF TS TimeSensor {
   stopTime -1
   cycleInterval 5
   loop TRUE
}

# linear animator
DEF PA1 PositionAnimator {
   fromTo 0.2 0.8
   key [ 0 0 0 0 0.25 0.5 0.75 1 1 1 1 ]
   keyValue [-2 0 0, -3 2 0, 2 2 0, 0 0 0, -1 -1 0, 3 -2 0, 2.5 0 0 ]
   keyValueType 3
   keyType 1
}

DEF XF1 Transform {
   children [
      Shape {
          appearance Appearance {
             material Material { diffuseColor 1 0 0 }
          }
          geometry Cone { height 1 bottomRadius 0.5 }
      }
     ]
}

ROUTE TS.fraction_changed TO PA1.set_fraction
ROUTE PA1.value_changed TO XF1.translation
ROUTE PA1.rotation_changed TO XF1.rotation
```

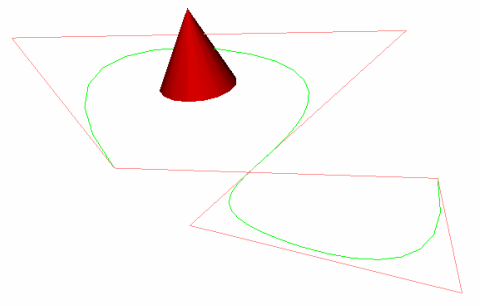


Figure 34 — Curve path (cubic B-spline), control points, and a cone moving along.

EXAMPLE Cumulating animations

Two animations are chained together but both use the same timer and output their position and orientation valued to the same node.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

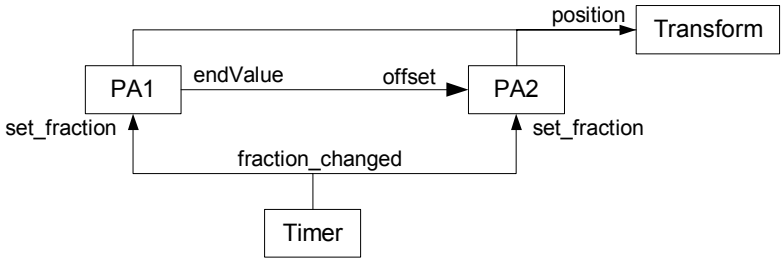


Figure 35 — Schematic of the events while cumulating two Animators.

```
DEF TIMER TimeSensor {
   stopTime -1
   cycleInterval 5
   loop TRUE
}


# linear animator
DEF PA1 PositionAnimator {
   fromTo 0 0.5
   key [ 0 0 0 0 0.25 0.5 0.75 1 1 1 1 ]
   keyValue [-2 0 0, -3 2 0, 2 2 0, 0 0 0, -1 -1 0, 3 -2 0, 2.5 0 0 ]
   keyValueType 3
   keyType 1
}

DEF PA2 PositionAnimator {
   fromTo 0.5 1
   key [ 0 0 0 0 1 1 1 1 ]
   keyValue [ 0 0 0, 1 3 1,3 3 -1, 4 0 2 ]
   keyValueType 3
   keyType 1
}

ROUTE PA1.endValue TO PA2.offset # cumul

DEF XF1 Transform {
   children [
      Shape {
    appearance Appearance { material Material { diffuseColor 1 0 0 } }
    geometry Cone { height 1 bottomRadius 0.5 }
      }
     ]
}
ROUTE TIMER.fraction_changed TO PA1.set_fraction
ROUTE TIMER.fraction_changed TO PA2.set_fraction
ROUTE PA1.value_changed TO XF1.translation
ROUTE PA1.rotation_changed TO XF1.rotation
ROUTE PA2.value_changed TO XF1.translation
ROUTE PA2.rotation_changed TO XF1.rotation
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

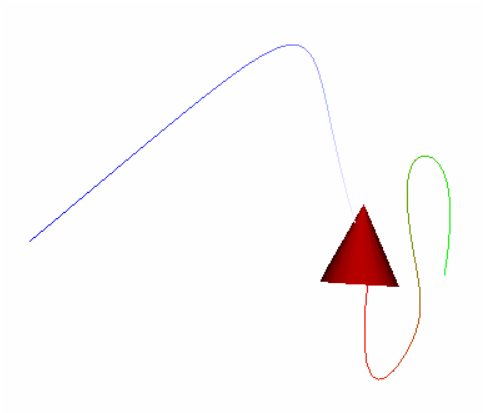


Figure 36 — Cumulating two Animators.

7.2.2.112 ScalarInterpolator

7.2.2.112.1 Node interface

```
ScalarInterpolator {
    eventIn         SFFloat     set_fraction
    exposedField    MFFloat     key                 []
    exposedField    MFFloat     keyValue            []
    eventOut        SFFloat     value_changed
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.112.2 Functionality and semantics

The semantics of the **ScalarInterpolator** node are specified in ISO/IEC 14772-1:1998, subclause 6.39.

7.2.2.113 Script

7.2.2.113.1 Node interface

```
Script {
    exposedField    MFString    url                 []
    field           SFBool      directOutput        FALSE
    field           SFBool      mustEvaluate        FALSE
    Any number of the following may then follow:
    eventIn         eventType   eventName
    field           fieldType   fieldName           initialValue
    eventOut        eventType   eventName
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.113.2 Functionality and semantics

The **Script** node is used to describe behaviour in a programmtic way 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

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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.

7.2.2.113.2.1 Detailed Semantics

The semantics of this node are as defined in ISO/IEC 14772-1:1998, subclause 6.40, with the following exception. The interface functions `CreateVRMLFromString()` and `CreateVRMLFromURL()` are not supported. The terminal shall support JavaScript.

EXAMPLE — The following scene contains two spheres that exchange colors when they are clicked with the mouse. The script is used to hold the current color state (in the variable `num`). The script variables `color1` and `color2` are used to hold the colors that are flipped back and forth between the two spheres. The script variable `color` is used to hold the last color state of the first sphere, and this color is routed to the second sphere. The first sphere color is set directly in the script.

```
Group {
   children [
      Viewpoint {
         fieldOfView 0.785398
      }
      DirectionalLight {
         color 1 1 1
      }
      Shape {
         geometry Sphere { radius 0.5}     # first sphere…
         appearance Appearance {
            material DEF COLOR Material {diffuseColor 1 0 0}
         }
      }
      Transform {
         translation -2 0 0
         children [
            Shape {
               geometry Sphere { radius 1.0} #second sphere…
               appearance Appearance {
               material DEF COLOR2 Material {diffuseColor 1 1 1}
               }
            }
            DEF TS TouchSensor{} #clicking on the 2nd sphere will activate the script
         ]
      }
      DEF SC Script {
         eventIn SFBool touch
         field SFNode node USE COLOR
         field SFColor color1 0 1 0 # constant color for sphere
         field SFColor color2 0 0 1 # same as above
         field SFInt32 num 1  # holds the current color state
         eventOut SFColor color # holds the last color in COLOR
         url "javascript:
            function touch (value, tp) {
               color = node.diffuseColor;
               if (num==1) {
                  node.diffuseColor = color1;
                  num = 2;
               } else {
                  node.diffuseColor = color2;
                  num = 1;
               }
            }
         "
      }
   ]
}
ROUTE TS.isActive TO SC.touch     # activates the script when sensor is touched
ROUTE SC.color TO COLOR2.diffuseColor # routes the last color of COLOR to COLOR2
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.114 ServerCommand

7.2.2.114.1 Node Interface

ServerCommand {			
eventIn	SFBool	**trigger**	
exposedField	SFBool	**enable**	FALSE
exposedField	MFString	**url**	[]
exposedField	SFString	**command**	""
}			

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.114.2 Functionality and Semantics

The **ServerCommand** in BIFS enables the application signaling in MPEG-4 Systems. The application-signaling framework allows an application to communicate the application signaling messages or commands to a server(s). Commands are sent to servers upon the occurrence of events (synchronous events specified in the scene description or asynchronous events as a result of user interaction). The **ServerCommand** framework consists of two elements; a **ServerCommand** node, and a **ServerCommandRequest** structure. While the **ServerCommand** enables event routing to the server, the **ServerCommandRequest** structure specifies the syntax for the messages communicated to the server over a back channel.

The **ServerCommand** is processed only when **trigger** receives a TRUE event and **enable** is TRUE. When the **ServerCommand** is processed, the **command** is sent to the servers indicated by the specified **url**. A **url** identifies the object descriptor that contains an elementary stream that flows from the terminal back to the server. If that object descriptor has more than one such elementary stream, then the one specified will be used. The **command** field contains the information that is transmitted back to the server. The syntax and semantics of the **command** string are application specific and not specified. The syntax of the **ServerCommandRequest** structures used to communicate the **command** to a server is specified below.

7.2.2.114.3 ServerCommandRequest

When the **ServerCommand** is processed the associated **command** is communicated to the servers specified in the **url** using the **ServerCommandRequest** structures. The **ServerCommandRequest** is encapsulated into SL packets, using the `SLConfigDescriptor` contained in the `ESDescriptor` of the upchannel elementary stream that carries the commands. If a timestamp is provided in the SL layer (either decoding or composition) then it is directly derived from the System Time Base of the terminal.

Syntax

```
class ServerCommandRequest(BIFSConfig cfg) {
     bit(cfg.nodeIDbits) nodeID;
     SFString  command;
   }
```

where `nodeID` is node ID of the **ServerCommand** node that triggered the command (all such nodes must have IDs in order to route events into them), and **command** is the string contained in the **ServerCommand** node's **command** field.

7.2.2.115 Shape

7.2.2.115.1 Node interface

Shape {			
exposedField	SFNode	**appearance**	NULL
exposedField	SFNode	**geometry**	NULL
}			

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.115.2 Functionality and semantics

The semantics of the **Shape** node are specified in ISO/IEC 14772-1:1998, subclause 6.41.

7.2.2.116 Sound

7.2.2.116.1 Node interface

Sound {

exposedField	SFVec3f	**direction**	0, 0, 1
exposedField	SFFloat	**intensity**	1.0
exposedField	SFVec3f	**location**	0, 0, 0
exposedField	SFFloat	**maxBack**	10.0
exposedField	SFFloat	**maxFront**	10.0
exposedField	SFFloat	**minBack**	1.0
exposedField	SFFloat	**minFront**	1.0
exposedField	SFFloat	**priority**	0.0
exposedField	SFNode	**source**	NULL
field	SFBool	**spatialize**	TRUE

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.116.2 Functionality and semantics

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 dscribed using ISO/IEC 14496-1 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, subclause 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 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 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 apparant 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,

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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.

EXAMPLE — **Sound** node S1 generates a spatialized sound s1 and five channels of multichannel ambient sound a1[1-5]. **Sound** node S2 generates a spatialized sound s2 and two channels of multichannel ambient sound a2[1-2]. S1 and S2 are grouped under a single **Group** node. The resulting sound is the superposition of the spatialized sound s1, the spatialized sound s2, and the five-channel ambient multichannel sound represented by a3[1-5], where

a3[1] = a1[1] + a2[1]

a3[2] = a1[2] + a2[2]

a3[3] = a1[3]

a3[4] = a1[4]

a3[5] = a1[5]

7.2.2.117 Sound2D

7.2.2.117.1 Node interface

Sound2D {

exposedField	SFFloat	**intensity**	1.0
exposedField	SFVec2f	**location**	0,0
exposedField	SFNode	**source**	NULL
field	SFBool	**spatialize**	TRUE

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.117.2 Functionality and semantics

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 (see 7.1.1.2.1). 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 one meter between the user and a 2D scene of size 2m x 1.5m, 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 (see 7.2.2.116). 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 as described in 7.2.2.116. If **Sound** and **Sound2D** nodes are both used in a scene, all shall be treated the same way according to these semantics.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.118 Sphere

7.2.2.118.1 Node interface

```
Sphere {
    field       SFFloat     Radius          1.0
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.118.2 Functionality and semantics

The semantics of the **Sphere** node are specified in ISO/IEC 14772-1:1998, subclause 6.43.

7.2.2.119 SphereSensor

7.2.2.119.1 Node interface

```
SphereSensor {
    exposedField    SFBool      autoOffset          TRUE
    exposedField    SFBool      enabled             TRUE
    exposedField    SFRotation  offset              0 1 0 0
    eventOut        SFBool      isActive
    eventOut        SFRotation  rotation_changed
    eventOut        SFVec3f     trackPoint_changed
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.119.2 Functionality and semantics

The semantics of the **SphereSensor** node are specified in ISO/IEC 14772-1:1998, subclause 6.44.

7.2.2.120 SpotLight

7.2.2.120.1 Node interface

```
SpotLight {
    exposedField    SFFloat     ambientIntensity    0.0
    exposedField    SFVec3f     attenuation         1, 0, 0
    exposedField    SFFloat     beamWidth           1.5708
    exposedField    SFColor     color               1, 1, 1
    exposedField    SFFloat     cutOffAngle         0.785398
    exposedField    SFVec3f     direction           0, 0, -1
    exposedField    SFFloat     intensity           1.0
    exposedField    SFVec3f     location            0, 0, 0
    exposedField    SFBool      on                  TRUE
    exposedField    SFFloat     radius              100.0
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.120.2 Functionality and semantics

The semantics of the **SpotLight** node are specified in ISO/IEC 14772-1:1998, subclause 6.45.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.121 SurroundingSound

7.2.2.121.1 Node interface

SurroundingSound {

exposedField	SFNode	**source**	NULL
exposedField	SFFloat	**intensity**	1.0
exposedField	SFFloat	**distance**	0.0
exposedField	SFVec3f	**location**	0, 0, 0
exposedField	SFFloat	**distortionFactor**	0.0
exposedField	SFRotation	**orientation**	0, 0, 1, 0
exposedField	SFBool	**isTransformable**	TRUE

}

NOTE — For binary encoding of this node see node coding tables in electronic attachment.

7.2.2.121.2 Functionality and semantics

The **SurroundingSound** node is used to attach sound to a scene. This causes spatial qualities and makes it related to the visual content of the scene. This includes multichannel signals that cannot be spatially transformed with the other sound nodes due to their restrictions to the specification of the **phaseGroup** field and the **spatialize** field.

The **SurroundingSound** 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 14496-1 may be filtered and mixed before being spatially composited into the scene.

The Ambisonics® coordinate system as well as the one used for multichannel setups differs from the one used in MPEG-4 scene description. A **Transform** or **Transform3DAudio** should be used to align the coordinate systems.

The **source** field allows connection to an audio sub-graph containing the sound. The **phaseGroup** field of the child node shall be ignored. Instead an **AudioChannelConfig** node has to be included into the audio subtree to deliver the required channel configuration information, e.g. Ambisonics® or multichannel format information.

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 **distance** field describes how the intensity of the sound field shall be faded with increasing distance of the listener (**ListeningPoint**) from the **location** of the **SurroundingSound**. Its functionality is identical to the distance dependent attenuation of the sound described in subclause 7.2.2.46.2 **DirectiveSound**. If **distance** is set to 0, no distance dependent fading shall be applied.

The **location** field specifies the center of the sound field in the local coordinate system of the **SurroundingSound.**

The **orientation** field is the orientation given to the sound field in the scene, with regard to the local coordinate system of the considered **SurroundingSound** node. This supposes applying sound field rotations at rendering stage.

The **distortionFactor** field describes the strength of the angular distortion to be applied on the surrounding sound field when the listener moves from the sound field's reference point (**SurroundingSound**'s **location**). This distortion effect assumes that "side" sources (*i.e.* in directions orthogonal to the motion) contained in the sound field, are at the same distance 1/**distortionFactor** from the reference point so that a small (*e.g.* forward) displacement *d* of the Listener produce a (resp. backward) angular change for side sources: $\phi \approx \tan\phi = d \times$ **distortionFactor**. At the same time, frontal and back scenes respectively enlarge and narrow in a way that should be as continuous and convincing as possible. Nevertheless the precise distortion in these latter sectors is not normative, since it may not be independently controlled. An informative part (see 7.1.1.2.13.6) provides suitable formulae for the case of Ambisonics® sound fields and suggests methods for other multi-channel contents. The default zero value means that no distortion is applied whatever the **ListeningPoint**'s position. **DistortionFactor** should take positive values in case of effects are consistent with the motion. Negative values may be used for supernatural effects.

isTransformable specifies whether the transformations from the transform hierarchy have to be applied or not.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2005 – All rights reserved
Not for Resale

7.2.2.122 Switch

7.2.2.122.1 Node interface

```
Switch {
   exposedField   MFNode      choice          []
   exposedField   SFInt32     whichChoice     -1
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.122.2 Functionality and semantics

The semantics of the **Switch** node are specified in ISO/IEC 14772-1:1998, subclause 6.46, with the following restrictions.

If some of the child sub-graphs contain audio content (i.e., the subgraphs contain **Sound** nodes), the child sounds are switched on and off according to the value of the **whichChoice** field. That is, only sound that corresponds to **Sound** nodes in the **whichChoice'th** subgraph of this node are played. The others are muted.

7.2.2.123 TemporalGroup

The **TemporalGroup** node carries the temporal constraints of its child nodes that will be used by the FlexTime model (or Advanced Synchronization Model). The FlexTime Model supports synchronization of objects from multiple sources with possibly different time bases. The FlexTime Model specifies timing using a flexible, constraint-based timing model. In this model, media objects can be linked to one another in a time graph using relationship constraints such as "CoStart", "CoEnd", or "Meet". And, in addition, to allow some flexibility to meet these constraints, each object may have a flexible duration with specific stretch and shrink mode preferences that may be applied.

The FlexTime model is based upon a so-called "spring" metaphor. A spring has a set of three constants: the minimum length below which it will not shrink, the maximum length beyond which it will not stretch, and the optimal length at which it rests comfortably being neither compressed nor extended. Following this spring model, the temporal playback of media objects can be viewed as springs, with a set of playback durations corresponding to these three spring constants. The optimal playback duration (optimal spring length) can be viewed as the author's preferred choice of playback duration for the media object. A player should, where possible, keep the playback length as close to the optimal duration as the presentation allows but may choose any duration between the minimum and maximum durations as specified by the author. Note, that whereas stretching or shrinking the duration continuous media, e.g. for video, implies respectively slowing down or speeding up playback, for discrete media such as a still image, shrinking or stretching is merely adjusting the rendering period to be shorter or longer.

The FlexTime model requires a small change to the MPEG-4 buffer model in terms of media delivery and decoding. Decoding may be delayed on the client, beyond the standard decoding time, by an amount determined by the flexibility expressed in the relationships. The buffer model for FlexTime can thus be specified as follows: "At any time from the instant of time corresponding to its DTS up to a time limit specified by FlexTime, and AU is instantaneously decoded and removed from the decoding buffer."

To support synchronization of nodes within the scene to a media stream, or part thereof, a new node supporting flexible transformation to scene time is introduced. This grouping node is the **TemporalTransform** and can flexibly support the slowing down, speeding up, freezing or shifting of the scene time for rendering of nodes contained within. This transform node is also a grouping node and provides the flexible component for the FlexTime model.

The **TemporalGroup** provides the constraint for the FlexTime model and gives it the tools it needs to align in time both nodes and media streams with nodes in the scene graph. **TemporalGroup** can examine the temporal properties of its children, check for the availability of media in the composition buffer, and consequently decide which temporal transformation parameters to apply to each of its child nodes.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.123.1 Node Interface

TemporalGroup {			
eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
field	SFBool	**costart**	TRUE
field	SFBool	**coend**	FALSE
field	SFBool	**meet**	FALSE
exposedField	MFFloat	**priority**	[]
eventOut	SFBool	**isActive**	
eventOut	SFInt32	**activeChild**	
}			

NOTE — For the binary encoding of this node see subclause node coding tables in electronic attachment.

7.2.2.123.2 Functionality and semantics

The **TemporalGroup** node specifies the temporal relationship between a given number of **TemporalTransform** nodes.

The **children** field specifies the list of **TemporalTransform** or **TemporalGroup** nodes on which the constraint is applied.

The **costart**, **coend** and **meet** fields specify the temporal relationships amongst the node's **children**. If **costart** is TRUE, all child nodes must be activated (start) together. If **coend** is TRUE, all child nodes must be deactivated (end) together. When **meet** is TRUE, the child nodes are activated one after another in a row. When one node ends, the next node in the list needs to start. If either **costart** or **coend** are set to TRUE the **meet** field is ignored.

The **priority** field specifies the list of priority numbers that determines the preferred scaling direction when two child nodes need to meet a constraint. The list of priorities is in the same order as the **children** field. More than one child can have the same value. In the case of **coend** the highest priority object will determine the end and cause all other objects to end at that time, providing all objects have at least reached their minimum durations. If the field is empty all nodes are assumed to have equal priority.

The **isActive** eventOut is triggered at the following events:

If **costart** is true a TRUE value will be sent when the co-start constraint is met.

If **coend** is true a FALSE value will be sent when the co-end constraint is met.

If **meet** is true a TRUE value will be sent when the first child is activated, and a FALSE value will be sent when the last one finishes.

The **activeChild** eventOut is sent when a new child is activated under a **meet** constant and will indicate the index of that child. The first child is index 0.

7.2.2.124 TemporalTransform

TemporalTransform is a grouping node that assigns temporal properties, and applies temporal transformation, to scene nodes and elementary streams.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.124.1 Node Interface

TemporalTransform {			
eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	MFString	**url**	[]
exposedField	SFTime	**startTime**	-1.0
exposedField	SFTime	**optimalDuration**	-1.0
exposedField	SFBool	**active**	FALSE
exposedField	SFFloat	**speed**	1.0
exposedField	SFVec2F	**scalability**	[1.0, 1.0]
exposedField	MFInt32	**stretchMode**	[0]
exposedField	MFInt32	**shrinkMode**	[0]
exposedField	SFTime	**maxDelay**	0
eventOut	SFTime	**actualDuration**	
}			

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.124.2 Functionality and semantics

The **TemporalTransform** node serves two purposes:

To apply temporal transformation to media objects.

To hold properties that will be used when the node has as a parent a **TemporalGroup** node.

The node operates on two types of objects. Its **children** field may contain a list of nodes of the type SF3DNode. In addition, it has a **url** field that may reference an elementary stream. In the first case, the node has the effect of slowing down, speeding up, freezing or shifting the time base of the compositor when it renders the child nodes that are transformed by the node. In the second case, the node affects the time base of the stream. Note that a Route between two nodes whose time bases are different, because one or both are affected by a **TemporalTransform**, will have undefined behavior. Also, a node with **startTime** / **StopTime** fields which is DEFed and USEd under different **TemporalTransform**s will have undefined behavior.

The **startTime** specifies the starting point of the media stream relative to the composition time of the first access unit received from the **url** that is controlled by this node. If **startTime** is negative, the entire media referred by this **url** is controlled.

The **optimalDuration** field specifies the nominal duration of the objects that are controlled by this node. This is also the optimal duration, which the FlexTime model opts for when scaling this node. If **optimalDuration** is negative, or outside the bounds defined by the **scalability** field, optimal duration is not available.

The **active** field determines whether the node, its children, and the stream controlled by the node are active. When the node is inactive, the time base of the compositor is frozen when the child nodes are composed. This means that:

The nodes are not visible and the stream is not played.

Timed nodes, e.g. **TimeSensor**, do not have their time running.

Node fields such as **startTime** and **stopTime** are processed as if the time is not running.

Nodes that react to user interaction, such as **TouchSensor**, or to their spatial position, such as **ProximitySensors** cannot be activated.

However operations that would normally be performed at that time are still performed, even if the node is frozen. For instance:

Script nodes are executed if activated.

ROUTEs are executed.

eventIns are processed (with no rendering).

DMIF (network stack) methods are called if necessary. Therefore the delivery of streams, if required, will be requested, even though their sync layer time base is frozen.

Another field that affects the temporal transformation is **speed**. When the value of this field is not 1 and the node is active, the scene time base of the node, its children, and the time base of the stream will slow down or speed up according to

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

this factor. If **speed** is set to zero, the node remains active but its time stops. Therefore time-related operations behave as if time is constant, and audio rendering pauses.

The other fields of the **TemporalTransform** node have no effect on the execution of the node, but are used by a parent **TemporalGroup** node to determine the temporal layout of the node in relation to other **TemporalTransform** objects.

The **scalability** field specifies the maximum ratios by which this object is allowed to shrink or stretch. If a nominal duration is known, either from **optimalDuration** or the stream length, the ratio determines the absolute values of the minimum and maximum duration. Otherwise, for unknown duration, the field dictates either the ratio by which the time bases controlled by the node are allowed to scale; or, when **optimalDuration** lies outside the bound of the values calculated, they are minimum and maximum durations (optimal duration unknown).

The **stretchMode** field specifies an ordered list of the preferred modes of stretching according to the table below.

The **shrinkMode** field specifies an ordered list of the preferred modes of shrinking according to the table below.

Table 23 — Preferred Mode of Stretch/Shrink Values.

StretchMode Value	StretchMode Description
0	Hold rendering of the last Access Unit
1	Linear Composition Unit rendering rate decrease
2	Repeat

ShrinkMode Value	ShrinkMode Description
0	Stop rendering
1	Linear Composition Unit rendering rate increase

The **maxDelay** field specifies how long the FlexTime model can wait for the stream specified by the **url** field. If this time elapses before the stream is available, the model behaves as if the node starts at that time, but the player will not render the children of this node and will discard the stream if it arrives later.

The **actualDuration** eventOut is triggered when the node is activated and sends the value of the estimated actual play duration of the node.

7.2.2.125 TermCap

7.2.2.125.1 Node interface

```
TermCap {
    eventIn     SFTime      evaluate
    field       SFInt32     capability      0
    eventOut    SFInt32     value
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.125.2 Functionality and semantics

The **TermCap** node is used to query the resources of the terminal. By ROUTEing the result to a **Switch** node, simple adaptive content may be authored using BIFS.

When this node is instantiated, the value of the **capability** field shall be examined by the system and the **value** eventOut generated to indicate the associated system capability. The **value** eventOut is updated and generated whenever an **evaluate** eventIn is received.

The **capability** field specifies a terminal resource to query. The semantics of the **value** field vary depending on the value of this field. The capabilities which may be queried are:

Table 24 — Semantics of value, dependent on capability

capability	Semantics of value
0	frame rate
1	color depth

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2	screen size
3	graphics hardware
32	audio output format
33	maximum audio sampling rate
34	spatial audio capability
64	CPU load
65	memory load

The exact semantics differ depending on the value of the **capability** field, as follows.

capability: 0 (frame rate)

For this value of **capability**, the current rendering frame rate is measured. The exact method of measurement not specified.

Table 25 — Semantics of value for capability=0

value	**Semantics**
0	unknown or can't determine
1	less than 5 fps
2	5-10 fps
3	10-20 fps
4	20-40 fps
5	more than 40 fps

For the breakpoint between overlapping values between each range (i.e. 5, 10, 20, and 40), the higher value of **value** shall be used (ie, 2, 3, 4, and 5 respectively). This applies to each of the subsequent **capability-value** tables as well.

capability: 1 (color depth)

For this value of **capability**, the color depth of the rendering terminal is measured. At the time this node is instantiated, the **value** field is set to indicate the color depth as follows:

Table 26 — Semantics of value for capability=1

value	**Semantics**
0	unknown or can't determine
1	1 bit/pixel
2	grayscale
3	color, 3-12 bit/pixel
4	color, 12-24 bit/pixel
5	color, more than 24 bit/pixel

capability: 2 (screen size)

For this value of **capability**, the window size (in horizontal lines) of the output window of the rendering terminal is measured:

Table 27 — Semantics of value for capability=2

value	**Semantics**
0	unknown or can't determine
1	less than 200 lines
2	200-400 lines
3	400-800 lines
4	800-1600 lines
5	1600 or more lines

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

capability: 3 (graphics hardware)

For this value of **capability**, the available of graphics acceleration hardware of the rendering terminal is measured. At the time this node is instantiated, the **value** field is set to indicate the available graphics hardware:

Table 28 — Semantics of value for capability=3

value	**Semantics**
0	unknown or can't determine
1	no acceleration
2	matrix multiplication
3	matrix multiplication + texture mapping (less than 1M memory)
4	matrix multiplication + texture mapping (less than 4M memory)
5	matrix multiplication + texture mapping (more than 4M memory)

capability: 32 (audio output format)

For this value of **capability**, the audio output format (speaker configuration) of the rendering terminal is measured. At the time this node is instantiated, the **value** field is set to indicate the audio output format.

Table 29 — Semantics of value for capability=32

value	**Semantics**
0	unknown or can't determine
1	mono
2	stereo speakers
3	stereo headphones
4	five-channel surround
5	more than five speakers

capability: 33 (maximum audio sampling rate)

For this value of **capability**, the maximum audio output sampling rate of the rendering terminal is measured. At the time this node is instantiated, the **value** field is set to indicate the maximum audio output sampling rate.

Table 30 — Semantics of value for capability=33

value	**Semantics**
0	unknown or can't determine
1	less than 16000 Hz
2	16000-32000 Hz
3	32000-44100 Hz
4	44100-48000 Hz
5	48000 Hz or more

capability: 34 (spatial audio capability)

For this value of **capability**, the spatial audio capability of the rendering terminal is measured. At the time this node is instantiated, the **value** field is set to indicate the spatial audio capability.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 31 — Semantics of value for capability=34

value	**Semantics**
0	unknown or can't determine
1	no spatial audio
2	panning only
3	azimuth only
4	full 3-D spatial audio

capability: 64 (CPU load)

For this value of **capability**, the CPU load of the rendering terminal is measured. The exact method of measurement is not specified. The value of the **value** eventOut indicates the available CPU resources as a percentage of the maximum available; that is, if all of the CPU cycles are being consumed, and no extra calculation can be performed without compromising real-time performance, the indicated value is 100%; if twice as much calculation as currently being done can be so performed, the indicated value is 50%.

Table 32 — Semantics of value for capability=64

value	**Semantics**
0	unknown or can't determine
1	less than 20% loaded
2	20-40% loaded
3	40-60% loaded
4	60-80% loaded
5	80-100% loaded

capability: 65 (RAM available)

For this value of **capability**, the available memory of the rendering terminal is measured. The exact method of measurement is not specified.

Table 33 — Semantics of value for capability=65

value	**Semantics**
0	unknown or can't determine
1	less than 100 KB free
2	100 KB – 500 KB free
3	500 KB – 2 MB free
4	2 MB – 8 MB free
5	8 MB – 32 MB free
6	32 MB – 200 MB free
7	more than 200 MB free

7.2.2.126 Text

7.2.2.126.1 Node interface

```
Text {
    exposedField    MFString    string          []
    exposedField    MFFloat     length          []
    exposedField    SFNode      fontStyle       NULL
    exposedField    SFFloat     maxExtent       0.0
}
```

--`,,```,,,,````-`-,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.126.2 Functionality and semantics

The semantics of the **Text** node are specified in ISO/IEC 14772-1:1998, subclause 6.47. When text is textured in a 2D context, the default texture mapping coordinates are defined as the four corners of the bounding box of the complete rendered text string.

7.2.2.127 TextureCoordinate

7.2.2.127.1 Node interface

```
TextureCoordinate {
    exposedField    MFVec2f     point                   []
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.127.2 Functionality and semantics

The semantics of the **TextureCoordinate** node are specified in ISO/IEC 14772-1:1998, subclause 6.48.

7.2.2.128 TextureTransform

7.2.2.128.1 Node interface

```
TextureTransform {
    exposedField    SFVec2f     center          0, 0
    exposedField    SFFloat     rotation        0.0
    exposedField    SFVec2f     scale           1, 1
    exposedField    SFVec2f     translation     0, 0
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.128.2 Functionality and semantics

The semantics of the **TextureTransform** node are specified in ISO/IEC 14772-1:1998, subclause 6.49.

7.2.2.129 TimeSensor

7.2.2.129.1 Node interface

```
TimeSensor {
    exposedField    SFTime      cycleInterval       1
    exposedField    SFBool      enabled             TRUE
    exposedField    SFBool      loop                FALSE
    exposedField    SFTime      startTime           0
    exposedField    SFTime      stopTime            0
    eventOut        SFTime      cycleTime
    eventOut        SFFloat     fraction_changed
    eventOut        SFBool      isActive
    eventOut        SFTime      time
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.129.2 Functionality and semantics

The semantics of the **TimeSensor** node are specified in ISO/IEC 14772-1:1998, subclause 6.50.

7.2.2.130 TouchSensor

7.2.2.130.1 Node interface

TouchSensor {			
exposedField	SFBool	**enabled**	TRUE
eventOut	SFVec3f	**hitNormal_changed**	
eventOut	SFVec3f	**hitPoint_changed**	
eventOut	SFVec2f	**hitTexCoord_changed**	
eventOut	SFBool	**isActive**	
eventOut	SFBool	**isOver**	
eventOut	SFTime	**touchTime**	
}			

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.130.2 Functionality and semantics

The semantics of the **TouchSensor** node are specified in ISO/IEC 14772-1:1998, subclause 6.51.

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.

7.2.2.131 Transform

7.2.2.131.1 Node interface

Transform {			
eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	SFVec3f	**center**	0, 0, 0
exposedField	MFNode	**children**	[]
exposedField	SFRotation	**rotation**	0, 0, 1, 0
exposedField	SFVec3f	**scale**	1, 1, 1
exposedField	SFRotation	**scaleOrientation**	0, 0, 1, 0
exposedField	SFVec3f	**translation**	0, 0, 0
}			

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.131.2 Functionality and semantics

The semantics of the **Transform** node are specified in ISO/IEC 14772-1:1998, subclause 6.52. ISO/IEC 14496-1 does not support the bounding box parameters (**bboxCenter** and **bboxSize**).

If some of the child subgraphs contain audio content (i.e., the subgraphs contain **Sound** nodes), the child sounds are transformed and mixed as follows.

If each of the child sounds is a spatially presented sound, the **Transform** node applies to the local coordinate system of the **Sound** nodes to alter the apparent spatial location and direction. If the children are not spatially presented but have equal numbers of channels, the **Transform** node has no effect on the childrens' sounds. After any such transformation, the combination of sounds is performed as described in 7.2.2.116.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If the children are not spatially presented but have equal numbers of channels, the **Transform** node has no effect on the childrens' sounds. The child sounds are summed equally to produce the audio output at this node.

If some children are spatially presented and some not, or all children do not have equal numbers of channels, the semantics are not defined.

7.2.2.132 Transform2D

7.2.2.132.1 Node interface

Transform2D {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	SFVec2f	**center**	0, 0
exposedField	MFNode	**children**	[]
exposedField	SFFloat	**rotationAngle**	0.0
exposedField	SFVec2f	**scale**	1, 1
exposedField	SFFloat	**scaleOrientation**	0.0
exposedField	SFVec2f	**translation**	0, 0

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.132.2 Functionality and semantics

The **Transform2D** node allows the translation, rotation and scaling of its 2D children objects.

The **rotation** field specifies a rotation of the child objects, in radians, which occurs about the point specified by **center**.

The **scale** field specifies a 2D scaling of the child objects. The scaling operation takes place following a rotation of the 2D coordinate system that is specified, in radians, by the **scaleOrientation** field. The rotation of the co-ordinate system is notional and purely for the purpose of applying the scaling and is undone before any further actions are performed. No permanent rotation of the co-ordinate system is implied.

The **translation** field specifies a 2D vector which translates the child objects.

The scaling, rotation and translation are applied in the following order: scale, rotate, translate.

The **children** field contains a list of zero or more children nodes which are grouped by the **Transform2D** node.

The **addChildren** and **removeChildren** eventIns are used to add or remove child nodes from the **children** field of the node. Children are added to the end of the list of children and special note should be taken of the implications of this for implicit drawing orders.

If some of the child subgraphs contain audio content (i.e., the subgraphs contain **Sound** nodes), the child sounds are transformed and mixed as follows.

If each of the child sounds is a spatially presented sound, the **Transform2D** node applies to the local coordinate system of the **Sound2D** nodes to alter the apparent spatial location and direction. If the children are not spatially presented but have equal numbers of channels, the **Transform2D** node has no effect on the childrens' sounds. After any such transformation, the combination of sounds is performed as described in 7.2.2.117.

If the children are not spatially presented but have equal numbers of channels, the **Transform** node has no effect on the children's sounds. The child sounds are summed equally to produce the audio output at this node.

If some children are spatially presented and some not, or all children do not have equal numbers of channels, the semantics are not defined.

7.2.2.133 Transform3DAudio

7.2.2.133.1 Node interface

Transform3DAudio {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[ ]

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

exposedField	SFFloat	**thirdCenterCoordinate**	0
exposedField	SFVec3f	**rotationVector**	[0, 0, 1]
exposedField	SFFloat	**thirdScaleCoordinate**	0.0
exposedField	SFVec3f	**scaleOrientationVector**	[0, 0, 1]
exposedField	SFFloat	**thirdTranslationCoordinate**	0.0
exposedField	SFRotation	**coordinateTransform**	[1, 0, 0, -π/2]

}

NOTE — For binary encoding of this node see node coding tables in electronic attachment.

7.2.2.133.2 Functionality and semantics

The node is used for adaptation of 2D visual scenes to 3D audio subtrees. The basic functionality is similar to a **Transform(2D)** node above the Sound nodes with additional functionalities:

- Add additional coordinates and vectors to the input data coming from the 2D transform hierarchy to achieve a full 3D addressing, if desired. This enables a full 3D functionality of the 3D audio nodes in a 2D visual context.
- Rotate this new coordinates and rotations about a desired value. This enables for example the x-y to x-z plane mapping, which is the default audio plane.

The context of the 2D transform hierarchy contains the following information: **center**, **rotationAngle**, **scale**, **scaleOrientation** and **translation**. These values have to be extended to a 3D context by using the fields **thirdCenterCoordinate**, **rotationVector**, **thirdScaleCoordinate**, **scaleOrientationVector** and **thirdTranslationCoordinate** as described in Table 34.

The rotation of the coordinate system is determined by **coordinateTransform**. The default value maps the x-y position into the x-z plane. In this case the z components from **thirdCenterCoordinate**, **thirdScaleCoordinate** and **thirdTranslationCoordinate** address the y-components of the sound object.

Table 34 — Field functionality table

Node field			Input from the transform hierarchy			Is composed as output to target field for 3D descendant node(s)	
thirdCenterCoordinate	z	SFFloat	**center**	{x, y}	SFVec2f	SFVec3f	{x, y, z}
rotationVector	{x, y, z}	SFVec3f	**rotationAngle**	φ	SFFloat	SFRotation	{x, y, z, φ}
thirdScaleCoordinate	z	SFFloat	**scale**	{x, y}	SFVec2f	SFVec3f	{x, y, z}
scaleOrientationVector	{x, y, z}	SFVec3f	**scaleOrientation**	φ	SFFloat	SFRotation	{x, y, z, φ}
thirdTranslationCoordinate	z	SFFloat	**translation**	{x, y}	SFVec2f	SFVec3f	{x, y, z}

7.2.2.134 TransformMatrix2D

7.2.2.134.1 Node interface

TransformMatrix2D **{**

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFFloat	**mxx**	1
exposedField	SFFloat	**mxy**	0
exposedField	SFFloat	**tx**	0
exposedField	SFFloat	**myx**	0
exposedField	SFFloat	**myy**	1
exposedField	SFFloat	**ty**	0

}

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.134.2 Functionality and semantics

The **TransformMatrix2D** node is a grouping node that defines a coordinate system for its children that is relative to the coordinate systems of its ancestors. See ISO/IEC 14772-1:1998 for a description of coordinate systems and transformations and for a description of the **children**, **addChildren**, and **removeChildren** fields and eventIns.

The **mxx, mxy, tx, myx, myy** and **ty** fields define a geometric 2D transformation based on the following transformation matrix:

$$T = \begin{pmatrix} m_{xx} & m_{xy} & t_x \\ m_{yx} & m_{yy} & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

Given a 2-dimensional point P and **TransformMatrix2D** node, P is transformed into point P' in its parent's coordinate system by the transformation whose matrix is T.

$$P' = T \times P$$

The behaviour of **TransformMatrix2D** with respect to the **Sound2D** node is the same as the behaviour of the **Transform2D** node.

The **addChildren** and **removeChildren** eventIns are used to add or remove child nodes from the children field of the node as for a **Transform2D** node.

7.2.2.135 Valuator

7.2.2.135.1 Node interface

Valuator {

eventIn	SFBool	**inSFBool**
eventIn	SFColor	**inSFColor**
eventIn	MFColor	**inMFColor**
eventIn	SFFloat	**inSFFloat**
eventIn	MFFloat	**inMFFloat**
eventIn	SFInt32	**inSFInt32**
eventIn	MFInt32	**inMFInt32**
eventIn	SFRotation	**inSFRotation**
eventIn	MFRotation	**inMFRotation**
eventIn	SFString	**inSFString**
eventIn	MFString	**inMFString**
eventIn	SFTime	**inSFTime**
eventIn	SFVec2f	**inSFVec2f**
eventIn	MFVec2f	**inMFVec2f**

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

eventIn	SFVec3f	**inSFVec3f**	
eventIn	MFVec3f	**inMFVec3f**	
eventOut	SFBool	**outSFBool**	
eventOut	SFColor	**outSFColor**	
eventOut	MFColor	**outMFColor**	
eventOut	SFFloat	**outSFFloat**	
eventOut	MFFloat	**outMFFloat**	
eventOut	SFInt32	**outSFInt32**	
eventOut	MFInt32	**outMFInt32**	
eventOut	SFRotation	**outSFRotation**	
eventOut	MFRotation	**outMFRotation**	
eventOut	SFString	**outSFString**	
eventOut	MFString	**outMFString**	
eventOut	SFTime	**outSFTime**	
eventOut	SFVec2f	**outSFVec2f**	
eventOut	MFVec2f	**outMFVec2f**	
eventOut	SFVec3f	**outSFVec3f**	
eventOut	MFVec3f	**outMFVec3f**	
exposedField	SFFloat	**Factor1**	1.0
exposedField	SFFloat	**Factor2**	1.0
exposedField	SFFloat	**Factor3**	1.0
exposedField	SFFloat	**Factor4**	1.0
exposedField	SFFloat	**Offset1**	0.0
exposedField	SFFloat	**Offset2**	0.0
exposedField	SFFloat	**Offset3**	0.0
exposedField	SFFloat	**Offset4**	0.0
exposedField	SFBool	**Sum**	FALSE

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.135.2 Functionality and semantics

The **Valuator** node serves as a simple type casting method. It can receive events of multiple types. On reception of such an event, eventOuts of many different types can be generated. Both the eventIn and the eventOut values can be single field (SF) or multiple field (MF) types. In addition, the possible eventIn and eventOut types include both scalar types, like SFBool, and vector types, like SFVec2f.

Each component of the (possibly vector) eventOut value is calculated from the corresponding component of the (possibly vector) eventIn value with the following relationship that is also visualized in Figure 37:

output.i = factor.i * input.i + offset.i

All values specified in the above equation are floating point values.

input.i is the value of the ith component of the eventIn type and output.i is the value of the ith component of one of the eventOut types specified in the node inteface. input.i shall be extended by zeros for all components i that do not exist in the input type (e.g., input.z=0.0 in case an SFVec2f is cast to an SFVec3f). factor.i and offset.i are the exposedField values for the ith component of the vectorial calculation.

In the special case of a scalar input type (e.g. SFBool, SFInt32) that is cast to a vectorial output type (e.g. SFVec2f), for all components i of output.i, input.i shall take the value of the scalar input type, after appropriate type conversion.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

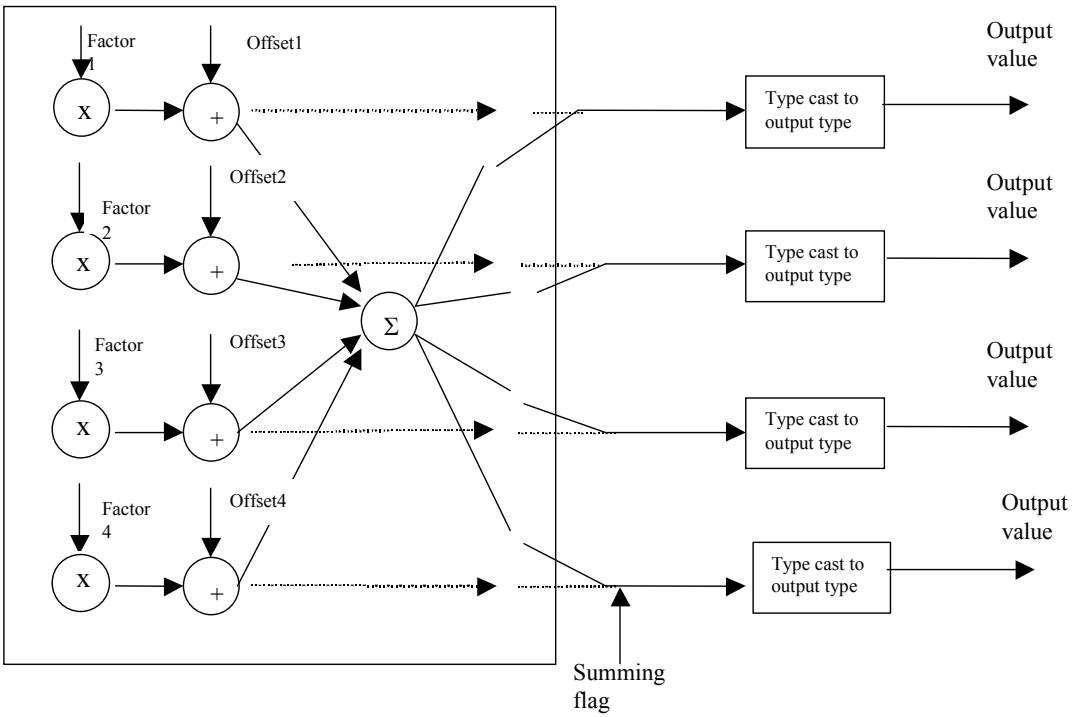


Figure 37 — Valuator functionaliy

Depending on the number of dimensions of the data type, there may be one up to four input values. For example an eventIn of type SFRotation will require four input paths but SFInt32 will only require the first input path. Each input path operates identically.

Each input value is converted to a floating-point value using a simple typecasting rule as illustrated in Table 35. After conversion, the values are multiplied by the corresponding factor.i value and added to the corresponding offset.i value as specified above. Depending on whether the summer is enabled, either the summed value or the individual values are presented at the output. The summer sums all 4 computed input paths independent of the number of dimensions of the eventIn type.

Table 35 — Simple typecasting conversion from other data types to float.

From	Conversion to float
Integer	Direct conversion. (1 to 1.0)
Boolean	true – 1.0 false – 0.0
Double	Truncate to 32-bit precision
String	Convert if the content of the string represents an int, float or double value. 'Boolean' string values "true" and "false" are converted to 1.0 and 0.0 respectively. Any other string is converted to 0.0

Table 36 — Simple typecasting conversion from float to other data types.

To	Conversion from float
Integer	Truncate floating point. eg (1.11 to 1)

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Boolean	0.0 to false Any other values to true
Double	Direct conversion
String	Convert to a string representing the float

For conversion of data types to and from strings the values of multiple valued data types, such as SFColor, are separated by spaces.

Depending on the dimension of the eventOut type, the corresponding number of output values are computed and converted to the output types according to Table 36 and as detailed below.

If the eventIn is of an SF type then an eventOut for an MF type shall consist of just one element, i.e., the MF type collapses to a SF type.

If the eventIn is of an MF type then an eventOut for an SF type shall be created by using the first element of the MF input only.

If the eventIn is of an MF type then an eventOut for an MF type shall be created by using each element of the MF input to generate one element of the MF output type, respecting the order of the elements in the eventIn MF type.

If the eventIn is of SFTime type then the conversion to string format shall be in the format “hh:mm:ss” where ‘hh’, ‘mm’, ‘ss’ are respectively hours, minutes and seconds of the input SFTime value.

EXAMPLE — The **Valuator** node can be seen as an event type adapter. One use of this node is the modification of the SFInt32 **whichChoice** field of a **Switch** node by an event. There is no interpolator or sensor node with a **SFInt32** eventOut. Thus, if a two-state button is described with a **Switch** containing the description of each state in choices 0 and 1. The triggering event of any type can be routed to a **Valuator** node whose **SFInt32** field is routed to the **whichChoice** field of the **Switch**.

SFVec4f fields cannot be routed to Valuator node.

7.2.2.136 Viewpoint

7.2.2.136.1 Node interface

```
Viewpoint {
   eventIn       SFBool      set_bind
   exposedField  SFFloat     fieldOfView     0.785398
   exposedField  SFBool      jump            TRUE
   exposedField  SFRotation  orientation     0, 0, 1, 0
   exposedField  SFVec3f     position        0, 0, 10
   field         SFString    description     ""
   eventOut      SFTime      bindTime
   eventOut      SFBool      isBound
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.136.2 Functionality and semantics

The semantics of the **Viewpoint** node are specified in ISO/IEC 14772-1:1998, subclause 6.53.

7.2.2.137 Viewport

7.2.2.137.1 Node interface

Viewport {

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

eventIn	SFBool	**set_bind**	
exposedField	SFVec2F	**position**	0 0
exposedField	SFVec2F	**size**	-1 -1
exposedField	SFFloat	**orientation**	0
exposedField	MFInt32	**alignment**	[0 0]
exposedField	SFInt32	**fit**	0
field	SFString	**description**	“”
eventOut	SFTime	**bindTime**	
eventOut	SFBool	**isBound**	

}

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.137.2 Functionality and semantics

A **Viewport** node can be placed in the **viewport** field of a **Layer2D** or **CompositeTexture2D** node or in the scene tree as a 2D node. It defines a new viewport and implicitly establishes a new local coordinate system. The bounds of the new viewport are defined by the **size** and **position** field. The new local coordinate system’s origin is at the center of the parent node in the parent’s local coordinate system.

The **orientation** field specifies the rotation which is applied to the viewport in the parent node’s local coordinate system with respect to the X-axis.

Viewport nodes are bindable nodes (see 7.1.1.2.14) and thus there exists a **Viewport** node stack which follows the same rules than other bindable nodes (e.g. **Background2D**).

The **description** field specifies a textual description of the **Viewport** node.

The **alignment** and **fit** fields specify how the viewing area is mapped to the rendering area of the parent node (i.e. **Layer2D**, **CompositeTexture2D**, or the 2D top-node).

If the **fit** field is set to 0, the viewing area is scaled to fit the rendering area without preserving the aspect ratio.

If the **fit** field is set to 1, the viewing area is scaled preserving the aspect ratio to fit entirely inside the rendering area. The scaling operation is performed possibly after rotation as specified by the **orientation** field.

If the **fit** field is set the 2, the viewing area is scaled preserving the aspect ratio to cover entirely the rendering area. The scaling operation is performed possibly after rotation as specified by the **orientation** field.

The **alignement** field is an MFInt32 field that contains two values. The first value specifies alignment along the X-axis and the second value specifies alignment along the Y-axis. The first value belongs to the following set of SFInt32: -1, 0, 1. The second value belongs to the following set of SFInt32: -1, 0, 1. An empty **alignement** field is equivalent to the default value. When the **fit** field is set to 0, the **alignment** field is ignored. The meaning of the different values of the **fit** and **alignment** fields is described in the following Figure.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

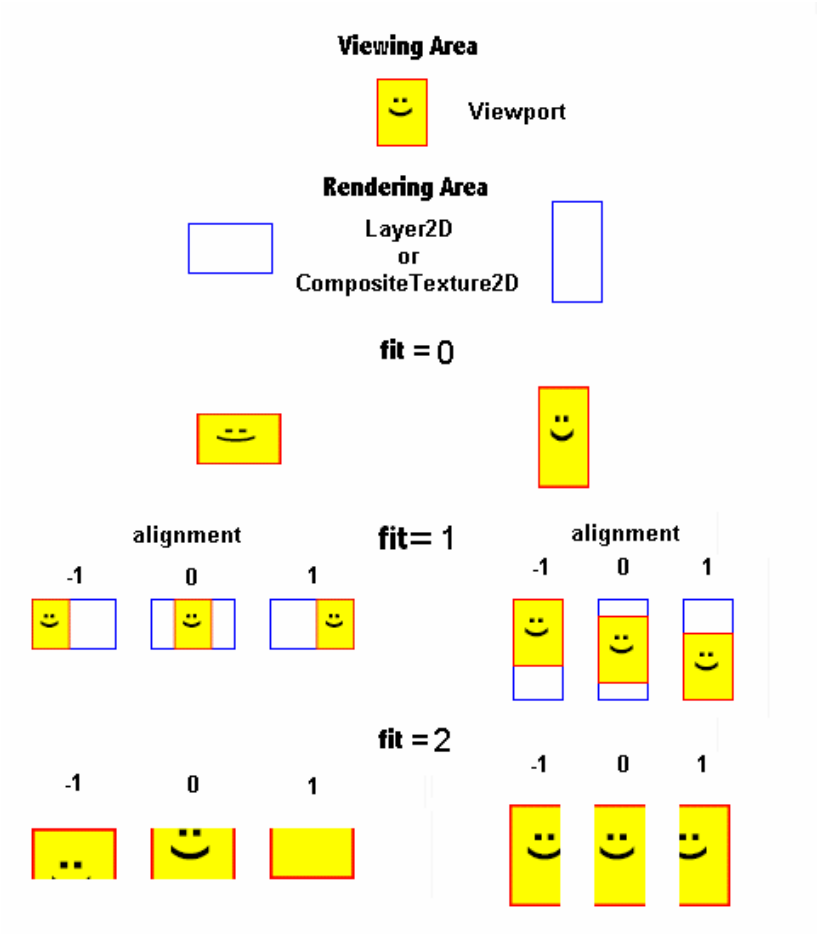


Figure 38 — description of alignment and fit fields

7.2.2.138 Viseme

7.2.2.138.1 Node interface

Viseme {

field	SFInt32	**viseme_select1**	0
field	SFInt32	**viseme_select2**	0
field	SFInt32	**viseme_blend**	0
field	SFBool	**viseme_def**	FALSE

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.138.2 Functionality and semantics

The **Viseme** node defines a blend of two visemes from a standard set of 14 visemes as defined in ISO/IEC 14496-2, Annex C, Table C-5.

The **viseme_select1** field specifies viseme 1.

The **viseme_select2** field specifies viseme 2.

The **viseme_blend** field specifies the blend of the two visemes.

If **viseme_def** is TRUE, the current FAPs shall be used to define a viseme and store it.

-`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.139 VisibilitySensor

7.2.2.139.1 Node interface

VisibilitySensor {			
exposedField	SFVec3f	**center**	0 0 0
exposedField	SFBool	**enabled**	TRUE
exposedField	SFVec3f	**size**	0 0 0
eventOut	SFTime	**enterTime**	
eventOut	SFTime	**exitTime**	
eventOut	SFBool	**isActive**	
}			

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.139.2 Functionality and semantics

The semantics of the **VisibilitySensor** node are specified in ISO/IEC 14772-1:1998, subclause 6.54.

7.2.2.140 WideSound

7.2.2.140.1 Node interface

WideSound {			
exposedField	SFNode	**source**	NULL
exposedField	SFFloat	**intensity**	1.0
exposedField	SFVec3f	**location**	0, 0, 0
exposedField	SFBool	**spatialize**	TRUE
exposedField	SFNode	**perceptualParameters**	NULL
exposedField	SFBool	**roomEffect**	FALSE
exposedField	SFInt32	**shape**	0
exposedField	MFFloat	**size**	[]
exposedField	SFVec3f	**direction**	0, 1, 0
exposedField	SFFloat	**density**	0.5
exposedField	SFInt32	**diffuseSelect**	1
exposedField	SFFloat	**decorrStrength**	1
field	SFFloat	**speedOfSound**	340
field	SFFloat	**distance**	1000
field	SFBool	**useAirabs**	FALSE
}			

NOTE — For binary encoding of this node see node coding tables in electronic attachment.

7.2.2.140.2 Functionality and semantics

The **WideSound** node is used to attach sound to a scene, thereby giving it spatial qualities with a determinable widening for not phase related signals from its descendant audio nodes and relating it to the visual content of the scene.

The **WideSound** node relates an audio BIFS subgraph 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 composed into the scene.

The semantics of the **source, intensity**, **location** and **spatialize** fields are described in subclause 7.2.2.116 **Sound.**

The semantics of the **speedOfSound**, **distance**, **useAirabs**, **perceptualParameters** and **roomEffect** fields are specified in subclause 7.2.2.46 **DirectiveSound.**

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Figure 39 shows a scene with two audio sources, a choir (or orchestra) and audience making applause. The choir consists of one **WideSound** node generating an ellipsoid and the audience consists out of three **WideSound** nodes generating boxes with decorrelated sounds positioned at three different locations.

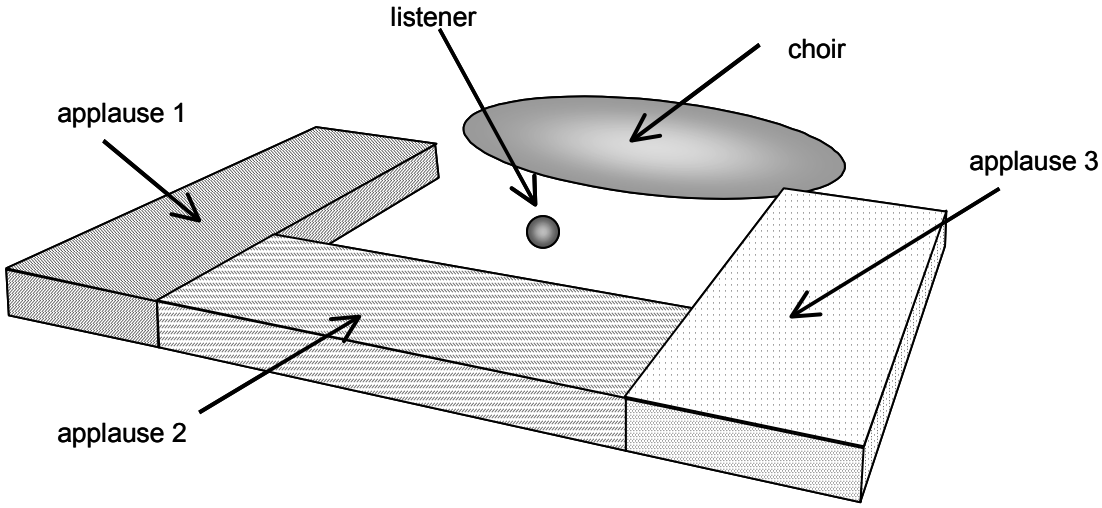


Figure 39 — Scene with 2 sources and 4 WideSound nodes

The single shape should consist of several equally distributed uncorrelated sound sources (see Figure 40).

The **density** distribution of the sound sources determines the subjective impression of the shape (sources/meter). To get different shapes from one source material, the sources have to be decorrelated independent from each other.

Different decorrelations can be selected with **diffuseSelect.** It should start with 1 and should be incremented for each new **WideSound** that is connected with the same source.

The field **decorrStrength** indicates the grade of decorrelation (measured by the cross-correlation function) each decorrelator should produce. A value of 0.0 signals full correlation.

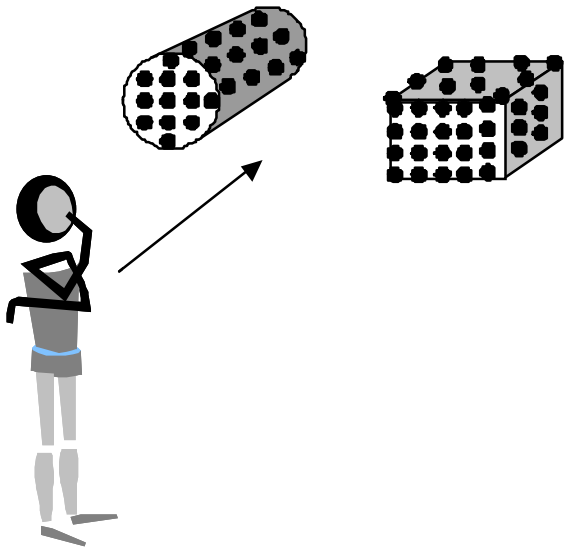


Figure 40 — Shapes consist of several uncorrelated sound sources

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In the case of the enabled **spatialize** field a geometric figure of the spatialized sound can be set with the **shape** field (see Table 37).

Table 37 — Selectable shapes and associated size and direction functionalities

Shape	**Description**	**Format of** size	**Range***	**Function of direction**
0	no wideness	--	--	--
1	shuck	width {horizontal, vertical}	{0…2π, 0…2π}	--
2	box	width {x, y, z} the orientation is normal to the x-z-plane (positive y-axis)	{0..inf, 0..inf, 0..inf}	object direction vector
3	ellipsoid	width {x, y, z} the orientation is normal to the x-z-plane (positive y-axis)	{0..inf, 0..inf, 0..inf}	object direction vector
4	cylinder	width {x, y, z} the cylinder is defined in the x-z-plane; the axis of the cylinder (orientation) is normal to the x-z-plane (positive y-axis)	{0..inf, 0..inf, 0..inf}	object direction vector (cylinder axis)
5..15	reserved for ISO use			
* The value –1 indicates an infinite value in the size field				

The *shuck* shape is a fixed format related to the default listener position. It describes a surrounding hull with a select-able opening angle at the **location**. The three shapes *box*, *ellipsoid* and *cylinder* can be seen in Figure 41.

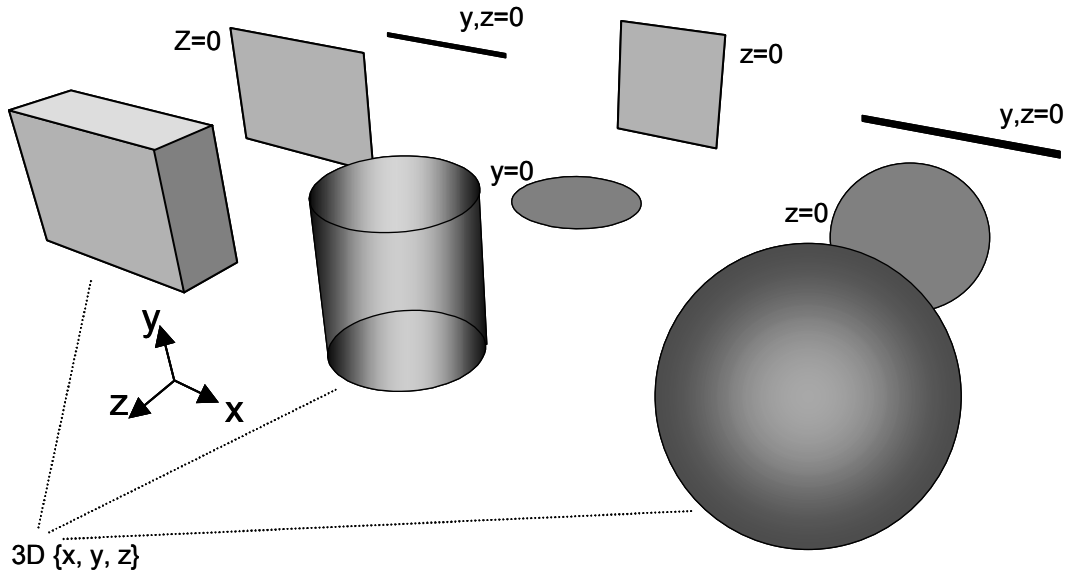


Figure 41 — Box, cylinder and ellipsoid (with equal widths == sphere) shapes

To signal plane waves the **decorrStrength** field can be set to 0, which means no decorrelation. Selecting the 'box' shape (**shape** = 2) and setting **size** to {-1, 0, 0} results in signaling a two-dimensional plane wave. Setting **size** to {-1, -1, 0} signals a three-dimensional plane wave.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.141 WorldInfo

7.2.2.141.1 Node interface

```
WorldInfo {
    field       MFString    info        []
    field       SFString    title       ""
}
```

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.141.2 Functionality and semantics

The semantics of the **WorldInfo** node are specified in ISO/IEC 14772-1:1998, subclause 6.55.

7.2.2.142 XCurve2D

7.2.2.142.1 Node interface

```
XCurve2D {
    exposedField    SFNode      point       NULL
    exposedField    SFFloat     fineness    0.5
    exposedField    MFInt32     type        []
}
```

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.142.2 Functionality and semantics

The **XCurve2D** node behave exactly as the **Curve2D** node except that is allows more values in the **type** field. The permitted values for the **type** field are:

0 = MoveTo: Same as the value 0 for the **type** field of the **Curve2D** node. In addition, the coordinate pair consumed from the **point** list also defines the starting point **P0** of a new subpath. MoveTo shall not occur as the first element in **type** field.

1 = LineTo: Same as the value 1 for the **type** field of the **Curve2D** node.

2 = CurveTo: Same as the value 2 for the **type** field of the **Curve2D** node.

3 = NextCurveTo: Same as the value 3 for the **type** field of the **Curve2D** node.

4 = CounterClockWiseArcTo: Three coordinate pairs in the **point** list are consumed, defining $\mathbf{F_1}$, $\mathbf{F_2}$ and **N**. $\mathbf{F_1}$ and $\mathbf{F_2}$ are the focal points of the ellipse to which **P** and **N** belong. On this ellipse, **P** and **N** define two arcs. Considering the polar parametric representation of the ellipse $(rx \cdot \cos(\theta), ry \cdot \sin(\theta))$ and assuming that $\mathbf{F_1}$ is the focal point with the negative coordinate on the x-axis, the drawn arc is the one that corresponds to an increase of θ when sweeping the arc from **P** to **N**. If the points **P**, $\mathbf{F_1}$, $\mathbf{F_2}$ and **N** do not belong to the same ellipse, then the arc is drawn using the quadruple of points **P**, $\mathbf{F_1}$', $\mathbf{F_2}$' and **N**, where $\mathbf{F_1}$' and $\mathbf{F_2}$' are scaled version of $\mathbf{F_1}$ and $\mathbf{F_2}$ with the middle of [$\mathbf{F_1F_2}$] being the middle of [$\mathbf{F_1'F_2'}$].

5 = ClockWiseArcTo: Same as CounterClockWiseArcTo except that the drawn arc is the one that corresponds to a decrease θ when sweeping the arc from **P** to **N**.

6 = ClosePath: No coordinate pair is consumed from the **point** list. The current subpath is closed by drawing a straight line from **P** to the current subpath's initial point $\mathbf{P_0}$. If a ClosePath is followed immediately by any other command than a MoveTo or RelativeMoveTo, then the next subpath starts at the same initial point as the current subpath, i.e. $\mathbf{P_0}$. Note: The difference between closing the subpath and explicitly drawing a line between **P** and $\mathbf{P_0}$ is that in the first case the line in $\mathbf{P_0}$ will be closed with the current value of line-join while in the second case the line will be closed using the current value of line-cap.

7 = QuadraticArcTo: Two coordinate pairs in the **point** list are consumed. The first coordinate pair constitutes the control point of a quadratic Bezier curve starting at the current point **P** and going to the point defined by the second consumed coordinate pair.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8 = Reserved

7.2.2.143 XFontStyle

7.2.2.143.1 Node Interface

XFontStyle {

exposedField	MFString	**fontName**	["SERIF"]
exposedField	SFBool	**horizontal**	TRUE
exposedField	MFString	**justify**	["BEGIN"]
exposedField	SFString	**language**	""
exposedField	SFBool	**leftToRight**	TRUE
exposedField	SFFloat	**size**	1.0
exposedField	SFFloat	**spacing**	1.0
exposedField	SFString	**stretch**	“NORMAL“
exposedField	SFFloat	**letterSpacing**	0.0
exposedField	SFFloat	**wordSpacing**	0.0
exposedField	SFInt32	**weight**	4
exposedField	SFBool	**fontKerning**	TRUE
exposedField	MFString	**style**	["PLAIN"]
exposedField	SFBool	**topToBottom**	TRUE
exposedField	MFString	**featureName**	[“”]
exposedField	MFInt32	**featureStartOffset**	0
exposedField	MFInt32	**featureLength**	0
exposedField	MFInt32	**featureValue**	0

}

NOTE - For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.143.2 Functionality and semantics

The fields of the **XFontStyle** node are defined as follows.

The **horizontal, justify, leftToRight** and **topToBottom** fields have the same meaning as in the **FontStyle** node.

The **fontName** field has the same semantic as the **family** field of the **FontStyle** node. 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.

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**.

The **language** field has the same meaning as in the **FontStyle** node. However, the format of the field is based on the RFC 3066, which supersedes RFC 1766.

The **stretch** field has an enumerated set of values that specify the font-stretch – desired amount of condensing or expansion of the glyphs in major and minor direction of the alignment **“NORMAL | ULTRA-CONDENSED | EXTRA-CONDENSED | CONDENSED | SEMI-CONDENSED | SEMI-EXPANDED | EXPANDED | EXTRA-EXPANDED | ULTRA-EXPANDED”**. The value of this field should be consistent with the value “usWidthClass” defined by a font. For description of the set of values please refer to OpenType Specification, version 1.4, chapter “Font File Tables”, section “Required Tables”, OS/2 table.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The **weight** field represents the boldness of a character. It has a range of values [100..900]. The value of this field should be consistent with the value "usWeightClass" defined by a font. For description of the set of values please refer to OpenType Specification, version 1.4, chapter "Font File Tables", section "Required Tables", OS/2 table.

The **style** field has an extended set of enumerated values specifying text attributes and decoration effects. The permitted values are **"PLAIN", "ITALIC", "BOLD", "BOLDITALIC", "UNDERLINE", "OUTLINE", "EMBOSS", "ENGRAVE", "LEFTDROPSHADOW", "RIGHTDROPSHADOW".** The font styles should be implemented according to the standard EIA-708-B "Digital Television (DTV) closed captioning", subclause 8.5.

The **fontKerning** field specifies whether font specific kerning data should be applied.

The **letterSpacing** field specifies, in the font metrics units, the additional space between letters after applying the font kerning.

The **wordSpacing** field specifies, in the font metrics units, the additional space between words after applying the **letterSpacing**.

The **featureName**, **featureStartOffset, featureLength** and **featureValue** fields are to be processed as a group. The four fields must have the same length to be valid. These parameters are a set that are to be applied to a run of text. The feature may apply to only one character in the run, or it may apply to the entire run. This is necessary to allow for correct typographic interaction during OpenType operations.

The **featureName** field specifies a list of four character registered tags that are assigned for a specific feature. Please refer to the registered features listed in the OpenType Specification, version 1.4, Appendix "OpenType Layout Common Table Format", section "OpenType Layout Registered Features".

The **featureStartOffset** field specifies the character offset into the run where the feature will be applied.

The **featureLength** field specifies the number of characters to which the feature is applied.

The **featureValue** field specifies the value that is applied for feature processing. A value of 0 is off (false), 1 is on (true). LookupType3 Alternate substitution may have values larger than 1. Please refer to OpenType Specification, version 1.4, chapter "Font File Tables", section "Advanced Typographic Tables", GSUB table.

EXAMPLE

```
Text {
    string "The red light is on"
    fontStyle XFontStyle {
        fontName ["OD:104;FSID:33"]
        size 24.0
        style "BOLD"
        featureSet ["smcp", "smcp"]
        featureStartOffset [4, 17]
        featureLength [3, 2]
        featureValue [1, 1]
    }
}
```

For the above example, the rendered text string:

The RED light is ON

7.2.2.144 XLineProperties

7.2.2.144.1 Node interface

XLineProperties {

exposedField	SFColor	**lineColor**	0, 0, 0
exposedField	SFInt32	**lineStyle**	0
exposedField	SFFloat	**width**	1.0
exposedField	SFBool	**isScalable**	TRUE
exposedField	SFInt32	**lineCap**	0
exposedField	SFInt32	**lineJoin**	0
exposedField	SFFloat	**miterlimit**	4
exposedField	SFFloat	**transparency**	0.0

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

exposedField	SFBool	**isCenterAligned**	TRUE
exposedField	SFFloat	**dash_offset**	0.0
exposedField	MFFloat	**dashes**	[]
exposedField	SFNode	**texture**	NULL
exposedField	SFNode	**textureTransform**	NULL

}

NOTE — For the binary encoding of this node see node coding tables in electronic attachment.

7.2.2.144.2 Functionality and semantics

The **XLineProperties** node specifies line parameters used in 2D rendering.

The semantics of **lineColor** and **lineStyle** are the same as for the **LineProperties** node. But, value 6 for the **lineStyle** field means that dashing uses the dash information of the **XLineProperties** node. For the other values of **lineStyle,** the values of the **dash_offset** and **dashes** fields are ignored.

The **dash_offset** field determines the position from the start of the outline, along the outline, in the local coordinate system, where dashing begins. In case of high level primitives such as Rectangle, Circle and Ellipse, the start of the outline is not specified and therefore the use of **dash_offset** is undetermined. For a deterministic behavior, authors should use primitives that explicitly define the start of the outline.

The **dashes** array specifies the dash pattern. Each element is a strictly positive number expressed relatively to the width of the pen. Values at even positions in the array specify the length of drawn parts of the line. Values at odd positions in the array specify the length of non-drawn parts of the line. Dashes shall be drawn using the **lineCap** information.

The **width** field determines the width, in the local coordinate system, of rendered lines. The width is subject to scaling only when the **isScalable** field is set. When **isScalable** is TRUE, the outline path of the shape is computed in the local coordinate system with the given width, then the world transform for the object is applied. Otherwise the outline path is of the object is directly computed in the world coordinate system with the given width.

The **lineCap** field specifies the line cap style type to apply to lines. The allowed values are:

Table 38 — lineCap description

lineCap	Description
0	flat
1	round
2	square
3	triangle
4-7	reserved



The **lineJoin** field specifies the line join style type to apply to lines. The allowed values are:

Table 39 — lineJoin description

lineJoin	Description
0	miter
1	round
2	bevel
3	reserved

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



The **miterlimit** field specifies the limit on the ratio of the miter length to the line width. The value of **miterlimit** must be a number greater than or equal to 1.

The **transparency** field specifies the transparency of the outline of a shape when drawn. It supersedes the value of the transparency of a **material** node.

The **isCenterAligned** field specifies the positioning of the outline a shape. If TRUE, the line that represents the outline of the shape is drawn centered on the outline of the shape. If FALSE, the outside edge of the line that represents the outline of the shape is aligned on the outside edge of the shape.

The **texture** field, if specified, shall contain one of the various types of texture nodes. If NULL or unspecified, the line is not textured. Texture mapping coordinates are defined by the four corners of the bounding rectangle of the outer polygon defining the shape (that is, taking into account the width of the line).

The **textureTransform** field, if specified, shall contain a texture transformation node (**TextureTransform** or **TransformMatrix2D** without children). If the **textureTransform** is NULL or unspecified, the texture is not transformed.

7.3 Informative: Differences Between MPEG-4 Scripts and ECMA Scripts

7.3.1 MPEG-4 Scripts Have a Rigid Representation

MPEG-4 scripts differ slightly from ECMA scripts. The most important difference is that MPEG-4 scripts are not represented textually, but are transmitted as a parse tree representation. This means that only constructs that can be represented by the MPEG-4 parse grammar can be encoded and transmitted. Not all ECMA script constructs can be represented in MPEG-4 scripts.

The differences between ECMA scripts and scripts that can be represented in MPEG-4 are given below.

7.3.2 Keywords

MPEG-4 scripts cannot utilize the following keywords: `catch delete do finally in instanceof throw try typeof void with .`

This means that `do - while` loops and `for - in` loops are not possible.

7.3.3 Relational operators

The relational operators "`===`" and "`!==`" cannot be included in MPEG-4 Scripts.

7.3.4 Labeled statements

In MPEG-4 scripts it is impossible to label statements and to `break` or `continue` to labeled statements.

7.3.5 Switch statement restriction

MPEG-4 scripts with `switch` statements can only take numerical `case` expressions and always must have at least one `case` statement.

In particular this means that

```
switch {
  case (x+1):  ….
}
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

is not possible, while

```
Switch {
  case 1:  ….
}
```

is okay.

7.3.6 Functions, not programs

The MPEG-4 event driven script model allows only functions to be called in response to events.

7.3.7 Expressions

Statements that include statement blocks, such as for are represented in the parse tree as having an empty statement block, where as in ECMA script they can omit this block. Functionally, the statements behave identically. For example, the expression:

```
for ( <expr>; <expr>; <expr>)
```

must be represented as

```
for ( <expr>; <expr>; <expr>) {}
```

7.3.8 Array and Object Literals

Array and object literals of the form `[value1, value2, .., valueN]` and `{property1:value1, property2: value2, .. propertyN:valueN}` cannot be used in MPEG-4 scripts.

7.4 Informative: FlexTime behavior

7.4.1 FlexTime behavior

At a very basic level, FlexTime allows nodes and a media stream to be synchronized, such that the nodes and the texture (coming from a media stream) can be rendered together. To do this, we transform the scene for the nodes to co-start with the media stream. We achieve this effect by inserting the nodes under a **TemporalTransform**. The removal of a node can also be synchronized with the end of the stream in this manner.

When the FlexTime manager has no corrections to make, the playback speed of continuous media is set to meet the optimal duration and hence this speed is expected to be the normal standard playback speed. The timing relationships of the temporal groups allow flexibility of duration. Alteration of duration can be done through adjustment of playing speed to allow these timing relationships to be met. Without playback speed modification only truncate and freeze are possible. Alterations, when permitted, are specified in the **TemporalTransform** with stretch and shrink preferences. Playback speed is not expressed explicitly in the **TemporalTransform** at all - rather the speed can be adjusted to meet timing needs and as such playback speed is a secondary effect not the prime motivator behind FlexTime. The prime motivator being synchronization and timing adjustments using application-level (author) specified guidelines.

7.4.2 Updates of fields

In order to support synchronization of field value replacements with a temporally transformed scene, an appropriate construct other than directly replacing the field with a BIFS command may need to be chosen. Such a construct should be temporally transformed commensurate with the scene that it is to update. For example a time sensor/valuator/route combination can be used, or a BIFS command frame containing the replace field can be put into a conditional node that is activated at the desired (temporally transformed) time.

BIFS-Anim streams, being media streams, can be re-timed as is done for other media streams.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.4.3 Authoring considerations

7.4.3.1 Plan the possible different playback scenarios

Authors should take special care so that the MPEG-4 usual constraints are always respected. However, authors should avoid possible ODid overlapping and Nodeid overlapping as streams are flexed for instance. Furthermore, it is possible to reuse ODid, but the author should ensure that streams that designate 2 different ODs, using the same ODid, should not occur together anywhere under a **TemporalGroup**.

7.4.3.2 Flexed Objects

The objects that are affected by the FlexTime model, called "flexed" objects, may be streams, such as a movie or audio clips, part of streams, or graphic animations, such as a progress bar, a revolving globe, an opening door etc. Such animations can be realized in MPEG via animation streams and BIFS update, but also by through interaction between scene nodes, Routes and **TimeSensor** nodes.

7.4.3.3 Why a children field?

The case when a "flexed" object is a stream is well covered with the **url** field in **TemporalTransform**. However, we need to support "flexed" objects that are made of group of scene nodes and interaction between them. For instance consider the case of two movies that are played one after the other, with a "credits" screen shown in between. The credits screen shows a scroll of textual information that needs to start after the first movie ends and complete when the next movie starts. However, the end and start times of the two movies are not fully deterministic and might shift in time. In this case we have three "flexed" objects – the two movies and the credit screen in between. The *FlexTime* model enables the player to apply temporal transformation on the objects so that they do play in perfect sequence. Therefore the speed of the text scrolling in the credit screen may be changed by the model.

With the current proposal, this scenario would be implemented very easily. The two movies will be define in the **url** fields of two **TemporalTransform** nodes, while the entire set of scene nodes that consist the credit screen would be referenced by the **children** field of a third **TemporalTransform** node. The **TemporalGroup** node would then define a *meet* constraint on these three objects.

It might be claimed that the same effect can be achieved without a **children** field, by defining only the two movies as "flexed" objects, and use the events generated when the two objects start and stop, routed into **Conditional** nodes, interpolators, evaluators and scripts. In theory this may work, but would be extremely difficult to author and there is no proof it will work in all cases.

7.5 Informative: Implementation of MaterialKey node

An example implementation is presented below to reveal the intended use of the color key information. To calculate the transparency (alpha) value for each pixel, first the distance d between the unnormalized key color (C_1, C_2, C_3) and the color (X_1, X_2, X_3) of the pixel of interest is calculated.

If the magnitude of the variance of the pixel falls between the 2 thresholds, the alpha value for that pixel will be scaled between completely transparent, and the transparency value given for the opaque region. The following describes how the alpha value for a given pixel is determined.

c_1 – The normalized value of the R (or Y) component of the keycolor (in range 0.0 to 1.0)

c_2 – The normalized value of the G (or U) component of the keycolor (in range 0.0 to 1.0)

c_3 – The normalized value of the B (or V) component of the keycolor (in range 0.0 to 1.0)

The respective unnormalized values of c_1, c_2, c_3 are C_1, C_2, C_3 and are obtained by multiplying c_1, c_2, c_3 by $k = 2^n - 1$, which for n=8 bit video is 255; this computation is performed only once at the time of selection of a new keycolor and the results are stored. Also, by scaling k, a factor $K=3\times k$ can be precomputed and stored; this needs to be done just once.

X_1 – The value of the R (orY) component (in the range 0 to k) of the pixel for which the alpha value is to be computed

X_2 – The value of the G (or U) component (in the range 0 to k) of the pixel for which the alpha value is to be computed

X_3 – The value of the B (or V) component (in the range 0 to k) of the pixel for which the alpha value is to be computed

T – Transparency value assigned to the opaque region (in range 0.0 to 1.0)

d_1 – Low threshold for transparency detection (in range 0.0 to T)

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

d_2 – High threshold for transparency detection (in range 0.0 to T)

$d = (| C_1 - X_1| + | C_2 - X_2| + | C_3 - X_3 |) * T/K$

The resulting normalized value of distortion d lies in the range of 0.0 to T.

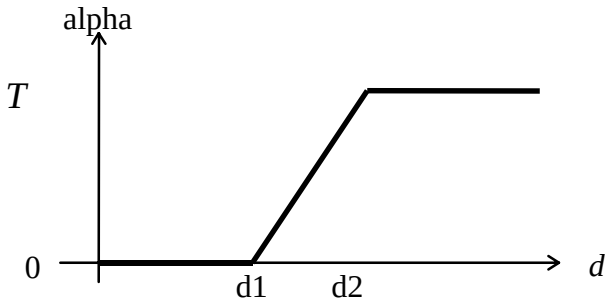


Figure 42 — Alpha value as a function of distance measure

The reconstructed alpha value for each pixel is computed by comparing the distance d with the thresholds as follows:

if ($d \leq d_1$) then *alpha* = 0,

else if ($d > d_2$) then *alpha* = T,

else if ($d_1 < d \leq d_2$) then $alpha = (d-d_1)/(d_2-d_1) * T$

Here, *alpha* = 0 is transparent and *alpha* = T is the transparency value assigned to the opaque region.

Further, $d_1 = d_2$ implies binary shape, otherwise grey scale shape is obtained.

7.6 Informative: Example implementation of spatial audio processing (perceptual approach)

7.6.1 Example algorithm implementation

This section describes a rendering algorithm which can be controlled by the proposed perceptual parameters. It receives as inputs an audio source url, the nine perceptual parameters, the position of the source with respect to the view point, the directivity diagram of the source, and the **directFilter** and **inputFilter** parameters. It produces seven channels, one for the direct sound C, two for the early reflections L and R, and four for the diffuse field S1, S2, S3 and S4. Ideally these are to be played back according to the diagram shown below. The four main parts of the *Room* module are detailed below in the case of an 8-channel implementation of the complete model (including the *early* and *cluster* blocks).

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

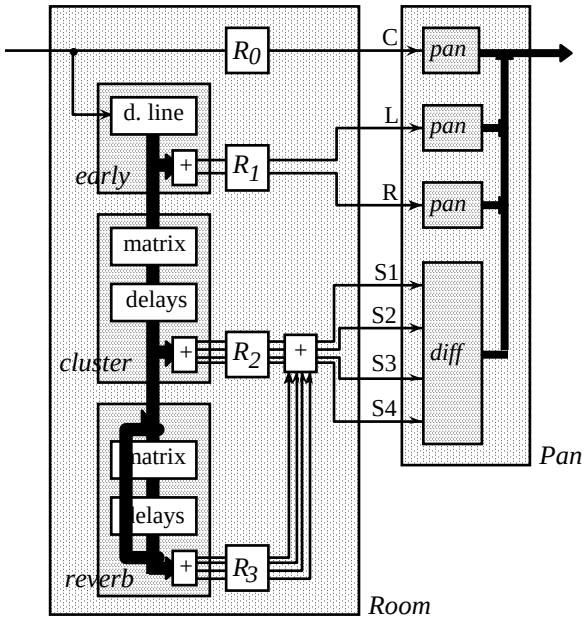


Figure 43 — Association of *Room* and *Pan* modules forming a *Spat* processor.

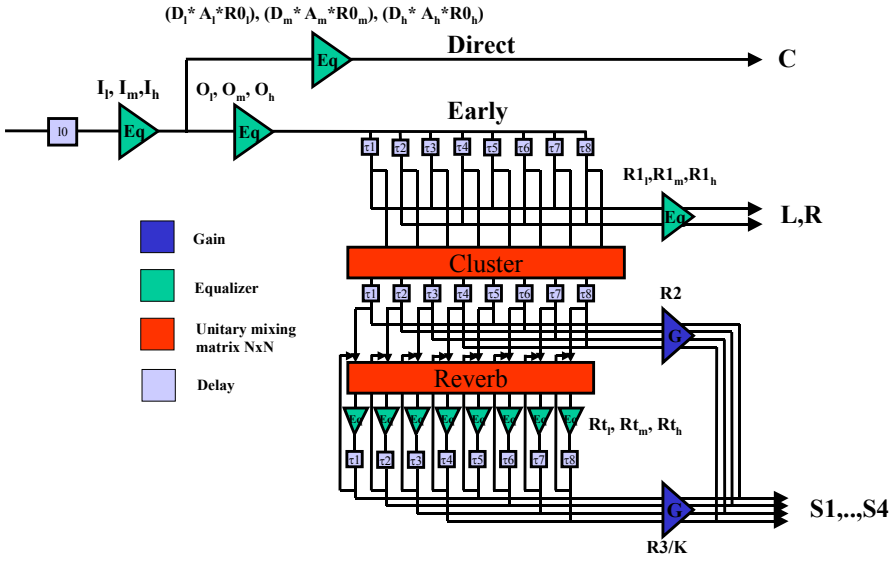
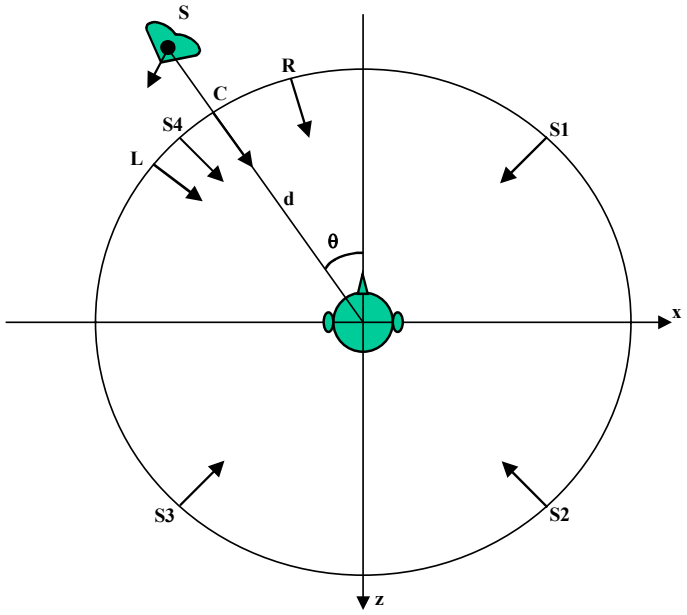


Figure 44 — Block diagram of the *Room* module

Copyright International Organization for Standardization © ISO/IEC 2005 All rights reserved
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



Figrue 45 — Directional rendering by the Pan module

7.6.2 Elementary spectral corrector

An elementary component used in multiple instances in the *Room* module is the second order IIR filter whose equation is given by :

$y(n)=b_o x(n)+b_1 x(n-1)+b_2 x(n-2)-a_1 y(n-1)-a_2 y(n-2)$.

This filter is used as a 3-band parametric equalizer, the characteristics of which are given by:

f_{low} : higher crossover frequency expressed in Hz

f_{high} : lower crossover frequency expressed in Hz

g_{low} : filter gain in the low band expressed w.r.t amplitude

g_{mid} : filter gain in the mid band expressed w.r.t amplitude

g_{high} : filter gain in the high band expressed w.r.t amplitude

The method to calculate the 2nd order cell coefficients is given by :

a)

$f^r_{low} = f_{low}/f_s$

$f^r_{high} = f_{high}/f_s$

where f_s is the sampling rate

b)

$g^r_{low} = g^r_{low}/g$

$g^r_{mid} = g^r_{mid}/g$

$g^r_{high} = g^r_{high}/g$

where g is a gain factor that prevent the filter coefficients from being inaccurate for very low values of g_{low}, g_{mid} and g_{high} and so degrading the filtering process. Generally, gain is taken to be equal to g_{mid} .

c)

$k_1 = g^r_{low} / g^r_{mid}$

$r_1 = \tan(\pi * f^r_{low})/k_1^{0.5}$

$\alpha_1 = (r_1 - 1)/(r_1 + 1)$

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$\beta_1 = (k_1 * r_1 - 1)/(k_1 * r_1 + 1)$

$\chi_1 = (k_1 * r_1 - 1)/(r_1 + 1)$

d)

$k_2 = g^{r}_{mid} / g^{r}_{high}$

$r_2 = \tan(\pi * f^{r}_{high})/k_2^{0.5}$

$\alpha_2 = (r_2 - 1)/(r_2 + 1)$

$\beta_2 = (k_2 * r_2 - 1)/(k_2 * r_2 + 1)$

$\chi_2 = (k_2 * r_2 - 1)/(r_2 + 1)$

e)

$k = g^{r}_{mid} * g$

$b_o = \chi_1 * \chi_2 * k$

$b_1 = (\beta_1 + \beta_2) * b_o$

$b_2 = (\beta_1 * \beta_2) * b_o$

$a_1 = \alpha_1 + \alpha_2$

$a_2 = \alpha_1 * \alpha_2$

7.6.3 Input Filter

In order to simulate sound sources that are outside the virtual room, a pre-filtering process can be performed with the values given in the ***inpultFilter*** field, I_{low}, I_{mid}, I_{high}, via a three-band equalization.

7.6.4 Direct path

The signal which stands for the direct path (without any reflection) is calculated via a 3-band equalizer from the input signal S. The coefficients of this equalizer are calculated according to the avant-propos with the following energetic parameters $A_{low} * D_{low} * R0_{low}$, $A_{mid} * D_{mid} * R0_{mid}$, $A_{high} * D_{high} * R0_{high}$, f_{min} and f_{max}, where the A's, D's and R0's represents respectively the axis directivity, the direct filter coefficients and the energetic repartition of the source in the three bands [0, f_{min}],[f_{min}, f_{max}] and [f_{max},f_s/2]

f_{min} and f_{max} are given in the **PerceptualParameters** node fields.

7.6.5 Directional early reflections

The source signal S is first filtered in an equalizer which depends only on the diffuse-field frequency response of the source, i.e. the omnidirectivity of the source. Its parameters are O_{low},O_{mid},O_{high}, f_{min} and f_{max} where the O's give the diffuse-field amplitude of the source in the three bands [0, f_{min}],[f_{min}, f_{max}] and [f_{max},f_s/2]

The output of this equalizer feeds a delay line that produces eight channels, early[i], i=0,..7, which are time shifted.

The eight delay lengths are randomly distributed between approximately 20 and 40 ms for a large room, but could be set differently in order to simulate a smaller room. The early signals are multiplied by gains, gi, and combined to produce two signals, Lo and Ro as follows :

$Lo = early[0] * g_0 + early[2] * g_2 + early[4] * g_4 + early[6] * g_8$

$Ro = early[1] * g_1 + early[3] * g_3 + early[5] * g_5 + early[7] * g_7$

NOTE - In the proposed implementation $g_i = 1.0$, i=0,...,7 .

Lo and Ro are then filtered with two equalizers having the following parameters $R1_{low}$, $R1_{mid}$, $R1_{high}$, f_{min} and $f_{max,}$ in order to produce L and R that represent the early reflections.

Diffuse early reflections

The eight outputs of the *Early* delay line are mixed in a unitary Hadamard (8x8) matrix to produce eight scrambled signals :

$[Cscramb_i]_{i=0,\ldots,7} = H_{8x8}[early_i]_{,i=0,\ldots,7}$

The scrambled signals are independently delayed :

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Cscramb ^{d}[i](t)=Cscramb[i](t-τ_i), i=0,...,7

The values of the τ_i are randomly distributed between 20 and 60 ms approximately (default setting for a large room). The exact values to be used are given in 1.2.6.6.6.

The Cscramb^{d}[i] are combined to produce four intermediate signals that will feed the cluster equalizers R2 :

$Ctemp_1$= Cscramb^{d}[0]+ Cscramb^{d}[4]

$Ctemp_2$= Cscramb^{d}[1]+ Cscramb^{d}[5]

$Ctemp_3$= Cscramb^{d}[2]+ Cscramb^{d}[6]

$Ctemp_4$= Cscramb^{d}[3]+ Cscramb^{d}[7]

The four signals $Ctemp_{i\ i=0,...,3,}$ are then filtered with four equalizers that have the following parameters $R2_{low}$,$R2_{mid}$,$R2_{high}$, f_{min} and f_{max}, in order to produce the diffuse field signals corresponding to the cluster part of the impulse response: $R2_{0,}$ $R2_{1,}$ $R2_2$ and $R2_3$

7.6.6 Diffuse late reverberation

This stage is quite similar to the previous one except that in order to reproduce the late reverberation decay, a feedback delay network (FDN) is used.

The eight input signals are mixed in a unitary Hadamard (8x8) matrix producing eight scrambled signals :

$[Rscramb_i]_{i=0,...,7}=H_{8x8}[Cscramb^{d}_i+Rscramb^{d}_i]_{,i=0,...,7}$

The scrambled signals are independently delayed:

Rscramb ^{d}[i](t)=Rscramb[i](t-τ_i), i=0,...,7

The values of the τ_i are randomly distributed between 60 and 140 ms approximately (for a large room).

Then these signals are filtered with 8 equalizers that have the following parameters :

p_{lowi},p_{midi},p_{highi},f_{min},f_{max}

where

$p_{lowi}=10^{(-60*\ \tau i/Rtlow)}$

$p_{midi}=10^{(-60*\ \tau i/Rtmid)}$

$p_{highi}=10^{(-60*\ \tau i/Rthigh)}$

to produce the Rscramb^{eq}[i] signals

The Rscramb^{eq}[i] are combined to produce four intermediate signals :

$Rtemp_1$= Rscramb^{eq} [0]+ Rscramb^{eq} [4]

$Rtemp_2$= Rscramb^{eq} [1]+ Rscramb^{eq} [5]

$Rtemp_3$= Rscramb^{eq} [2]+ Rscramb^{eq} [6]

$Rtemp_4$= Rscramb^{eq} [3]+ Rscramb^{eq} [7]

The four signals $Rtemp_{i\ i=0,...,3,}$ are then scaled by a gain R3.

The FDN (FeedBack Delay Network) used for the late reverberation introduces a gain K that have to be compensated for in order to achieve the expected energy R3.

This gain can be estimated from the absorptive gains $ki=10^{(-60*\tau i\ /Rtmid)/10}$ by the following formulae :

$$k = \frac{1}{N}\sum_{i=0}^{N-1} ki,$$

$$K = \sqrt{\frac{k}{k-1}}$$

The complete diffuse field is simply calculated from the cluster and the late reverberation fields signals as follows :

$S1= R2_0 + R3_0$

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$S2 = R2_1 + R3_1$

$S3 = R2_2 + R3_2$

$S4 = R2_3 + R3_3$

7.6.7 Setting the delays

In the Perceptual node field, four values related to the temporal characteristics of the impulse response are given: the time limits l_1, l_2, l_3 and the modal density.

The Room structure has three sets of delays, respectively for the *Early*, *Cluster* and *Reverb* modules.

The delay ranges to be used can be calculated as follows :

Table 40 — delay ranges

	min	max
early delays	l1	l2
cluster delays	l2−l1	L3−l2+e
reverb delays	l3−l2−e	(*)

e can be used to create some overlapping between the temporal sections R_2 and R_3 if necessary.

(*): in the *Reverb* module, the distribution of delay lengths is not constrained by their maximum, but by their sum, which is equal to the value *modal density* (expressed indifferently in seconds or modes per hertz).

7.6.8 Scalability

The above modular signal processing model provides several forms of scalability:

- modifying the number of discrete reflections in the *early* block
- modifying the number of delay channels in the *cluster* and *reverb* blocks (typically 4, 6 or 8 channels)
- suppressing the *cluster* block, and possibly the *early* block
- sharing the *reverb* block and possibly the *cluster* block between several sources (located in the same room)
- replacing the equalizers by simple gains. In that case the frequency effects will not be rendered, e.g room liveness and source warmth.

7.7 Informative: MPEG-4 Audio TTS application with Facial Animation

To clarify the basic architecture and operations of an MPEG-4 terminal when the MPEG-4 Audio Text-to-Speech Decoder is used with Facial Animation, application specific interpretations of the bitstream syntax and semantics of MPEG-4 Systems and MPEG-4 Audio are addressed here.

As this application has two different outputs including synthesized speech and animated face decoders, the TTS synthesizer and the face decoder should be incorporated. In addition to these decoders, a special component "Phoneme/bookmark-to-FAP converter" is used to animate the face synchronously with synthesized phonemes. As the TTS stream drives the face decoder, the Phoneme/bookmark-to-FAP converter generates FAPs with appropriate timing information. The speech synthesizer feeds phonemes and their duration to the Phoneme/bookmark-to-FAP converter. The MPEG-4 terminal is configured to associate a **Sound** node and a **Face** node through the **TTSsource** field of the **Face** node which may contain the **AudioSource** node of the TTS.

If the MPEG-4 terminal receives a **Face** node with a non-NULL **TTSsource** field, it connects the **Face** node to the **AudioSource** node as defined in this **TTSsource** field. The **AudioSource** node contains the MPEG-4 Audio Text-to-Speech. The MPEG-4 Audio Text-to-Speech Decoder communicates with the **Face** node using the ttsFAPInterface of the Phoneme/bookmark-to-FAP converter.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.8 Informative: 3D Mesh Coding in BIFS scenes

7.8.1 Vertex Ordering

The **IndexedFaceSet** (IFS) node in BIFS can be compressed using 3D Mesh Coding (3DMC). While the decompressed 3DMC bitstream produces the results for IFS node, the order of vertices in original IFS node may be changed due to the fact that Topological Surgery, a key element in 3DMC, may alter the order of vertices in the encoding process. This means that different encoders may yield different vertex orders for the same IFS node. However, given the 3D Mesh bitstream, the order of vertices in the decoded 3D mesh is unique.

This may not be a problem for the BIFS Scenes where the order of vertices has no impact. For those applications where the order of the vertices is critical, the order of vertices after the encoding process can be used instead of the original order of the vertices.

For example, when 3DMC is used for IFS node, the index of the coord field may be changed. If there is a node that relates to **coord** field, such as **CoordinateInterpolator** node, the changed index of **coord** should be applied.

7.8.2 Using separate streams

In order to use 3DMC for the **IndexedFaceSet** node, the use3DMeshCoding flag in BIFSv2Config should be set to TRUE, as described in subclause 8.5.3. This will cause every **IndexedFaceSet** node in that stream to be coded with 3DMC. However, if the user wants some **IndexedFaceSet** nodes within a scene compressed with 3DMC and have some uncompressed without 3DMC, they must be sent as separate elementary streams. In order to keep them in a single scene and have them in the same name scope, these elementary streams should be defined in the same (Initial) ObjectDescriptor. Each stream will be of type SceneDescriptionStream but will have the use3DMeshCoding flag set true or false, as required, in the BIFSv2Config carried within the DecoderSpecificInfo inside the DecoderConfigDescriptor for each stream.

7.9 Profiles

7.9.1 Introduction

This subclause defines profiles and levels for the usage of the tools defined in this part of ISO/IEC 14496. Each profile at a given level constitutes a subset of this part of ISO/IEC 14496 to which system manufacturers and content creators can claim conformance in order to ensure interoperability.

The scene graph profiles specify the allowed scene graph elements of the BIFS tool. The graphics profiles specify the graphics elements of the BIFS tool that are allowed. The MPEG-J profiles specify the packages of the MPEG-J API specification that are allowed in an MPEG-J terminal.

Profile definitions, by themselves, are not sufficient to provide a full characterization of a receiving terminal's capabilities and the resources needed for a presentation. For this reason, levels are defined within each profile. Levels constrain the values of parameters in a given profile in order to specify an upper complexity bound.

7.9.2 Scene Graph Profile Definitions

7.9.2.1 Overview

The scene graph profiles specify the scene graph elements of the BIFS tool that are allowed. These elements provide the means to describe the spatio-temporal locations, the hierarchical dependencies as well as the behaviors of audio-visual objects in a scene. Profiling of scene graph elements of the BIFS tool serves to restrict the memory requirements and computational complexities of scene graph traversal and processing of specified behaviors during the composition and rendering processes.

7.9.2.2 Scene Graph Profiles Tools

The following tools are available to construct the definitions for scene graph profiles:

BIFS nodes related to scene description as defined in Table 41.

BIFS commands and BIFS animation as defined in 8.6 and 8.8 respectively.

BIFS ROUTES as defined in 8.7.56.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3D audio scene graph profile as defined in 7.9.2.3.3.

7.9.2.3 Scene Graph Profiles

The following table defines the scene graph profiles:

Table 41 — Scene graph profiles

	Scene Graph Profiles								
Scene Graph Tools	**Basic 2D**	**Simple 2D**	**Core 2D**	**Main 2D**	**Advanced 2D**	**Complete 2D**	**Audio**	**3D Audio**	**Complete**
AcousticScene								X	
AnimationStream					X	X		X	X
Anchor			X	X	X	X		X	X
AudioBuffer					X	X	X	X	X
AudioDelay						X	X	X	X
AudioFX						X	X	X	X
AudioMix						X	X	X	X
AudioSwitch					X	X	X	X	X
Billboard								X	X
ColorInterpolator			X	X	X	X			X
Collision									X
CompositeTexture2D						X			X
CompositeTexture3D									X
Conditional			X	X	X	X		X	X
CoordinateInterpolator2D			X	X	X	X			X
CoordinateInterpolator								X	X
CylinderSensor									X
DirectiveSound								X	
DiscSensor				X	X	X			X
Form						X			X
Group					X	X	X	X	X
Inline			X	X	X	X		X	X
InputSensor			X	X	X				
Layer2D				X	X	X			X
Layer3D									X
Layout						X			X
ListeningPoint						X	X	X	X
LOD								X	X
MediaBuffer					X				
MediaControl			X	X	X				
MediaSensor			X	X	X				
NormalInterpolator									X
NavigationInfo									X
OrderedGroup	X	X	X	X	X	X			X
OrientationInterpolator								X	X
PerceptualParameters								X	
PlaneSensor2D				X	X	X			X
PlaneSensor									X

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

PositionInterpolator								X	X
PositionInterpolator2D			X	X	X	X			X
ProximitySensor								X	X
ProximitySensor2D				X	X	X			X
QuantizationParameter			X	?	X	X		X	X
ScalarInterpolator			X	X	X	X			X
Script					X			X	X
Sound							X	X	X
Sound2D	X	X	X	X	X	X		X	X
ServerCommand			X	X	X				
SphereSensor									X
Switch			X	X	X	X		X	X
TemporalTransform				X	X				
TemporalGroup				X	X				
TermCap					X	X		X	X
TimeSensor			X	X	X	X		X	X
TouchSensor			X	X	X	X		X	X
Transform								X	X
Transform2D		X	X	X	X	X			X
Valuator			X	X	X	X		X	X
Viewpoint								X	X
VisibilitySensor								X	X
WorldInfo					X	X		X	X
Node Update			X	X	X	X		X	X
Route Update			X	X	X	X		X	X
Scene Update		X	X	X	X	X	X	X	X
ROUTE			X	X	X	X		X	X
PROTO					X				
PredictiveMF coding					X				

Decoders that claim compliance to a given profile shall implement all the tools with an ‘X’ entry for that profile.

7.9.2.3.1 BIFS nodes for audio objects

The presence of **AudioClip** and **AudioSource** nodes in BIFS scene graph depends on the selected Audio profile. Note, however, that certain Scene Graph profiles place limitations on the complexity of these nodes. The following table describes what nodes are allowed in the BIFS scene graph depending on the Audio profile. Note that Systems profiles supporting the **AudioFX** node in decoders where the Audio Main or Synthetic Profiles are not supported, shall at least be compliant with Audio Object Type 16 “Algorithmic Synthesis and AudioFX,” as defined in ISO/IEC 14496-3 Subpart 1, in order to guarantee the correct functionality of the Audio subtree.

Table 42 — BIFS nodes for audio objects

Audio Profiles	Allowed Audio Object Nodes
Main	AudioClip, AudioSource
Scalable	AudioClip, AudioSource
Speech	AudioClip, AudioSource
Synthesis	AudioClip, AudioSource
High Quality Audio	AudioClip, AudioSource
Low Delay Audio	AudioClip, AudioSource
Natural Audio	AudioClip, AudioSource
Mobile Audio	AudioClip, AudioSource

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Internetworking	

7.9.2.3.2 BIFS nodes for visual objects

The presence of ImageTexture, Background2D, Background, MovieTexture, Face, Expression, FAP, FDP, FIT, FaceDefMesh, FaceDefTable, FaceDefTransform, Viseme nodes in a BIFS scene graph depends on the selected Visual profile. The following table describes what nodes are allowed in the BIFS scene graph depending on the choice of the Visual profile.

Table 43 — BIFS nodes for visual objects

Visual Profiles	Allowed visual object nodes
Simple	ImageTexture, Background2D, Background, MovieTexture
Simple Scalable	ImageTexture, Background2D, Background, MovieTexture
Core	ImageTexture, Background2D, Background, MovieTexture
Main	ImageTexture, Background2D, Background, MovieTexture
N-Bit	ImageTexture, Background2D, Background, MovieTexture
Advanced Real Time Simple	ImageTexture, Background2D, Background, MovieTexture
Core Scalable	ImageTexture, Background2D, Background, MovieTexture
Advanced Coding Efficiency	ImageTexture, Background2D, Background, MovieTexture
Advance Core Profile	ImageTexture, Background2D, Background, MovieTexture
Hybrid	ImageTexture, Background2D, Background, MovieTexture, Face, Expression, FAP, FDP, FIT, FaceDefMesh, FaceDefTable, FaceDefTransform, Viseme
Basic Animated Texture	ImageTexture, Background2D, Background, Face, Expression, FAP, FDP, FIT, FaceDefMesh, FaceDefTable, FaceDefTransform, Viseme
Scaleable Texture	ImageTexture, Background2D, Background
Simple Face Animation	Face, Expression, FAP, FDP, FIT, FaceDefMesh, FaceDefTable, FaceDefTransform, Viseme
Advanced Scalable Texture	ImageTexture, Background2D, Background
Simple FBA	Face, Expression, FAP, FDP, FIT, FaceDefMesh, FaceDefTable, FaceDefTransform, Viseme, Body, BAP, BDP,BodyDefTable, BodySegmentConnectionHint

If the terminal complies with a 2D graphics profile only, the terminal may choose to ignore the contents of the FDP, FIT, FaceDefMesh, FaceDefTable, FaceDefTransform nodes.

7.9.2.3.3 3D Audio Scene Graph Profile

The 3D Audio Scene Graph profile provides tools for three-dimensional sound positioning in relation either with acoustic parameters of the scene or its perceptual attributes. The user can interact with the scene by changing the position of the sound source, changing the room effect or by moving the listening point.

The following list defines the 3D Audio Profile scene graph profiles:

AcousticScene, **Anchor**, **AudioBuffer**, **AudioClip**, **AudioDelay**, **AudioFX**, **AudioMix**, **AudioSource**, **AudioSwitch**, **Billboard**, **Conditional**, **DirectiveSound**, **Group**, **Inline**, **ListeningPoint**, **LOD**, **NavigationInfo**, **OrderedGroup**, **PerceptualParameters**, **QuantizationParameter**, **Sound**, **Sound2D**, **Switch**, **Transform**, **Viewpoint**, **WorldInfo**, Node Update, Route Update, Scene Update, **AnimationStream**, **Script**, **CoordinateInterpolator**, **OrientationInterpolator**, **PositionInterpolator**, **PositionInterpolator2D**, **ProximitySensor**, ROUTE, **TermCap**, **TimeSensor**, **TouchSensor**, **VisibilitySensor**, **Valuator**.

For nodes that are also included in the Audio Scene Graph Profile, values reported in Table 45 (and consequently Table 46) shall be used, with the exception of the number of spatialized sources, as reported in Table 47.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.9.2.4 Scene Graph Profiles@Levels

The following table gives the `sceneProfileLevelIndication` used in the initial object descriptor, as described in 14496-1.

Table 44 — sceneProfileLevelIndication Values

Value	Profile	Level
0x00	Reserved for ISO use	-
0x01	Simple 2D	L1
0x02	Simple 2D	L2
0x03	Audio	L1
0x04	Audio	L2
0x05	Audio	L3
0x06	Audio	L4
0x07	3D Audio	L1
0x08	3D Audio	L2
0x09	3D Audio	L3
0x0A	3D Audio	L4
0x0B	Basic 2D	L1
0x0C	Core 2D	L1
0x0D	Core 2D	L2
0x0E	Advanced 2D	L1
0x0F	Advanced 2D	L2
0x10	Advanced 2D	L3
0x11	Main 2D	L1
0x12	Main 2D	L2
0x13	Main 2D	L3
0x14-0x7F	reserved for ISO use	-
0x80-0xFD	user private	-
0xFE	no scene graph profile specified	-
0xFF	no scene graph capability required	-
NOTE — Usage of the value 0xFE indicates that the content described by this InitialObjectDescriptor does not comply to any scene graph profile specified in ISO/IEC 14496-1. Usage of the value 0xFF indicates that none of the scene graph profile capabilities are required for this content.		

7.9.2.4.1 Levels for the Audio Scene Graph Profile

7.9.2.4.1.1 Functionalities provided

The Audio scene graph profile provides for a set of BIFS scene graph elements for usage in audio only applications. The Audio scene graph profile supports applications like broadcast radio. When the Audio Scene graph profile is used, OT16 shall be supported.

7.9.2.4.1.2 Levels for the Audio Scene Graph Profile

In order to define Levels for the Audio Scene Graph Profile, the following parameters have been selected that may influence in a considerable way the decoding complexity of a bitstream.

Table 45 — BIFS Complexity restriction parameters

	Restriction parameters
Audio Feature	

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

BIFS Field Update	Maximum reaction time until a BIFS field update is audible
AudioMix, AudioSwitch, AudioSource	Maximum width, maximum depth of the sub-tree, click-free switching
AudioDelay, AudioClip, AudioBuffer	Total buffer memory, click-free delay
Sample Rate Conversion	Total conversion processing power, sample-rate conversion ratios.
AudioFX	According to the restrictions of SA approved by the Audio group (SAOL level definition based on abstract complexity metrics)
Sound, Sound2D	# spatialized

Parameters mentioned in the above table are defined as follows:

- Depth of an audio sub-tree: maximum number of consecutive nodes from the output of a **AudioSource** or **AudioClip** node to the input of a **Sound**/**Sound2D** node.
- Width of audio sub-tree: maximum number of parallel channels from the output of an **AudioSource** or **AudioClip** node to the input of a **Sound**/**Sound2D** node.
- Total Memory Buffer: an amount of memory needed to store samples shared between the different **AudioDelay**, **AudioClip** and **AudioBuffer** nodes present in a scene according to the formula:

Total Memory = ***SUM****(NbChannels(j)*NbBufferedSamples(j))*

where*: j is the considered node*

NbChannels is the number of channels for this node

*NbBufferedSamples = Delay(j)*SamplingFrequency(j)*

- Reaction Time of a BIFS field update is the maximum time in milliseconds. until the changes is audible.
- Total Conversion Processing Power: an amount of PCU shared among the different sampling conversions present in a scene according to: ISO/IEC 14496-3, Subpart 1, subclause 1.5.2 Audio Profiles and Levels, subclause 1.5.2.2 Complexity Units.
- Spatializable Objects: number of possible spatialized channels.
- **AudioFX**: see Table 47 below and abstract complexity metrics as defined in ISO/IEC 14496-4 (Conformance), Clause 6 (Audio Conformance) for the Algorithmic Synthesis and AudioFX Object Type.
- Reaction Time of a BIFS field update: the maximum time in milliseconds until the changes is audible.

Levels for the Audio scene graph profile are defined in the following table:

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 46 — Systems Audio Scene Graph Profile Levels

Audio Parameter	Level 1	Level 2	Level 3	Level 4
Reaction time [msec]	64	32	32	16
Width	8	32	64	128
Depth	1	4	6	8
Click free fadings	N	Y	Y	HQ
Total memory buffer	256 ksamples	512 ksamples	2 Megasamples	6 Megasamples (2s for 64 channels at 48 kHz)
SR Conversion ratio	1	INT	any allowed ratio	any allowed ratio
Total Conversion Processing Power	0 (sampling rate conversion is forbidden)	16 PCU	64 PCU	128 PCU
AudioFX	Very Low Complexity (Table 47)	Low Complexity (Table 47)	Medium Complexity (Table 47)	High Complexity (Table 47)
Spatialization	0	4	16	32

Table 47 — Complexity values for AudioFX node levels

Parameter	Very Low Complexity	Low Complexity	Medium Complexity	High Complexity
Total opcode calls	1M	1M	4M	8M
Floating-point operations	0	4M	12M	20M
Multiplications	0	2M	8M	16M
Tests	0	1M	4M	8M
Math methods	0	2M	6M	12M
Noise generators	0	0.05 M	0.2M	0.5M
Interpolations	0	0.3M	1.2M	2M
Multiply-and-add	2M	2M	4M	8M
Filters	0.2M	0.2M	1M	4M
Effects	96k	96k	0.4M	2M
Allocated memory	96k	96k	1M	16M

7.9.2.4.2 Levels for the Basic 2D Scene graph Profile

7.9.2.4.2.1 Functionalities provided

The Basic 2D Scene Graph profile is designed for very simple scenes that may handle only few (possibly only 1) audio and visual elements. This profile includes basic 2D composition and audio and video nodes interfaces. The application area for the Basic 2D Scene Graph Profile is related to audio-video only scene description.

The only allowed BIFS nodes for audio objects is : {**AudioSource**}

The allowed BIFS nodes for visual objects are : {**ImageTexture**, **MovieTexture**}

7.9.2.4.2.2 Level 1

Level 1 of Basic 2D Scene Graph Profile is used to describe 1 audio and/or 1 visual object only.

The following restrictions apply for the Basic 2D Scene Graph Profile at Level 1 :

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 48 — Restrictions for Basic 2D Scene Graph Profile at Level 1

Nodes	Restrictions
AudioSource	*addChildren, removeChildren, chidren, pitch*, *speed, numChan, phaseGroup* not supported
ImageTexture	*repeatS*, *repeatT* not supported
MovieTexture	*speed* ignored (no rewind nor fast forward) *repeatS*, *repeatT* not supported
OrderedGroup	*addChildren* and *removeChildren* not supported 2 *children* maximum allowed only 1 *OrderedGroup* node per scene used as top node
Sound2D	*intensity*, *spatialize*, *location* not supported

The usage of repeated **OrderedGroup** nodes to build bigger scenes is forbidden.

7.9.2.4.3 Levels for the Simple 2D Scene Graph Profile

7.9.2.4.3.1 Functionalities provided

The Simple 2D scene graph profile provides for only those BIFS scene graph elements necessary to place one or more audio-visual objects in a scene. The Simple 2D scene graph profile allows presentation of audio-visual content with potential update of the complete scene but no interaction capabilities. The Simple 2D scene graph profile supports applications like broadcast television. Note that the AudioClip node is not included in the Simple2D profile, irrespective of the Audio profile used.

7.9.2.4.3.2 Level 1

This level defines a scene that includes only audio and video objects; there are no capabilities to transform or manipulate the objects in the scene. It is intended for very simple, low complexity applications with image/video composition in 2D.

The following restrictions apply for the Simple 2D scene graph profile at Level 1:

Table 49 — Restrictions for Simple 2D scene graph profile at Level 1

Transform2D	
Field name	
addChildren	Ignored
removeChildren	Ignored
children	X.
center	Ignored
rotationAngle	0
scale	1, 1
scaleOrientation	0
translation	X
X = allowed; else: default value	

The metric shall be the pixel metrics. BIFSConfig.isPixel=1.

A cascade of Transform2D nodes is not allowed. Children nodes of a Transform2D node shall not be Transform2D nodes. Only one initial update to convey the complete scene graph is allowed.

7.9.2.4.3.3 Level 2

The following restrictions apply for the Simple 2D Graphics Profile at Level 2:

Table 50 — Restrictions for Simple 2D Scene Graph Profile at Level 2

Nodes	Restrictions
AudioClip	*pitch*, *description* not supported
AudioSource	*addChildren, removeChildren, chidren, pitch*, *speed, numChan,*

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

	PhaseGroup not supported
ImageTexture	*tepeatS*, *repeatT* not supported
MovieTexture	*speed* ignored (no rewind nor fast forward) *repeatS*, *repeatT* not supported
OrderedGroup	*addChildren* and *removeChildren* not supported *order* not supported 31 *children* maximum allowed
Scene Update	No restriction
Sound2D	*Intensity, spatialize*, *location* not supported
Transform2D	*addChildren* and *removeChildren* not supported *center*, *rotationAngle*, *ScaleOrientation* not supported (only translations and scalings are allowed) 31 *children* maximum allowed

For Simple 2D Scene Graph Profile @ Level 2, the maximum number of nodes in a scene is limited to 64 including all instances of these nodes through DEF/USE mechanism.

7.9.2.4.4 Levels for the Core 2D Scene Graph Profile

7.9.2.4.4.1 Functionalities provided

The Core 2D Scene Graph profile includes basic 2D composition, 2D texturing, local interaction, local animation, BIFS updates, quantization, access to web links and sub-scenes, in addition to audio and visual elements. It also introduces tools such as back channel (**ServerCommand**) and VoD features (**MediaControl**, **MediaSensor**).

7.9.2.4.4.2 Level 1

The following restrictions apply for the Core 2D Scene Graph Profile at Level 1:

Table 51 — Restrictions for Core 2D Scene Graph Profile at Level 1

Nodes	Restrictions
Anchor	*addChildren* and *removeChildren* not supported 31 *children* maximum allowed
AudioClip	*pitch*, *description* ignored
AudioSource	*addChildren, removeChildren, chidren, pitch*, *speed, numChan,* *phaseGroup* not supported (no rewind nor fast forward)
ColorInterpolator	255 key-value pairs
Conditional	No restriction
CoordinateInterpolator2D	31 coordinates per *keyValue* 255 key-value pairs
ImageTexture	*repeatS*, *repeatT* not supported
Inline	No restriction
InputSensor	restriction to mice, keyboards, remote controls
MediaControl	*mediaSpeed* not supported (no rewind nor fast forward) 31 *url* maximum
MediaSensor	*Info* ignored
MovieTexture	*speed* ignored (no rewind nor fast forward) *repeatS*, *repeatT* not supported
Node Update	Add and remove commands for children fields are not allowed
OrderedGroup	*addChildren* and *removeChildren* not supported 31 *children* maximum allowed
PositionInterpolator2D	255 key-value pairs
QuantizationParameter	No restriction
ROUTE	No restriction
ROUTE Update	No restriction
ScalarInterpolator	255 key-value pairs

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Scene Update	No restriction
ServerCommand	No restriction
Sound2D	*Intensity*, *spatialize*, *location* not supported
Switch	No restriction
TimeSensor	Ignored if *cycleInterval* < 0.03 second
TouchSensor	No restriction
Transform2D	*addChildren* and *removeChildren* not supported. *center*, *rotationAngle*, *scaleOrientation* not supported (only translations and scalings are allowed) 31 *children* maximum allowed. Negative *scale* not allowed.
Valuator	No restriction

The maximum number of nodes that is allowed in a scene compliant with the Core 2D Scene Graph Profile @ Level 1 is 8,191 including all instances of these nodes through DEF/USE mechanism or Inlined content.

7.9.2.4.4.3 Level 2

The following restrictions apply for the Core 2D Scene Graph Profile at Level 2:

Table 52 — Restrictions for Core 2D Scene Graph Profile at Level 2

Nodes	Restrictions
Anchor	127 *children* maximum allowed
AudioClip	*pitch*, *description* ignored
AudioSource	*addChildren, removeChildren, chidren, pitch*, *speed, numChan, phaseGroup* not supported (no rewind nor fast forward)
ColorInterpolator	255 key-value pairs
Conditional	No restriction
CoordinateInterpolator2D	127 coordinates per *keyValue* 255 key-value pairs
ImageTexture	*repeatS*, *repeatT* not supported
Inline	No restriction
InputSensor	restriction to mice, keyboards, remote controls
MediaControl	*mediaSpeed* not supported (no rewind nor fast forward) 127 *url* maximum
MediaSensor	*info* ignored
MovieTexture	*speed* ignored *repeatS*, *repeatT* not supported
Node Update	No restriction
OrderedGroup	127 *children* maximum allowed
PositionInterpolator2D	255 key-value pairs
QuantizationParameter	No restriction
ROUTE	No restriction
ROUTE Update	No restriction
ScalarInterpolator	255 key-value pairs
Scene Update	No restriction
ServerCommand	No restriction
Sound2D	*intensity*, *spatialize*, *location* not supported
Switch	No restriction
TimeSensor	Ignored if *cycleInterval* < 0.03 second
TouchSensor	No restriction
Transform2D	*addChildren* and *removeChildren* not supported *center*, *rotationAngle*, *scaleOrientation* not supported (only translations and scaling are allowed)

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

	127 *children* maximum allowed
Valuator	No restriction

The maximum number of nodes that is allowed in a scene compliant with the Core 2D Scene Graph Profile @ Level 2 is 32,767 including all instances of these nodes through DEF/USE mechanism or Inlined content.

7.9.2.4.5 Levels for the Main 2D Scene Graph Profile

7.9.2.4.5.1 Functionalities provided

The Main2D Scene Graph profile includes basic 2D composition, 2D texturing, local interaction, local animation, BIFS updates, access to web links and sub-scenes, in addition to audio and visual elements. It also introduces tools such as back channel (**ServerCommand**), VoD features (**MediaControl**, **MediaSensor**) and Flextime nodes (**TemporalGroup**, **TemporalTransform**) for advanced, flexible synchronization.

7.9.2.4.5.2 Level 1

The following restrictions apply for the Main2D Scene Graph Profile at Level 1:

Table 53 — Restrictions for Main2D Scene Graph Profile at Level 1

Nodes	Restrictions
Anchor	*addChildren* and *removeChildren* not supported 31 children maximum allowed
AudioClip	*pitch, description* ignored
AudioSource	*addChildren, removeChildren, chidren, pitch*, *speed, numChan, phaseGroup* not supported (no rewind nor fast forward)
ColorInterpolator	255 key-value pairs
Conditional	No restriction
CoordinateInterpolator2D	31 coordinates per *keyValue* 255 key-value pairs
DiscSensor	No restriction
Inline	Ignored
ImageTexture	*repeatS*, *repeatT* not supported; always treated as TRUE
InputSensor	restriction to mice, keyboards, remote controls
Layer2D	*addChildren* and *removeChildren* not supported 31 children maximum allowed
MediaControl	*mediaSpeed* not supported (no rewind nor fast forward) 31 *url* maximum
MediaSensor	*info* ignored, *speed* shall be 1.
MovieTexture	*speed* ignored (no rewind nor fast forward) *repeatS*, *repeatT* not supported; always treated as TRUE
Node Update	No restriction
OrderedGroup	*addChildren* and *removeChildren* not supported 31 children maximum allowed
PlaneSensor2D	No restriction
PositionInterpolator2D	255 key-value pairs
ProximitySensor2D	No restriction
QuantizationParameter	Ignored
ROUTE	No restriction
ROUTE Update	No restriction
ScalarInterpolator	255 key-value pairs
Scene Update	No restriction
ServerCommand	No restriction
Sound2D	*intensity*, *spatialize*, *location* not supported
Switch	31 choices maximum allowed

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TemporalGroup	*addChildren* and *removeChildren* not supported 7 children maximum allowed
TemporalTransform	*addChildren* and *removeChildren* not supported 31 children maximum allowed *speed* not supported *stretchMode* values linear and repeat not supported *shrinkMode* value linear not supported
TimeSensor	Ignored if *cycleInterval* < 0.03 second
TouchSensor	No restriction
Transform2D	*addChildren* and *removeChildren* not supported *center*, *rotationAngle*, *scaleOrientation* not supported (only translations and scalings are allowed) 31 children maximum allowed. Negative *scale* not allowed.
Valuator	No restriction

The maximum number of nodes that is allowed in a scene compliant with the Main2D Scene Graph Profile @ Level 1 is 8,191 including all instances of these nodes through DEF/USE mechanism or Inlined content.

NOTE — Where **addChildren** and **removeChildren** fields are not supported this refers only to in-scene routing. Node update is permitted to add and remove children from all MFNode fields.

7.9.2.4.5.3 Level 2

The following restrictions apply for the Main2D Scene Graph Profile at Level 2:

Table 54 — Restrictions for Main2D Scene Graph Profile at Level 2

Nodes	Restrictions
Anchor	*addChildren* and *removeChildren* not supported 255 children maximum allowed
AudioClip	*pitch*, *description* ignored
AudioSource	*addChildren, removeChildren, chidren, pitch*, *speed, numChan, phaseGroup* not supported (no rewind nor fast forward)
ColorInterpolator	511 key-value pairs
Conditional	No restriction
CoordinateInterpolator2D	511 coordinates per *keyValue* 511 key-value pairs
DiscSensor	No restriction
Inline	No more than 4 nodes in the scene at any one time. Restricted to supporting content to that created with Simple or Basic Profile Scene Graph, and Simple2D Graphics (Simple2D+Text)
ImageTexture	*repeatS*, *repeatT* not supported; always treated as TRUE
InputSensor	restriction to mice, keyboards, remote controls
Layer2D	*addChildren* and *removeChildren* not supported 255 children maximum allowed
MediaControl	255 *url* maximum speed adjustments supported, but audio resampling is not required
MediaSensor	*info* ignored
MovieTexture	*speed* ignored *repeatS*, *repeatT* not supported; always treated as TRUE
Node Update	No restriction
OrderedGroup	*addChildren* and *removeChildren* not supported 255 children maximum allowed
PlaneSensor2D	No restriction
PositionInterpolator2D	1023 key-value pairs
ProximitySensor2D	No restriction
QuantizationParameter	No restriction

-`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ROUTE	No restriction
ROUTE Update	No restriction
ScalarInterpolator	1023 key-value pairs
Scene Update	No restriction
ServerCommand	No restriction
Sound2D	*Spatialize*, *location* not supported
Switch	255 choices maximum allowed
TemporalGroup	*addChildren* and *removeChildren* not supported 15 children maximum allowed
TemporalTransform	*addChildren* and *removeChildren* not supported 255 children maximum allowed speed adjustments supported, but audio resampling is not required
TimeSensor	Ignored if *cycleInterval* < 0.03 second
TouchSensor	No restriction
Transform2D	*addChildren* and *removeChildren* not supported 255 children maximum allowed
Valuator	No restriction

The maximum number of nodes that is allowed in a scene compliant with the Main2D Scene Graph Profile @ Level 2 is 65,535 including all instances of these nodes through DEF/USE mechanism or Inlined content.

NOTE — Where **addChildren** and **removeChildren** fields are not supported this refers only to in-scene routing. Node update is permitted to add and remove children from all MFNode fields.

7.9.2.4.5.4 Level 3

The following restrictions apply for the Main2D Scene Graph Profile at Level 3:

Table 55 — Restrictions for Main2D Scene Graph Profile at Level 3

Nodes	Restrictions
Anchor	*addChildren* and *removeChildren* not supported 255 children maximum allowed
AudioClip	*pitch*, *description* ignored
AudioSource	*addChildren, removeChildren, chidren, pitch*, *speed, numChan, phaseGroup* not supported
ColorInterpolator	511 key-value pairs
Conditional	No restriction
CoordinateInterpolator2D	511 coordinates per *keyValue* 511 key-value pairs
DiscSensor	No restriction
Inline	Inlined content shall not exceed profiles of parent scene
ImageTexture	*repeatS*, *repeatT* not supported; always treated as TRUE
InputSensor	restriction to mice, keyboards, remote controls
Layer2D	*addChildren* and *removeChildren* not supported 255 children maximum allowed
MediaControl	255 *url* maximum speed adjustments supported, but audio resampling is not required
MediaSensor	*info* ignored
MovieTexture	*speed* ignored *repeatS*, *repeatT* not supported; always treated as TRUE
Node Update	No restriction
OrderedGroup	*addChildren* and *removeChildren* not supported 255 children maximum allowed
PlaneSensor2D	No restriction

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

PositionInterpolator2D	1023 key-value pairs
ProximitySensor2D	No restriction
QuantizationParameter	*isLocal*, *useEfficientCoding* not supported
ROUTE	No restriction
ROUTE Update	No restriction
ScalarInterpolator	1023 key-value pairs
Scene Update	No restriction
ServerCommand	No restriction
Sound2D	No restriction
Switch	255 choices maximum allowed
TemporalGroup	*addChildren* and *removeChildren* not supported 15 children maximum allowed
TemporalTransform	*addChildren* and *removeChildren* not supported 255 children maximum allowed speed adjustments supported, but audio resampling is not required
TimeSensor	Ignored if *cycleInterval* < 0.03 second
TouchSensor	No restriction
Transform2D	*addChildren* and *removeChildren* not supported 255 children maximum allowed
Valuator	No restriction

The maximum number of nodes that is allowed in a scene compliant with the Main2D Scene Graph Profile @ Level 3 is 65,535 including all instances of these nodes through DEF/USE mechanism or Inlined content.

NOTE — Where **addChildren** and **removeChildren** fields are not supported this refers only to in-scene routing. Node update is permitted to add and remove children from all MFNode fields.

7.9.2.4.6 Levels for the Advanced 2D Scene Graph Profile

7.9.2.4.6.1 Functionalities provided

The Advanced 2D Scene Graph profile comprises all Basic 2D and Core 2D Scene Graph functionalities. The Advanced 2D Scene Graph profile allows in addition : advanced 2D composition, advanced local interaction, streamed animation (BIFS-Anim.), scripting, advanced audio, PROTO.

7.9.2.4.6.2 Level 1

The following restrictions apply for the Advanced 2D Scene Graph Profile at Level 1:

Table 56 — Restrictions for Advanced 2D Scene Graph Profile at Level 1

Nodes	Restrictions
Anchor	511 *children* maximum allowed
AnimationStream	No restriction
AudioBuffer	No restriction
AudioClip	*pitch* not supported
AudioSource	63 *children* maximum allowed *pitch, speed, numChan, phaseGroup* not supported (no rewind nor fast forward)
AudioSwitch	No restriction
ColorInterpolator	1,023 key-value pairs
Conditional	No restriction
CoordinateInterpolator2D	128 coordinates per *keyValue* 1,023 key-value pairs
DiscSensor	No restriction
Group	511 *children* maximum allowed

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ImageTexture	No restriction
Inline	No restriction
InputSensor	No restriction
Layer2D	511 *children* maximum allowed
MediaBuffer	No restriction
MediaControl	255 *url* maximum speed adjustments supported but audio resampling is not required
MediaSensor	No restriction
MovieTexture	*speed* ignored (no rewind nor fast forward)
Node Update	No restriction
OrderedGroup	511 *children* maximum allowed
PlaneSensor2D	No restriction
PositionInterpolator2D	1,023 key-value pairs
PredictiveMF coding	No restriction
PROTO	31 fields 31 eventIns 31 eventOuts 31 exposedFields 7 levels
ProximitySensor2D	No restriction
QuantizationParameter	No restriction
ROUTE	No restriction
ROUTE Update	No restriction
ScalarInterpolator	1,023 key-value pairs
Scene Update	No restriction
Script	31 eventIns 31 eventOuts 31 fields
ServerCommand	No restriction
Sound2D	*spatialize*, *location* not supported
Switch	No restriction
TemporalGroup	*addChildren* and *removeChildren* not supported 15 children maximum allowed
TemporalTransform	*addChildren* and *removeChildren* not supported 255 children maximum allowed speed adjustments supported but audio resampling is not required
TermCap	No restriction
TimeSensor	Ignored if *cycleInterval* < 0.03 second
TouchSensor	No restriction
Transform2D	511 *children* maximum allowed
Valuator	No restriction
WorldInfo	No restriction

The maximum number of nodes that is allowed in a scene compliant with the Advanced 2D Scene Graph Profile @ Level 1 is 32,767 including all instances of these nodes through Inline, DEF/USE or PROTO mechanisms.

7.9.2.4.6.3 Level 2

The following restrictions apply for the Advanced 2D Scene Graph Profile at Level 2:

Table 57 — Restrictions for Advanced 2D Scene Graph Profile at Level 2

Nodes	Restrictions
Anchor	16,383 *children* maximum allowed

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

AnimationStream	No restriction
AudioBuffer	No restriction
AudioClip	*pitch* not supported
AudioSource	255 *children* maximum allowed *pitch, speed, numChan, phaseGroup* not supported (no rewind nor fast forward)
AudioSwitch	No restriction
ColorInterpolator	16,383 key-value pairs
Conditional	No restriction
CoordinateInterpolator2D	1,023 coordinates per *keyValue* 16,383 key-value pairs
DiscSensor	No restriction
Group	16,383 *children* maximum allowed
ImageTexture	No restriction
Inline	No restriction
InputSensor	No restriction
Layer2D	16,383 *children* maximum allowed
MediaBuffer	No restriction
MediaControl	255 *url* maximum speed adjustments supported but audio resampling is not required
MediaSensor	No restriction
MovieTexture	*speed* ignored (no rewind nor fast forward)
Node Update	No restriction
OrderedGroup	16,383 *children* maximum allowed
PlaneSensor2D	No restriction
PositionInterpolator2D	16,383 key-value pairs
PredictiveMF coding	No restriction
PROTO	255 fields 255 eventIns 255 eventOuts 255 exposedFields 7 levels
ProximitySensor2D	No restriction
QuantizationParameter	No restriction
ROUTE	No restriction
ROUTE Update	No restriction
ScalarInterpolator	16,383 key-value pairs
Scene Update	No restriction
Script	255 eventIns 255 eventOuts 255 fields
ServerCommand	No restriction
Sound2D	*spatialize*, *location* not supported
Switch	No restriction
TemporalGroup	*addChildren* and *removeChildren* not supported 15 children maximum allowed
TemporalTransform	*addChildren* and *removeChildren* not supported 255 children maximum allowed speed adjustments supported but audio resampling is not required
TermCap	No restriction
TimeSensor	Ignored if *cycleInterval* < 0.03 second
TouchSensor	No restriction
Transform2D	16,383 *children* maximum allowed

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Valuator	No restriction
WorldInfo	No restriction

The maximum number of nodes that is allowed in a scene compliant with the Advanced 2D Scene Graph Profile @ Level 2 is 131,071 including all instances of these nodes through Inline, DEF/USE or PROTO mechanism.

7.9.2.4.6.4 Level 3

The following restrictions apply for the Advanced 2D Scene Graph Profile at Level 2:

Table 58 — Restrictions for Advanced 2D Scene Graph Profile at Level 3

Nodes	Restrictions
Anchor	16,383 *children* maximum allowed
AnimationStream	No restriction
AudioBuffer	No restriction
AudioClip	*pitch* not supported
AudioSource	255 *children* maximum allowed *pitch, speed, numChan, phaseGroup* not supported (no rewind nor fast forward)
AudioSwitch	No restriction
ColorInterpolator	16,383 key-value pairs
Conditional	No restriction
CoordinateInterpolator2D	1,023 coordinates per *keyValue* 16,383 key-value pairs
DiscSensor	No restriction
Group	16,383 *children* maximum allowed
ImageTexture	No restriction
Inline	No restriction
InputSensor	No restriction
Layer2D	16,383 *children* maximum allowed
MediaBuffer	No restriction
MediaControl	255 *url* maximum
MediaSensor	No restriction
MovieTexture	*speed* ignored (no rewind nor fast forward)
Node Update	No restriction
OrderedGroup	16,383 *children* maximum allowed
PlaneSensor2D	No restriction
PositionInterpolator2D	16,383 key-value pairs
PredictiveMF coding	No restriction
PROTO	255 fields 255 eventIns 255 eventOuts 255 exposedFields 7 levels
ProximitySensor2D	No restriction
QuantizationParameter	No restriction
ROUTE	No restriction
ROUTE Update	No restriction
ScalarInterpolator	16,383 key-value pairs
Scene Update	No restriction
Script	255 eventIns 255 eventOuts

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

	255 fields
ServerCommand	No restriction
Sound2D	*spatialize*, *location* not supported
Switch	No restriction
TemporalGroup	*addChildren* and *removeChildren* not supported 15 children maximum allowed
TemporalTransform	*addChildren* and *removeChildren* not supported 255 children maximum allowed
TermCap	No restriction
TimeSensor	Ignored if *cycleInterval* < 0.03 second
TouchSensor	No restriction
Transform2D	16,383 *children* maximum allowed
Valuator	No restriction
WorldInfo	No restriction

The maximum number of nodes that is allowed in a scene compliant with the Advanced 2D Scene Graph Profile @ Level 3 is 131,071 including all instances of these nodes through Inline, DEF/USE or PROTO mechanism.

7.9.2.4.7 Levels for the Complete 2D Scene Graph Profile

7.9.2.4.7.1 Functionalities provided

The Complete 2D scene graph profile provides for all the 2D scene description elements of the BIFS tool. It supports features such as 2D transformations and alpha blending. The Complete 2D scene graph profile enables 2D applications that require extensive and customized interactivity.

7.9.2.4.7.2 Levels

No levels are yet defined for the Complete 2D scene graph profile.

7.9.2.4.8 Levels for the Complete Scene Graph Profile

7.9.2.4.8.1 Functionalities provided

The Complete scene graph profile provides the complete set of scene graph elements of the BIFS tool. The Complete scene graph profile will enable applications like dynamic virtual 3D world and games.

7.9.2.4.8.2 Levels

No levels are yet defined for the Complete scene graph profile..

7.9.2.4.9 Levels for the 3D Audio Profile

7.9.2.4.9.1 Functionalities Provided

The 3D Audio profile provides for a set of BIFS scene graph elements for usage in audio only applications. The 3D Audio profile supports applications with advanced 3D rendering of audio.

7.9.2.4.9.2 3D Audio Profile Level Definitions

In the following table, levels definitions for the 3D Audio profile are given. The levels are based on sampling rate of 44100 Hz at 16-bit resolution. Their complexities depend on:

- the maximum number of spatialized sources per scene (these spatialized sources can include discrete reflections that are perceptually equivalent to individual sound sources);

- the number of temporal sections whose levels and time limits can be controlled individually for each source;

- the maximum number of independent late reverberation processes per scene;

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- the maximum number of control frequencies in reverberation process filters, source directivity filters, and material filters.

Table 59 — 3D Audio Scene Graph Profile Levels.

Level	Level 1	Level 2	Level 3	Level 4
Maximum number of spatialized sources per scene	8	32	64	128
Number of temporal sections whose levels and time limits can be controlled individually for each source	1	1	2	3
Maximum number of independent late reverberation processes per scene	1	1	2	4
Maximum number of control frequencies in reverberation process filters, source directivity filters, and material filters	2	2	3	3

For nodes that are also included in the Audio Scene Graph Profile, criteria exposed in section 7.9.2.4.1.2 and values reported in Table 46 and Table 47 shall be used in addition, with the exception of the number of spatialized sources, as reported in Table 60.

Table 60 — 3D Audio Scene Graph Profile Levels - II.

Audio Parameter	Level 1	Level 2	Level 3	Level 4
Reaction time [msec]	64	32	32	16
Width	8	32	64	128
Depth	1	4	6	8
Click free fadings	N	Y	Y	HQ
Total memory buffer	256 ksamples	512 ksamples	2 Megasamples	6 Megasamples (2s for 64 channels at 48 kHz)
SR Conversion ratio	1	INT	any allowed ratio	any allowed ratio
Total Conversion Processing Power	0 (sampling rate conversion is forbidden)	16 PCU	64 PCU	128 PCU
AudioFX	Very Low Complexity (Table 47)	Low Complexity (Table 47)	Medium Complexity (Table 47)	High Complexity (Table 47)

7.9.3 Graphics Profile Definitions

7.9.3.1 Overview

The graphics profiles specify the graphics elements of the BIFS tool that are allowed. These elements provide means to represent graphics visual objects in a scene. Profiling of graphics elements of the BIFS tool serves to restrict the memory requirements for the storage of the graphical elements as well as to restrict the computational complexities of composition and rendering processes.

7.9.3.2 Graphics Profiles Tools

The following tools are available to construct the graphics profiles:

BIFS nodes related to graphics as defined in Table 61.

3D Audio Graphics Profile as defined in subclause 7.9.3.3.1.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.9.3.3 Graphics Profiles

The following table defines the graphics profiles:

Table 61 — Graphics profiles

	Graphics Profiles						
Graphics Tools	Simple2D	Simple2D +Text	Core 2D	Advanced 2D	Complete 2D	3D Audio	Complete
AcousticMaterial						X	
Appearance	X	X	X	X	X	X	X
Background							X
Background2D		X	X	X	X		X
BAP							
BDP							
Bitmap	X	X	X	X	X		X
Body							
BodyDefTable							
BodySegment ConnectionHint							
Box							X
Circle			X	X	X		X
Color			X	X	X		X
Cone							X
Coordinate						X	X
Coordinate2D			X	X	X		X
Curve2D				X	X		X
Cylinder							X
DirectionalLight							X
ElevationGrid							X
Expression							X
Extrusion							X
Face							X
FaceDefMesh							X
FaceDefTable							X
FaceDefTransform							X
FAP							X
FDP							X
FIT							X
Fog							X
FontStyle		X	X	X	X		X
IndexedFaceSet						X	X
IndexedFaceSet2D			X	X	X		X
IndexedLineSet							X
IndexedLineSet2D				X	X		X
LineProperties				X	X		X
Material							X
Material2D		X	X	X	X		X
MaterialKey				X			
MatteTexture				X			
Normal						X	X
PixelTexture			X	X	X		X
PointLight							X
PointSet							X
PointSet2D					X		X
Rectangle		X	X	X	X		X
Shape	X	X	X	X	X	X	X
Sphere							X

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SpotLight							X
Text		X	X	X	X		X
TextureCoordinate				X	X		X
TextureTransform				X	X		X
Viseme							X

Decoders that claim compliance to a given profile shall implement all the tools with an 'X' entry for that profile.

7.9.3.3.1 3D Audio Graphics Profile

This profile defines graphics tools that are required to define the acoustical properties of the scene (geometry, acoustics absorption, diffusion, transparency of the material). The following list defines the 3D Audio Graphics profile: **AcousticMaterial**, **Appearance**, **Coordinate**, **IndexedFaceSet**, **Normal**, **Shape**.

7.9.3.4 Graphics Profiles@Levels

The following table gives the `graphicsProfileLevelIndication` used in the initial object descriptor, as described in 14496-1.

Table 62 — graphicsProfileLevelIndication Values

Value	Profile	Level
0x00	Reserved for ISO use	
0x01	Simple2D profile	L1
0x02	Simple 2D + Text profile	L1
0x03	Simple 2D + Text profile	L2
0x04	Core 2D profile	L1
0x05	Core 2D profile	L2
0x06	Advanced 2D profile	L1
0x07	Advanced 2D profile	L2
0x08-0x7F	reserved for ISO use	
0x80-0xFD	user private	
0xFE	no graphics profile specified	
0xFF	no graphics capability required	
NOTE — Usage of the value 0xFE may indicate that the content described by this InitialObjectDescriptor does not comply to any conformance point specified in ISO/IEC 14496-1. Usage of the value 0xFF indicates that none of the graphics profile capabilities are required for this content.		

7.9.3.4.1 Levels for the Simple 2D Graphics Profile

7.9.3.4.1.1 Functionalities provided

The Simple 2D graphics profile provides for only those graphics elements of the BIFS tool that are necessary to place one or more visual objects in a scene.

7.9.3.4.1.2 Levels

The following restrictions apply for the Simple 2D Graphics Profile at Level 1:

Table 63 — Restrictions for Simple 2D Graphics Profile at Level 1

Nodes	Restrictions
Appearance	*material* not supported *textureTransform* not supported
Bitmap	No restriction
Shape	No restriction

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.9.3.4.2 Levels for the Simple 2D + Text Graphics Profile

7.9.3.4.2.1 Functionalities provided

The Simple 2D + Text Graphics profile is designed for applications where the only graphics to be used are text elements (possibly colored or transparent, and maybe in addition to audio and visual objects).

7.9.3.4.2.2 Level 1

The following restrictions apply for the Simple 2D + Text Graphics Profile at Level 1:

Table 64 — Restrictions for Simple 2D + Text Graphics Profile at Level 1

Nodes	Restrictions
Appearance	*textureTransform* not supported
Background2D	only 1 Background2D node allowed in a scene for color only background *url*, *set_bind* not supported
Bitmap	No restriction
FontStyle	No restriction
Material2D	*lineProps* not supported Used for colored and/or transparent text and transparent visual objects
Rectangle	No restriction
Shape	No restriction
Text	*maxExtent* not supported No texture mapping allowed 1,200 characters maximum in the scene at a time

7.9.3.4.3 Levels for the Core 2D Graphics Profile

7.9.3.4.3.1 Functionalities provided

The Core 2D Graphics profile is designed for applications using some simple graphics elements (may be in addition to audio and visual objects).

7.9.3.4.3.2 Level 1

The following restrictions apply for the Core 2D Graphics Profile at Level 1 :

Table 65 — Restrictions for Core 2D Graphics Profile at Level 1

Nodes	Restrictions
Appearance	*textureTransform* not supported
Background2D	only 1 Background2D node allowed in a scene for color and image background only *set_bind* not supported
Bitmap	No restriction
Circle	No texture mapping allowed
Color	No restriction (not used at this level)
Coordinate2D	4 *points* maximum
FontStyle	No restriction

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IndexedFaceSet2D	15 *IndexedFaceSet2D* nodes maximum in a scene *set_colorIndex*, *set_coordIndex*, *set_texCoordIndex* not supported EventIns are ignored, the only field that can be modified is coord color not supported *colorIndex*, *colorPerVertex* , *texCoordIndex* not supported The number of points is restricted to be equal to 4 (quadrilateral) texCoord field is always considered to be (00 10 11 01) coordIndex field is always considered to be (0 1 2 3 -1) convex is always considered to be TRUE Face list shall be well-defined as follows : 1. Each face contains at least three non-coincident vertices 2. A given *coordIndex* is not repeated in a face 3. The vertices of a face shall define a planar polygon 4. The vertices of a face shall not define a self-intersecting polygon
Material2D	*lineProps* not supported
PixelTexture	32x32 maximum image size 8 PixelTexture nodes maximum in a scene at a time
Rectangle	No restriction
Shape	No restriction
Text	*maxExtent* not supported No texture mapping allowed 6,480 characters maximum in the scene at a time

7.9.3.4.3.3 Level 2

The following restrictions apply for the Core 2D Graphics Profile at Level 2 :

Table 66 — Restrictions for Core 2D Graphics Profile at Level 2

Nodes	Restrictions
Appearance	*textureTransform* not supported
Background2D	only 1 Background2D node allowed in a scene for color and image backgroound only *set_bind* not supported
Bitmap	No restriction
Circle	No texture mapping allowed
Color	255 *colors* maximum in the scene at a time
Coordinate2D	255 *points* maximum in the scene at a time
FontStyle	No restriction
IndexedFaceSet2D	31 *IndexedFaceSet2D* nodes maximum in a scene *set_colorIndex*, *set_coordIndex*, *set_texCoordIndex* not supported EventIns are ignored *colorIndex*, *colorPerVertex*, *texCoordIndex* not supported convex is always considered to be TRUE 255 total indices maximum in all index fields in the scene Face list shall be well-defined as follows : 1. Each face is terminated with -1, including the last face in the array 2. Each face contains at least three non-coincident vertices 3. A given *coordIndex* is not repeated in a face 4. The vertices of a face shall define a planar polygon 5. The vertices of a face shall not define a self-intersecting polygon
Material2D	*lineProps* not supported
PixelTexture	32x32 maximum image size 8 PixelTexture nodes maximum in a scene at a time
Rectangle	No restriction
Shape	No restriction

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Text	*maxExtent* not supported No texture mapping allowed 6,480 characters maximum in the scene at a time

7.9.3.4.4 Levels for the Advanced 2D Graphics Profile

7.9.3.4.4.1 Functionalities provided

The Advanced 2D Graphics profile is designed for applications using advanced graphics elements (possibly in addition to audio and visual objects).

7.9.3.4.4.2 Level 1

The following restrictions apply for the Advanced 2D Graphics Profile at Level 1 :

Table 67 — Restrictions for Advanced 2D Graphics Profile at Level 1

Nodes	Restrictions
Appearance	No restriction
Background2D	No restriction
Bitmap	No restriction
Circle	No restriction
Color	65,535 *colors* maximum in the scene at a time
Coordinate2D	65,535 *points* maximum in the scene at a time
Curve2D	255 *Curve2D* nodes maximum in a scene 65,535 total elements in all type fields in the scene
FontStyle	No restriction
IndexedFaceSet2D	255 *IndexedFaceSet2D* nodes maximum in a scene 65,535 total indices maximum in all index fields in the scene Face list shall be well-defined as follows : 1. Each face is terminated with -1, including the last face in the array 2. Each face contains at least three non-coincident vertices 3. A given *coordIndex* is not repeated in a face 4. The vertices of a face shall define a planar polygon 5. The vertices of a face shall not define a self-intersecting polygon
IndexedLineSet2D	255 *IndexedLineSet2D* nodes maximum in a scene 65,535 total indices maximum in all index fields in the scene
LineProperties	No restriction
Material2D	No restriction
MaterialKey	No restriction
MatteTexture	Only one MatteTexture node allowed No BLUR function for s>2 Total pixel area of MatteTexture less than or equal to twice CIF
PixelTexture	32x32 maximum image size 8 PixelTexture nodes maximum in a scene at a time
Rectangle	No restriction
Shape	No restriction
Text	12,288 characters maximum in the scene at a time
TextureCoordinate	65,535 coordinates maximum in the scene at a time
TextureTransform	No restriction

7.9.3.4.4.3 Level 2

The following restrictions apply for the Advanced 2D Graphics Profile at Level 2:

Table 68 — Restrictions for Advanced 2D Graphics Profile at Level 2

Nodes	Restrictions

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Appearance	No restriction
Background2D	No restriction
Bitmap	No restriction
Circle	No restriction
Color	65,535 *colors* maximum in the scene at a time
Coordinate2D	65,535 *points* maximum in the scene at a time
Curve2D	32,767 *Curve2D* nodes maximum in a scene 1,048,575 total elements in all type fields in the scene
FontStyle	No restriction
IndexedFaceSet2D	32,767 *IndexedFaceSet2D* nodes maximum in a scene 1,048,575 total indices maximum in all index fields in the scene
IndexedLineSet2D	32,767 *IndexedLineSet2D* nodes maximum in a scene 1,048,575 total indices maximum in all index fields in the scene
LineProperties	No restriction
Material2D	No restriction
MaterialKey	No restriction
MatteTexture	No BLUR function for s>2 Total pixel area of MatteTexture less than or equal to twice CCIR 601
PixelTexture	32x32 maximum image size 8 PixelTexture nodes maximum in a scene at a time
Rectangle	No restriction
Shape	No restriction
Text	1,048,575 characters maximum in the scene at a time
TextureCoordinate	65,535 coordinates maximum in the scene at a time
TextureTransform	No restriction

7.9.3.4.5 Levels for the Complete 2D Graphics Profile

7.9.3.4.5.1 Provided functionality

The Complete 2D graphics profile provides two-dimensional graphics functionalities and supports features such as arbitrary two-dimensional graphics and text, possibly in conjunction with visual objects.

7.9.3.4.5.2 Levels

No levels are yet defined for the Complete 2D graphics profile. Future definition of Levels is anticipated; this will happen by means of an amendment to this part of the standard.

7.9.3.4.6 Levels for the Complete Graphics Profile

7.9.3.4.6.1 Provided functionality

The Complete graphics profile provides advanced graphical elements such as elevation grids and extrusions and allows creating content with sophisticated lighting. The Complete Graphics profile enables applications such as complex virtual worlds that exhibit a high degree of realism.

7.9.3.4.6.2 Levels

No levels are yet defined for the Complete Graphics profile. Future definition of Levels is anticipated; this will happen by means of an amendment to this part of the standard.

7.9.3.4.7 3D Audio Graphics Level Definitions

No visual rendering should be implemented.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.9.4 MPEG-J Profile Definitions

7.9.4.1 Overview

MPEG-J specifies the format, delivery, and behavior of downloadable byte code on MPEG-4 terminals. This enables content owners to embed complex control algorithms with the data. MPEG-J aplications, however, can be local or remote (MPEGlet). These applications use a specified set of Java APIs.

7.9.5 MPEG-J Profiles Tools

The following API packages are available to construct MPEG-J profiles:

Scene APIs (package org.iso.mpeg.mpegj.scene) as defined in 10.4.3.

Resource APIs (package org.iso.mpeg.mpegj.resource) as defined in 10.4.3.4.

Net APIs (package org.iso.mpeg.mpegj.net) as defined in 10.4.6.

Decoder APIs (package org.iso.mpeg.mpegj.decoder) as defined in 10.4.5.

Section Filtering and Service Information as defined in 10.4.7.

Please note that the package org.iso.mpeg.mpegj is required in all terminals.

7.9.6 MPEG-J Profiles

The MPEG-J profiles are defined in Table 69. Currently, the are two profiles defined, comprising all the API packages.

The Personal profile addresses a range of constrained devices ranging from mobile and portable devices up to personal computers. Examples of such devices are cell videophones, PDAs, personal gaming devices, multimedia computers, etc.

The Main profile is a superset of Personal profile and it addresses the broadcast oriented devices including entertainment devices. Examples of such devices are set top boxes, digital TVs, etc.

Table 69 — MPEG-J Profiles

MPEG-J Packages	MPEG-J Profiles	
	Personal	Main
Scene	X	X
Resource	X	X
Decoder	X	X
Net	X	X
SI/SF		X

Decoders that claim compliance to a given profile shall implement all the packages with an ‘X’ entry for that profile and the org.iso.mpeg.mpegj package (required for all profiles).

7.9.7 MPEG-J Profiles@Levels

7.9.7.1 Levels for the Personal MPEG-J Profile

No levels are defined yet for the MPEG-J Personal profile. No Levels are foreseen at the moment, but the possibility of adding Levels through amendments is left open.

7.9.7.2 Levels for the Main MPEG-J Profile

No levels are defined yet for the MPEG-J Main profile. No Levels are foreseen at the moment, but the possibility of adding Levels through amendments is left open.

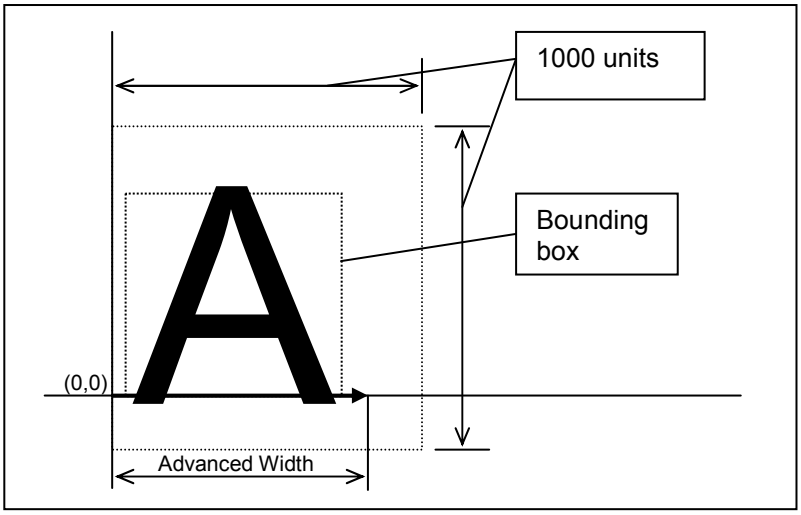
--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.10 Metric information for resident fonts

All measurements and information on font metrics presented in this document is given in units equal to 1/1000 of the scale factor and are defined by the coordinate system used in the ISO 9541 “Font Information Interchange”. The origin of the coordinate system for roman characters, in which these units are defined, is located on the baseline to the left of the character. The X axis runs along baseline, as illustrated below:



The font metrics information for MPEG-4 resident fonts is derived from the AFM files AlbanyforMPEG.AFM, CumberlandforMPEG.AFM and ThorndaleforMPEG.AFM included in ISO/IEC 14496-5.

7.11 Font metrics for SANS SERIF font (Albany)

7.11.1 Global font information

The following table provides global font information:

Character Set	Basic Latin, Latin-1 Supplement
Weight	Regular
Italic Angle	0
Fixed Pitch	false
Writing Direction	0 (horizontal)
Underline Position	-100
Underline Thickness	50
Cap Height	689
X height	509
Ascender	726
Descender	-199
Font Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})	-184, -210, 1002, 933

7.11.2 Character metrics

Character Unicode	Advanced Width (WX)	Character Name	Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})
U+0020	278	space	0, 0, 0, 0
U+0021	278	exclam	80, -8, 197, 689
U+0022	355	quotedbl	50, 435, 305, 689
U+0023	556	numbersign	21, 0, 535, 689
U+0024	556	dollar	44, -103, 509, 782

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Character Unicode	Advanced Width (WX)	Character Name	Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})
U+0025	889	percent	58, -13, 831, 702
U+0026	667	ampersand	40, -14, 660, 702
U+0027	191	quotesingle	50, 435, 140, 689
U+0028	333	parenleft	68, -162, 301, 726
U+0029	333	parenright	33, -162, 266, 726
U+002A	389	asterisk	18, 311, 371, 689
U+002B	584	plus	55, 96, 529, 570
U+002C	278	comma	77, -117, 202, 108
U+002D	333	hyphen	32, 215, 301, 298
U+002E	278	period	80, -8, 197, 108
U+002F	278	slash	0, -12, 278, 702
U+0030	556	zero	45, -13, 511, 702
U+0031	556	one	97, 0, 362, 695
U+0032	556	two	40, 0, 498, 702
U+0033	556	three	27, -13, 497, 702
U+0034	556	four	28, 0, 525, 694
U+0035	556	five	34, -13, 504, 689
U+0036	556	six	51, -12, 517, 702
U+0037	556	seven	51, 0, 499, 689
U+0038	556	eight	43, -13, 512, 702
U+0039	556	nine	44, -13, 510, 702
U+003A	278	colon	80, -8, 197, 518
U+003B	278	semicolon	77, -117, 202, 518
U+003C	584	less	55, 90, 529, 574
U+003D	584	equal	55, 189, 529, 477
U+003E	584	greater	55, 90, 529, 574
U+003F	556	question	51, -8, 492, 702
U+0040	1015	at	68, -197, 948, 702
U+0041	667	A	5, 0, 663, 689
U+0042	667	B	80, 0, 616, 689
U+0043	722	C	47, -12, 691, 702
U+0044	722	D	80, 0, 672, 689
U+0045	667	E	80, 0, 606, 689
U+0046	611	F	80, 0, 557, 689
U+0047	778	G	47, -13, 698, 702
U+0048	722	H	80, 0, 642, 689
U+0049	278	I	94, 0, 184, 689
U+004A	500	J	21, -12, 420, 689
U+004B	667	K	80, 0, 661, 689
U+004C	556	L	80, 0, 526, 689
U+004D	833	M	80, 0, 753, 689
U+004E	722	N	80, 0, 643, 689
U+004F	778	O	47, -12, 731, 702
U+0050	667	P	80, 0, 610, 689
U+0051	778	Q	47, -147, 732, 702
U+0052	722	R	80, 0, 704, 689
U+0053	667	S	60, -12, 608, 701
U+0054	611	T	21, 0, 589, 689
U+0055	722	U	77, -13, 646, 689
U+0056	667	V	4, 0, 663, 689
U+0057	944	W	15, 0, 928, 689
U+0058	667	X	24, 0, 646, 689
U+0059	667	Y	5, 0, 662, 689
U+005A	611	Z	33, 0, 578, 689
U+005B	278	bracketleft	68, -199, 262, 726

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Character Unicode	Advanced Width (WX)	Character Name	Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})
U+005C	278	backslash	0, -12, 278, 702
U+005D	278	bracketright	16, -199, 210, 726
U+005E	469	asciicircum	27, 311, 444, 702
U+005F	500	underscore	-6, -115, 506, -65
U+0060	333	grave	56, 566, 256, 703
U+0061	556	a	39, -14, 510, 521
U+0062	556	b	68, -13, 516, 726
U+0063	500	c	40, -14, 473, 522
U+0064	556	d	40, -13, 488, 726
U+0065	556	e	40, -14, 515, 521
U+0066	278	f	4, 0, 315, 734
U+0067	556	g	40, -210, 488, 522
U+0068	556	h	68, 0, 490, 726
U+0069	222	i	60, 0, 163, 726
U+006A	222	j	-54, -210, 163, 726
U+006B	500	k	68, 0, 496, 726
U+006C	222	l	69, 0, 154, 726
U+006D	833	m	68, 0, 767, 522
U+006E	556	n	68, 0, 490, 522
U+006F	556	o	40, -13, 517, 521
U+0070	556	p	68, -199, 516, 522
U+0071	556	q	40, -199, 488, 522
U+0072	333	r	68, 0, 342, 521
U+0073	500	s	49, -13, 457, 521
U+0074	278	t	14, -14, 278, 648
U+0075	556	u	66, -13, 488, 509
U+0076	500	v	8, 0, 492, 509
U+0077	722	w	12, 0, 708, 509
U+0078	500	x	15, 0, 484, 509
U+0079	500	y	-6, -209, 492, 509
U+007A	500	z	31, 0, 481, 509
U+007B	334	braceleft	28, -210, 310, 728
U+007C	260	bar	92, -189, 168, 726
U+007D	334	braceright	24, -210, 306, 728
U+007E	584	asciitilde	43, 262, 542, 406
U+00A1	333	exclamdown	109, -176, 226, 521
U+00A2	556	cent	67, -119, 500, 612
U+00A3	556	sterling	56, 0, 528, 702
U+00A4	556	currency	38, 93, 518, 572
U+00A5	556	yen	2, 0, 555, 689
U+00A6	260	brokenbar	92, -189, 168, 726
U+00A7	556	section	47, -185, 513, 726
U+00A8	333	dieresis	25, 583, 309, 686
U+00A9	737	copyright	10, -13, 726, 702
U+00AA	370	ordfeminine	31, 355, 341, 702
U+00AB	556	guillemotleft	44, 55, 512, 461
U+00AC	584	logicalnot	55, 177, 529, 477
U+00AE	737	registered	10, -13, 726, 702
U+00B0	400	degree	67, 437, 333, 702
U+00B1	584	plusminus	55, 0, 529, 600
U+00B2	333	twosuperior	25, 280, 308, 702
U+00B3	333	threesuperior	14, 273, 307, 702
U+00B4	333	acute	77, 566, 277, 703
U+00B5	584	mu	68, -165, 490, 509
U+00B6	537	paragraph	9, -199, 529, 690

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

Character Unicode	Advanced Width (WX)	Character Name	Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})
U+00B8	333	cedilla	57, -210, 271, 18
U+00B9	333	onesuperior	58, 280, 230, 695
U+00BA	365	ordmasculine	24, 354, 338, 702
U+00BB	556	guillemotright	44, 55, 512, 461
U+00BC	834	onequarter	61, -13, 789, 702
U+00BD	834	onehalf	61, -13, 807, 702
U+00BE	834	threequarters	24, -13, 789, 702
U+00BF	611	questiondown	98, -189, 539, 521
U+00C0	667	Agrave	5, 0, 663, 873
U+00C1	667	Aacute	5, 0, 663, 873
U+00C2	667	Acircumflex	5, 0, 663, 873
U+00C3	667	Atilde	5, 0, 663, 862
U+00C4	667	Adieresis	5, 0, 663, 856
U+00C5	667	Aring	5, 0, 663, 933
U+00C6	1000	AE	5, 0, 944, 689
U+00C7	722	Ccedilla	47, -210, 691, 702
U+00C8	667	Egrave	80, 0, 606, 873
U+00C9	667	Eacute	80, 0, 606, 873
U+00CA	667	Ecircumflex	80, 0, 606, 873
U+00CB	667	Edieresis	80, 0, 606, 856
U+00CC	278	Igrave	20, 0, 220, 873
U+00CD	278	Iacute	60, 0, 260, 873
U+00CE	278	Icircumflex	-4, 0, 281, 873
U+00CF	278	Idieresis	-3, 0, 281, 856
U+00D0	722	Eth	10, 0, 672, 689
U+00D1	722	Ntilde	80, 0, 643, 862
U+00D2	778	Ograve	47, -12, 731, 873
U+00D3	778	Oacute	47, -12, 731, 873
U+00D4	778	Ocircumflex	47, -12, 731, 873
U+00D5	778	Otilde	47, -12, 731, 862
U+00D6	778	Odieresis	47, -12, 731, 856
U+00D7	584	multiply	69, 110, 515, 556
U+00D8	778	Oslash	41, -14, 738, 702
U+00D9	722	Ugrave	77, -13, 646, 873
U+00DA	722	Uacute	77, -13, 646, 873
U+00DB	722	Ucircumflex	77, -13, 646, 873
U+00DC	722	Udieresis	77, -13, 646, 856
U+00DD	667	Yacute	5, 0, 662, 873
U+00DE	667	Thorn	80, 0, 620, 689
U+00DF	611	germandbls	68, -13, 577, 726
U+00E0	556	agrave	39, -14, 510, 703
U+00E1	556	aacute	39, -14, 510, 703
U+00E2	556	acircumflex	39, -14, 510, 703
U+00E3	556	atilde	39, -14, 510, 692
U+00E4	556	adieresis	39, -14, 510, 686
U+00E5	556	aring	39, -14, 510, 763
U+00E6	889	ae	39, -14, 849, 521
U+00E7	500	ccedilla	40, -210, 473, 522
U+00E8	556	egrave	40, –14, 515, 703
U+00E9	556	eacute	40, -14, 515, 703
U+00EA	556	ecircumflex	40, -14, 515, 703
U+00EB	556	edieresis	40, -14, 515, 686
U+00EC	278	igrave	23, 0, 223, 703
U+00ED	278	iacute	61, 0, 261, 703
U+00EE	278	icircumflex	-5, 0, 280, 703

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Character Unicode	Advanced Width (WX)	Character Name	Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})
U+00EF	278	idieresis	-4, 0, 280, 686
U+00F0	556	eth	40, -13, 517, 727
U+00F1	556	ntilde	68, 0, 490, 692
U+00F2	556	ograve	40, -13, 517, 703
U+00F3	556	oacute	40, -13, 517, 703
U+00F4	556	ocircumflex	40, -13, 517, 703
U+00F5	556	otilde	40, -13, 517, 692
U+00F6	556	odieresis	40, -13, 517, 686
U+00F7	584	divide	55, 96, 529, 570
U+00F8	611	oslash	20, -53, 591, 562
U+00F9	556	ugrave	66, -13, 488, 703
U+00FA	556	uacute	66, -13, 488, 703
U+00FB	556	ucircumflex	66, -13, 488, 703
U+00FC	556	udieresis	66, -13, 488, 686
U+00FD	500	yacute	-6, -209, 492, 703
U+00FE	556	thorn	68, -199, 516, 726
U+00FF	500	ydieresis	-6, -209, 492, 686
U+0131	278	dotlessi	97, 0, 182, 509
U+0141	556	Lslash	-21, 0, 526, 689
U+0142	222	lslash	-22, 0, 252, 726
U+0152	1000	OE	47, -12, 944, 702
U+0153	944	oe	40, -14, 903, 521
U+0160	667	Scaron	60, -12, 608, 873
U+0161	500	scaron	49, -13, 457, 703
U+0178	667	Ydieresis	5, 0, 662, 856
U+017D	611	Zcaron	33, 0, 578, 873
U+017E	500	zcaron	31, 0, 481, 703
U+0192	556	florin	29, -194, 511, 734
U+02C6	333	circumflex	25, 566, 310, 703
U+02C7	333	caron	24, 566, 309, 703
U+02C9	333	macron	19, 599, 314, 672
U+02D8	333	breve	22, 570, 311, 697
U+02D9	333	dotaccent	115, 598, 218, 701
U+02DA	333	ring	61, 551, 273, 763
U+02DB	333	ogonek	89, -208, 300, 12
U+02DC	333	tilde	3, 579, 330, 692
U+02DD	333	hungarumlaut	-13, 566, 347, 703
U+2013	556	endash	-2, 224, 558, 294
U+2014	1000	emdash	-2, 224, 1002, 294
U+2018	222	quoteleft	50, 478, 175, 703
U+2019	222	quoteright	48, 477, 173, 702
U+201A	222	quotesinglbase	46, -117, 171, 108
U+201C	333	quotedblleft	15, 478, 319, 703
U+201D	333	quotedblright	13, 477, 317, 702
U+201E	333	quotedblbase	13, -117, 317, 108
U+2020	556	dagger	39, -189, 517, 726
U+2021	556	daggerdbl	39, -189, 517, 726
U+2022	350	bullet	51, 178, 301, 428
U+2026	1000	ellipsis	109, -8, 892, 108
U+2030	1000	perthousand	20, -12, 979, 702
U+2039	333	guilsinglleft	44, 55, 281, 461
U+203A	333	guilsinglright	53, 55, 290, 461
U+20AC	556	Euro	4, -13, 535, 702
U+2122	1000	trademark	72, 276, 878, 689
U+2212	584	minus	55, 295, 529, 371

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Character Unicode	Advanced Width (WX)	Character Name	Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})
U+2215	167	fraction	-184, -13, 350, 702
U+2219	278	periodcentered	80, 284, 197, 400
U+FB01	500	fi	4, 0, 441, 734
U+FB02	500	fl	4, 0, 432, 734

7.11.3 Kerning Data

Kerning Pair	Value
space A	-55
space T	-18
space Y	-18
one one	-74
A space	-55
A T	-74
A V	-74
A W	-37
A Y	-74
A v	-18
A w	-18
A y	-18
A quoterigh	t -74
F comma	-111
F period	-111
F A	-55
L space	-37
L T	-74
L V	-74
L W	-74
L Y	-74
L y	-37
L quoteright	-55
P space	-18
P comma	-129
P period	-129
P A	-74
R T	-18
R V	-18
R W	-18
R Y	-18
T space	-18
T comma	-111
T hyphen	-55
T period	-111
T colon	-111
T semicolon	-111
T A	–74
T O	-18
T a	-111
T c	-111
T e	-111
T I	-37
T o	-111
T r	-37
T s	-111

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Kerning Pair	Value
T u	-37
T w	-55
T y	-55
V comma	-92
V hyphen	-55
V period	-92
V colon	-37
V semicolon	-37
V A	-74
V a	-74
V e	-55
V l	-18
V o	-55
V r	-37
V u	-37
V y	-37
W comma	-55
W hyphen	-18
W period	-55
W colon	-18
W semicolon	-18
W A	-37
W a	-37
W e	-18
W o	-18
W r	-18
W u	-18
W y	-9
Y space	-18
Y comma	-129
Y hyphen	-92
Y period	-129
Y colon	-55
Y semicolon	-65
Y A	-74
Y a	-74
Y e	-92
Y l	-37
Y o	-92
Y p	-74
Y q	-92
Y u	-55
Y v	-55
f f	-18
f quoteright	18
r comma	-55
r period	-55
r quoteright	37
v comma	-74
v period	-74
w comma	-55
w period	-55
y comma	-74
y period	-74
quoteleft quoteleft	-18
quoteright space	-37

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Kerning Pair	Value
quoteright s	-18
quoteright quoteright	-18

7.12 Font metrics for SERIF font (Thorndale)

7.12.1 Global font information

The following table provides global font information:

Character Set	Basic Latin, Latin-1 Supplement
Weight	Regular
Italic Angle	0
Fixed Pitch	False
Writing Direction	0 (horizontal)
Underline Position	-100
Underline Thickness	50
Cap Height	662
X height	457
Ascender	694
Descender	-214
Font Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})	-166, -216, 1009, 877

7.12.2 Character metrics

Character Unicode	Advanced Width (WX)	Character Name	Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})
U+0020	250	space	0, 0, 0, 0
U+0021	333	exclam	110, -15, 222, 677
U+0022	408	quotedbl	66, 392, 343, 677
U+0023	500	numbersign	18, -15, 482, 677
U+0024	500	dollar	53, -76, 452, 718
U+0025	833	percent	77, -15, 749, 677
U+0026	778	ampersand	35, -15, 764, 676
U+0027	180	quotesingle	41, 392, 139, 677
U+0028	333	parenleft	41, -216, 311, 694
U+0029	333	parenright	21, -216, 291, 694
U+002A	500	Asterisk	61, 263, 439, 677
U+002B	564	plus	19, 69, 545, 595
U+002C	250	comma	54, -167, 190, 93
U+002D	333	hyphen	40, 186, 293, 263
U+002E	250	period	71, -15, 179, 93
U+002F	278	slash	0, -14, 278, 694
U+0030	500	zero	37, -15, 463, 677
U+0031	500	one	90, 0, 405, 676
U+0032	500	two	34, 0, 449, 677
U+0033	500	three	24, -14, 461, 677
U+0034	500	four	15, 0, 474, 676
U+0035	500	five	40, -15, 451, 662
U+0036	500	six	36, -14, 468, 676
U+0037	500	seven	49, -14, 455, 662
U+0038	500	eight	31, -11, 466, 675
U+0039	500	nine	31, -13, 463, 678
U+003A	278	colon	85, -15, 193, 461

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Character Unicode	Advanced Width (WX)	Character Name	Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})
U+003B	278	semicolon	70, -167, 206, 461
U+003C	564	less	20, 91, 545, 573
U+003D	564	equal	20, 233, 545, 431
U+003E	564	greater	20, 91, 545, 573
U+003F	444	question	57, -15, 378, 677
U+0040	921	At	49, -214, 860, 677
U+0041	722	A	10, 0, 712, 677
U+0042	667	B	25, 0, 614, 662
U+0043	667	C	38, -15, 632, 677
U+0044	722	D	25, 0, 685, 662
U+0045	611	E	25, 0, 568, 662
U+0046	556	F	25, 0, 529, 662
U+0047	722	G	38, -15, 709, 677
U+0048	722	H	25, 0, 697, 662
U+0049	333	I	25, 0, 307, 662
U+004A	389	J	16, -15, 384, 662
U+004B	722	K	25, -9, 729, 662
U+004C	611	L	25, 0, 563, 662
U+004D	889	M	25, 0, 864, 662
U+004E	722	N	19, -11, 710, 662
U+004F	722	O	38, -15, 685, 676
U+0050	556	P	25, 0, 526, 662
U+0051	722	Q	38, -190, 745, 676
U+0052	667	R	25, -9, 675, 662
U+0053	556	S	61, -15, 504, 677
U+0054	611	T	35, 0, 575, 662
U+0055	722	U	9, -15, 712, 662
U+0056	722	V	10, -15, 711, 662
U+0057	944	W	12, -15, 934, 662
U+0058	722	X	9, 0, 712, 662
U+0059	722	Y	13, 0, 711, 662
U+005A	611	Z	28, 0, 572, 662
U+005B	333	bracketleft	81, -197, 297, 677
U+005C	278	backslash	0, -14, 278, 694
U+005D	333	bracketright	36, -198, 252, 677
U+005E	469	asciicircum	18, 324, 451, 676
U+005F	500	underscore	-8, -125, 508, -75
U+0060	333	grave	58, 513, 218, 658
U+0061	444	A	34, -14, 440, 471
U+0062	500	B	3, -14, 465, 694
U+0063	444	c	35, -15, 419, 471
U+0064	500	d	35, -14, 497, 694
U+0065	444	e	35, -15, 419, 471
U+0066	333	f	38, 0, 430, 694
U+0067	500	g	25, -213, 479, 471
U+0068	500	h	12, 0, 492, 694
U+0069	278	i	32, 0, 258, 677
U+006A	278	j	-69, -213, 196, 677
U+006B	500	k	13, 0, 506, 694
U+006C	278	l	33, 0, 248, 694
U+006D	778	m	12, 0, 766, 471
U+006E	500	n	12, 0, 487, 471
U+006F	500	o	34, -14, 466, 471
U+0070	500	p	3, -214, 465, 471
U+0071	500	q	35, -214, 489, 471

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Character Unicode	Advanced Width (WX)	Character Name	Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})
U+0072	333	r	13, 0, 337, 471
U+0073	389	s	38, -14, 359, 471
U+0074	278	t	10, -14, 279, 583
U+0075	500	u	10, -14, 495, 457
U+0076	500	v	4, -14, 491, 457
U+0077	722	w	8, -14, 714, 457
U+0078	500	x	10, 0, 489, 457
U+0079	500	y	4, -215, 491, 457
U+007A	444	z	18, 0, 419, 457
U+007B	480	braceleft	104, -216, 376, 694
U+007C	200	bar	80, -216, 120, 694
U+007D	480	braceright	104, -216, 376, 694
U+007E	541	Asciitilde	23, 252, 519, 412
U+00A1	333	exclamdown	110, -216, 222, 475
U+00A2	500	cent	58, -185, 442, 647
U+00A3	500	sterling	30, -15, 479, 677
U+00A4	500	currency	30, 112, 470, 552
U+00A5	500	yen	1, 0, 499, 662
U+00A6	200	brokenbar	80, -216, 120, 694
U+00A7	500	section	71, -191, 429, 678
U+00A8	333	dieresis	27, 536, 306, 644
U+00A9	760	copyright	34, -15, 726, 677
U+00AA	276	ordfeminine	2, 384, 295, 677
U+00AB	500	guillemotleft	37, 0, 463, 458
U+00AC	564	logicalnot	20, 233, 545, 431
U+00AE	760	registered	34, -15, 726, 677
U+00B0	400	degree	48, 379, 351, 676
U+00B1	564	plusminus	19, 69, 545, 595
U+00B2	300	twosuperior	9, 324, 255, 677
U+00B3	300	threesuperior	19, 316, 254, 677
U+00B4	333	Acute	115, 513, 275, 658
U+00B5	500	mu	10, -216, 495, 457
U+00B6	453	paragraph	6, -176, 442, 662
U+00B8	333	cedilla	75, -184, 257, 0
U+00B9	300	onesuperior	63, 324, 238, 677
U+00BA	310	ordmasculine	13, 383, 297, 677
U+00BB	500	guillemotright	37, 0, 463, 458
U+00BC	750	onequarter	63, -15, 735, 677
U+00BD	750	onehalf	63, -15, 716, 677
U+00BE	750	threequarters	19, -15, 734, 677
U+00BF	444	questiondown	66, -216, 387, 475
U+00C0	722	Agrave	10, 0, 712, 854
U+00C1	722	Aacute	10, 0, 712, 854
U+00C2	722	Acircumflex	10, 0, 712, 854
U+00C3	722	Atilde	10, 0, 712, 850
U+00C4	722	Adieresis	10, 0, 712, 840
U+00C5	722	Aring	10, 0, 712, 831
U+00C6	889	AE	-10, 0, 846, 662
U+00C7	667	Ccedilla	38, -184, 632, 677
U+00C8	611	Egrave	25, 0, 568, 854
U+00C9	611	Eacute	25, 0, 568, 854
U+00CA	611	Ecircumflex	25, 0, 568, 854
U+00CB	611	Edieresis	25, 0, 568, 840
U+00CC	333	Igrave	25, 0, 307, 854
U+00CD	333	Iacute	25, 0, 307, 854

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Character Unicode	Advanced Width (WX)	Character Name	Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})
U+00CE	333	Icircumflex	25, 0, 307, 854
U+00CF	333	Idieresis	25, 0, 307, 840
U+00D0	722	Eth	18, 0, 685, 662
U+00D1	722	Ntilde	19, -11, 710, 850
U+00D2	722	Ograve	38, -15, 685, 854
U+00D3	722	Oacute	38, -15, 685, 854
U+00D4	722	Ocircumflex	38, -15, 685, 854
U+00D5	722	Otilde	38, -15, 685, 850
U+00D6	722	Odieresis	38, -15, 685, 840
U+00D7	564	multiply	81, 131, 483, 533
U+00D8	722	Oslash	38, -23, 685, 683
U+00D9	722	Ugrave	9, -15, 712, 854
U+00DA	722	Uacute	9, -15, 712, 854
U+00DB	722	Ucircumflex	9, -15, 712, 854
U+00DC	722	Udieresis	9, -15, 712, 840
U+00DD	722	Yacute	13, 0, 711, 854
U+00DE	556	Thorn	25, 0, 526, 662
U+00DF	500	germandbls	20, -7, 466, 694
U+00E0	444	Agrave	34, -14, 440, 658
U+00E1	444	Aacute	34, -14, 440, 658
U+00E2	444	Acircumflex	34, -14, 440, 658
U+00E3	444	Atilde	34, -14, 440, 654
U+00E4	444	Adieresis	34, -14, 440, 644
U+00E5	444	Aring	34, -14, 440, 693
U+00E6	667	Ae	34, -14, 643, 471
U+00E7	444	ccedilla	35, -184, 419, 471
U+00E8	444	egrave	35, -15, 419, 658
U+00E9	444	eacute	35, -15, 419, 658
U+00EA	444	ecircumflex	35, -15, 419, 658
U+00EB	444	edieresis	35, -15, 419, 644
U+00EC	278	igrave	31, 0, 258, 658
U+00ED	278	iacute	32, 0, 258, 658
U+00EE	278	icircumflex	1, 0, 277, 658
U+00EF	278	idieresis	0, 0, 279, 644
U+00F0	500	eth	34, -14, 466, 694
U+00F1	500	ntilde	12, 0, 487, 654
U+00F2	500	ograve	34, -14, 466, 658
U+00F3	500	oacute	34, -14, 466, 658
U+00F4	500	ocircumflex	34, -14, 466, 658
U+00F5	500	otilde	34, -14, 466, 654
U+00F6	500	odieresis	34, -14, 466, 644
U+00F7	564	divide	20, 139, 545, 525
U+00F8	500	oslash	19, -37, 481, 494
U+00F9	500	ugrave	10, -14, 495, 658
U+00FA	500	uacute	10, -14, 495, 658
U+00FB	500	ucircumflex	10, -14, 495, 658
U+00FC	500	udieresis	10, -14, 495, 644
U+00FD	500	yacute	4, -215, 491, 658
U+00FE	500	thorn	3, -214, 465, 694
U+00FF	500	ydieresis	4, -215, 491, 644
U+0131	278	dotlessi	32, 0, 258, 471
U+0141	611	Lslash	13, 0, 563, 662
U+0142	278	lslash	2, 0, 283, 694
U+0152	889	OE	38, -15, 846, 676
U+0153	722	oe	34, -14, 697, 471

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Character Unicode	Advanced Width (WX)	Character Name	Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})
U+0160	556	Scaron	61, -15, 504, 877
U+0161	389	scaron	38, -14, 359, 681
U+0178	722	Ydieresis	13, 0, 711, 840
U+017D	611	Zcaron	28, 0, 572, 877
U+017E	444	zcaron	18, 0, 419, 681
U+0192	500	florin	-86, -206, 500, 694
U+02C6	333	circumflex	28, 513, 304, 658
U+02C7	333	caron	28, 536, 304, 681
U+02C9	333	macron	11, 559, 322, 622
U+02D8	333	breve	26, 521, 306, 658
U+02D9	333	dotaccent	113, 536, 221, 644
U+02DA	333	ring	73, 505, 261, 693
U+02DB	333	ogonek	72, -177, 287, 8
U+02DC	333	tilde	11, 526, 323, 654
U+02DD	333	hungarumlaut	46, 513, 323, 658
U+2013	500	endash	-9, 221, 509, 257
U+2014	1000	emdash	-9, 221, 1009, 257
U+2018	333	quoteleft	98, 417, 234, 677
U+2019	333	quoteright	98, 417, 234, 677
U+201A	333	quotesinglbase	98, -167, 234, 93
U+201C	444	quotedblleft	48, 417, 395, 677
U+201D	444	quotedblright	48, 417, 395, 677
U+201E	444	quotedblbase	48, -167, 395, 93
U+2020	500	dagger	49, -197, 451, 677
U+2021	500	daggerdbl	49, -216, 451, 677
U+2022	350	bullet	64, 220, 286, 442
U+2026	1000	ellipsis	112, -15, 888, 93
U+2030	1000	perthousand	31, -15, 969, 677
U+2039	333	guilsinglleft	57, 0, 276, 458
U+203A	333	guilsinglright	57, 0, 276, 458
U+20AC	500	Euro	-11, -15, 472, 677
U+2122	980	trademark	34, 265, 947, 662
U+2212	564	minus	19, 312, 545, 352
U+2215	167	fraction	-166, -15, 334, 677
U+2219	250	periodcentered	71, 277, 179, 385
U+FB01	556	fi	38, 0, 536, 694
U+FB02	556	fl	38, 0, 526, 694

7.12.3 Kerning Data

Kerning Pair	Value
Space A	-55
Space T	-18
Space V	-18
space W	-18
Space Y	-37
one one	-37
A space	-55
A T	-111
A V	-129
A W	-80
A Y	-92
A v	-74
A w	-92

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

Kerning Pair	Value
A y	-92
A quoteright	-111
F comma	-80
F period	-80
F A	-74
L space	-37
L T	-92
L V	-92
L W	-74
L Y	-100
L y	-55
L quoteright	-92
P space	-37
P comma	-111
P period	-111
P A	-92
R T	-60
R V	-80
R W	-55
R Y	-55
R y	-40
T space	-18
T comma	-74
T hyphen	-92
T period	-74
T colon	-50
T semicolon	-55
T A	-80
T O	-18
T a	-70
T c	-70
T e	-70
T i	-35
T o	-70
T r	-35
T s	-70
T u	-35
T w	-70
T y	-70
V space	-18
V comma	-129
V hyphen	-92
V period	-129
V colon	-74
V semicolon	-74
V A	-129
V a	-111
V e	-111
V i	-60
V o	-129
V r	-60
V u	-60
V y	-111
W space	-18
W comma	-92
W hyphen	-55

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Kerning Pair	Value
W period	-92
W colon	-37
W semicolon	-37
W A	-111
W a	-80
W e	-80
W i	-40
W o	-80
W r	-40
W u	-40
W y	-60
Y space	-37
Y comma	-129
Y hyphen	-111
Y period	-129
Y colon	-92
Y semicolon	-92
Y A	-111
Y a	-100
Y e	-100
Y i	-55
Y o	-100
Y p	-92
Y q	-111
Y u	-111
Y v	-100
f f	-18
f quoteright	55
r comma	-40
r hyphen	-20
r period	-55
r g	-18
r quoteright	37
v comma	-65
v period	-65
w comma	-65
w period	-65
y comma	-65
y period	-65
quoteleft quoteleft	-74
quoteright space	-74
quoteright s	-55
quoteright t	-18
quoteright quoteright	-74

7.13 Font metrics for TYPEWRITER font (Cumberland)

7.13.1 Global font information

The following table provides global font information:

Character Set	Basic Latin, Latin-1 Supplement
Weight	Regular
Italic Angle	0
Fixed Pitch	true

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Writing Direction	0 (horizontal)
Underline Position	-100
Underline Thickness	50
Cap Height	681
X height	462
Ascender	681
Descender	-184
Font Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})	-13, -184, 613, 932

7.13.2 Character metrics

Character Unicode	Advanced Width (WX)	Character Name	Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})
U+0020	600	space	0, 0, 0, 0
U+0021	600	exclam	243, -15, 358, 681
U+0022	600	quotedbl	146, 459, 454, 681
U+0023	600	numbersign	67, -16, 534, 697
U+0024	600	dollar	101, -42, 499, 697
U+0025	600	percent	36, -16, 563, 697
U+0026	600	ampersand	67, -16, 534, 648
U+0027	600	quotesingle	256, 459, 344, 681
U+0028	600	parenleft	278, -138, 468, 698
U+0029	600	parenright	132, -138, 323, 698
U+002A	600	asterisk	95, 291, 506, 681
U+002B	600	plus	77, 116, 524, 565
U+002C	600	comma	220, -147, 377, 118
U+002D	600	hyphen	91, 253, 509, 312
U+002E	600	period	227, -15, 377, 118
U+002F	600	slash	110, -27, 491, 697
U+0030	600	zero	80, -16, 521, 697
U+0031	600	one	115, 0, 519, 691
U+0032	600	two	76, 0, 492, 697
U+0033	600	three	66, -16, 507, 697
U+0034	600	four	56, 0, 555, 691
U+0035	600	five	66, -16, 508, 681
U+0036	600	six	88, -16, 519, 697
U+0037	600	seven	86, 0, 511, 681
U+0038	600	eight	84, -16, 518, 696
U+0039	600	nine	83, -15, 514, 697
U+003A	600	colon	225, -15, 375, 478
U+003B	600	semicolon	218, -147, 375, 478
U+003C	600	less	69, 92, 532, 587
U+003D	600	equal	76, 220, 523, 460
U+003E	600	greater	69, 92, 532, 587
U+003F	600	question	123, -15, 498, 696
U+0040	600	at	70, -96, 528, 697
U+0041	600	A	17, 0, 582, 681
U+0042	600	B	41, 0, 560, 681
U+0043	600	C	50, -15, 540, 697
U+0044	600	D	41, 0, 548, 681
U+0045	600	E	41, 0, 533, 681
U+0046	600	F	41, 0, 533, 681
U+0047	600	G	50, -15, 512, 697
U+0048	600	H	41, 0, 559, 681
U+0049	600	I	118, 0, 482, 681

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Character Unicode	Advanced Width (WX)	Character Name	Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})
U+004A	600	J	51, -15, 579, 681
U+004B	600	K	41, 0, 579, 681
U+004C	600	L	41, 0, 533, 681
U+004D	600	M	17, 0, 583, 681
U+004E	600	N	31, 0, 568, 681
U+004F	600	O	51, -15, 550, 697
U+0050	600	P	41, 0, 555, 681
U+0051	600	Q	51, -145, 550, 697
U+0052	600	R	41, 0, 579, 681
U+0053	600	S	74, -15, 526, 697
U+0054	600	T	57, 0, 543, 681
U+0055	600	U	31, -16, 569, 681
U+0056	600	V	17, 0, 582, 681
U+0057	600	W	10, 0, 586, 681
U+0058	600	X	26, 0, 573, 681
U+0059	600	Y	27, 0, 572, 681
U+005A	600	Z	47, 0, 550, 681
U+005B	600	bracketleft	249, -120, 462, 681
U+005C	600	backslash	109, -27, 490, 697
U+005D	600	bracketright	138, -120, 351, 681
U+005E	600	asciicircum	52, 298, 547, 681
U+005F	600	underscore	-2, -104, 602, -42
U+0060	600	grave	212, 542, 398, 719
U+0061	600	a	74, -16, 564, 478
U+0062	600	b	44, -16, 539, 681
U+0063	600	c	61, -16, 541, 478
U+0064	600	d	61, -16, 556, 681
U+0065	600	e	61, -16, 541, 478
U+0066	600	f	84, 0, 569, 697
U+0067	600	g	61, -184, 556, 478
U+0068	600	h	44, 0, 562, 681
U+0069	600	i	104, 0, 525, 681
U+006A	600	j	77, -177, 397, 681
U+006B	600	k	46, 0, 562, 681
U+006C	600	l	104, 0, 525, 681
U+006D	600	m	5, 0, 596, 478
U+006E	600	n	44, 0, 562, 478
U+006F	600	o	61, -16, 539, 478
U+0070	600	p	44, -184, 539, 478
U+0071	600	q	61, -184, 556, 478
U+0072	600	r	92, 0, 536, 478
U+0073	600	s	67, -16, 525, 478
U+0074	600	t	82, -16, 557, 622
U+0075	600	u	44, -16, 562, 462
U+0076	600	v	27, 0, 572, 462
U+0077	600	w	10, 0, 589, 462
U+0078	600	x	26, 0, 573, 462
U+0079	600	y	18, -184, 583, 462
U+007A	600	z	79, 0, 532, 462
U+007B	600	braceleft	188, -137, 411, 697
U+007C	600	bar	270, -127, 331, 697
U+007D	600	braceright	189, -137, 412, 697
U+007E	600	asciitilde	82, 202, 518, 368
U+00A1	600	exclamdown	241, -183, 356, 513
U+00A2	600	cent	61, -16, 541, 681

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Character Unicode	Advanced Width (WX)	Character Name	Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})
U+00A3	600	sterling	62, 0, 567, 697
U+00A4	600	currency	47, 55, 560, 629
U+00A5	600	yen	27, 0, 572, 681
U+00A6	600	brokenbar	270, -127, 331, 697
U+00A7	600	section	67, -133, 525, 697
U+00A8	600	dieresis	162, 544, 442, 630
U+00A9	600	copyright	2, -14, 599, 585
U+00AA	600	ordfeminine	161, 390, 465, 697
U+00AB	600	guillemotleft	106, 46, 493, 438
U+00AC	600	logicalnot	77, 148, 524, 427
U+00AE	600	registered	2, -14, 599, 585
U+00B0	600	degree	165, 427, 436, 696
U+00B1	600	plusminus	79, 0, 526, 565
U+00B2	600	twosuperior	164, 299, 417, 698
U+00B3	600	threesuperior	147, 289, 431, 698
U+00B4	600	acute	227, 542, 413, 719
U+00B5	600	mu	44, -148, 562, 462
U+00B6	600	paragraph	31, -124, 573, 681
U+00B8	600	cedilla	230, -183, 395, 7
U+00B9	600	onesuperior	179, 302, 424, 697
U+00BA	600	ordmasculine	151, 391, 453, 697
U+00BB	600	guillemotright	105, 46, 492, 438
U+00BC	600	onequarter	0, 0, 599, 697
U+00BD	600	onehalf	0, 0, 599, 697
U+00BE	600	threequarters	-3, 0, 599, 698
U+00BF	600	questiondown	117, -178, 492, 533
U+00C0	600	Agrave	17, 0, 582, 915
U+00C1	600	Aacute	17, 0, 582, 915
U+00C2	600	Acircumflex	17, 0, 582, 915
U+00C3	600	Atilde	17, 0, 582, 871
U+00C4	600	Adieresis	17, 0, 582, 826
U+00C5	600	Aring	17, 0, 582, 932
U+00C6	600	AE	0, 0, 594, 681
U+00C7	600	Ccedilla	53, -183, 543, 697
U+00C8	600	Egrave	41, 0, 533, 915
U+00C9	600	Eacute	41, 0, 533, 915
U+00CA	600	Ecircumflex	41, 0, 533, 915
U+00CB	600	Edieresis	41, 0, 533, 826
U+00CC	600	Igrave	118, 0, 482, 915
U+00CD	600	Iacute	118, 0, 482, 915
U+00CE	600	Icircumflex	118, 0, 482, 915
U+00CF	600	Idieresis	118, 0, 482, 826
U+00D0	600	Eth	15, 0, 548, 681
U+00D1	600	Ntilde	31, 0, 568, 871
U+00D2	600	Ograve	51, -15, 550, 915
U+00D3	600	Oacute	51, -15, 550, 915
U+00D4	600	Ocircumflex	51, -15, 550, 915
U+00D5	600	Otilde	51, -15, 550, 871
U+00D6	600	Odieresis	51, -15, 550, 826
U+00D7	600	multiply	98, 138, 502, 542
U+00D8	600	Oslash	52, -27, 551, 698
U+00D9	600	Ugrave	31, -16, 569, 915
U+00DA	600	Uacute	31, -16, 569, 915
U+00DB	600	Ucircumflex	31, -16, 569, 915
U+00DC	600	Udieresis	31, -16, 569, 826

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Character Unicode	Advanced Width (WX)	Character Name	Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})
U+00DD	600	Yacute	27, 0, 572, 915
U+00DE	600	Thorn	41, 0, 550, 681
U+00DF	600	germandbls	44, -16, 565, 697
U+00E0	600	agrave	74, -16, 564, 719
U+00E1	600	aacute	74, -16, 564, 719
U+00E2	600	acircumflex	74, -16, 564, 719
U+00E3	600	atilde	74, -16, 564, 675
U+00E4	600	adieresis	74, -16, 564, 630
U+00E5	600	aring	74, -16, 564, 738
U+00E6	600	ae	11, -16, 596, 478
U+00E7	600	ccedilla	61, -183, 541, 478
U+00E8	600	egrave	61, -16, 541, 719
U+00E9	600	eacute	61, -16, 541, 719
U+00EA	600	ecircumflex	61, -16, 541, 719
U+00EB	600	edieresis	61, -16, 541, 630
U+00EC	600	igrave	104, 0, 525, 719
U+00ED	600	iacute	104, 0, 525, 719
U+00EE	600	icircumflex	104, 0, 525, 719
U+00EF	600	idieresis	104, 0, 525, 630
U+00F0	600	eth	61, -16, 539, 702
U+00F1	600	ntilde	44, 0, 562, 675
U+00F2	600	ograve	61, -16, 539, 719
U+00F3	600	oacute	61, -16, 539, 719
U+00F4	600	ocircumflex	61, -16, 539, 719
U+00F5	600	otilde	61, -16, 539, 675
U+00F6	600	odieresis	61, -16, 539, 630
U+00F7	600	divide	91, 111, 509, 569
U+00F8	600	oslash	35, -33, 562, 496
U+00F9	600	ugrave	44, -16, 562, 719
U+00FA	600	uacute	44, -16, 562, 719
U+00FB	600	ucircumflex	44, -16, 562, 719
U+00FC	600	udieresis	44, -16, 562, 630
U+00FD	600	yacute	18, -184, 583, 719
U+00FE	600	thorn	44, -184, 539, 681
U+00FF	600	ydieresis	18, -184, 583, 630
U+0131	600	dotlessi	104, 0, 525, 462
U+0141	600	Lslash	7, 0, 533, 681
U+0142	600	lslash	104, 0, 525, 681
U+0152	600	OE	13, 0, 594, 681
U+0153	600	oe	11, -16, 596, 478
U+0160	600	Scaron	74, -15, 526, 916
U+0161	600	scaron	67, -16, 525, 720
U+0178	600	Ydieresis	27, 0, 572, 826
U+017D	600	Zcaron	47, 0, 550, 916
U+017E	600	zcaron	79, 0, 532, 720
U+0192	600	florin	17, -159, 589, 697
U+02C6	600	circumflex	125, 542, 477, 719
U+02C7	600	caron	125, 543, 477, 720
U+02C9	600	macron	152, 543, 450, 602
U+02D8	600	breve	125, 543, 475, 720
U+02D9	600	dotaccent	269, 575, 340, 681
U+02DA	600	ring	202, 544, 398, 738
U+02DB	600	ogonek	266, -161, 440, 2
U+02DC	600	tilde	114, 544, 486, 675
U+02DD	600	hungarumlaut	118, 542, 482, 719

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Character Unicode	Advanced Width (WX)	Character Name	Bounding Box (X_{min}, Y_{min}, X_{max}, Y_{max})
U+2013	600	endash	91, 253, 509, 312
U+2014	600	emdash	-3, 253, 599, 312
U+2018	600	quoteleft	234, 433, 391, 698
U+2019	600	quoteright	209, 432, 366, 697
U+201A	600	quotesinglbase	220, -147, 377, 118
U+201C	600	quotedblleft	113, 433, 516, 698
U+201D	600	quotedblright	113, 432, 516, 697
U+201E	600	quotedblbase	113, -147, 516, 118
U+2020	600	dagger	109, -108, 490, 681
U+2021	600	daggerdbl	109, -108, 490, 681
U+2022	600	bullet	179, 190, 421, 432
U+2026	600	ellipsis	42, -8, 558, 108
U+2030	600	perthousand	17, -16, 600, 697
U+2039	600	guilsinglleft	207, 46, 394, 438
U+203A	600	guilsinglright	206, 46, 393, 438
U+20AC	600	Euro	0, -15, 540, 587
U+2122	600	trademark	-13, 259, 613, 681
U+2212	600	minus	75, 309, 522, 371
U+2215	600	fraction	79, 230, 513, 515
U+2219	600	periodcentered	227, 244, 377, 377
U+FB01	600	fi	8, 0, 598, 697
U+FB02	600	fl	8, 0, 598, 697

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8 BIFS

8.1 Introduction

BIFS data consists of two distinct elements in the multiplexed bitstream. Terminal configuration information is first sent in the object descriptor. The remaining BIFS information is sent in a separate elementary stream.

The syntax and semantics of the terminal configuration is described in 8.5.2 and 8.5.3. Two different kinds of session can take place: a BIFS-Command session or a BIFS-Anim session.

If the session is a BIFS-Command session, a sequence of commands to modify the scene is sent. The syntax and semantics of these commands are described in 8.6.

If the session is a BIFS-Anim session, a sequence of animation data to change the values of specific fields in the scene is sent. The syntax and semantics of this session is described in 8.8.

8.2 Decoding tables, data structures and associated functions

8.2.1 Function of decoding tables, data structures and functions

This subclause describes tables and data structures used to contain necessary data, along with the associated functions, for decoding the BIFS elementary streams. These are not syntax elements but are descriptions, often in code or pseudo-code, of data and functions that are required to decode the bitstream. The tables and data structures may be known a priori at the terminal or may be constructed from data parsed from the bitstream. They are referenced throughout the syntax.

NOTE — The code or pseudo-code for the non-syntax data elements is purely notational and does not imply a normative requirement to use these code fragments in implementations.

Coding of individual nodes and field values is very regular, and follows a depth-first order (children or sub-nodes of a node are present in the bitstream before its siblings).

8.2.2 Node Data Type Tables

Identification of nodes and fields within a BIFS scene graph is context-dependent. Each field of a BIFS node that accepts nodes as fields can only accept a specific set of nodes. Each of these sets of nodes is stored in a node data type table and is referenced by a node data type (NDT).

A field of type SFNode is fully described by its NDT. Each node belongs to one or more NDT tables. These tables are provided in node coding tables in electronic attachment and identify the various nodes and node types they contain.

Identification of a particular node depends on the context of the NDT specified for its parent field. The node data types are listed in node coding tables in electronic attachment, and group 2 node data types in node coding tables in electronic attachment. For each node, the value zero (encoded with as many bits as required to encode the total number of nodes in that NDT table in node coding tables in electronic attachment.) is used before the actual node type to indicate that the node is of an extended node type. The value 0 in each extended NDT table is reserved for future extensions. Value one in each extended NDT table is reserved for encoding of PROTOs (see 8.7.2).

EXAMPLE 1 — **Anchor** is identified by the 5-bit code 0b0000.1 when the context of its parent's field is SF2DNode, whereas the 7-bit code 0b0000.001 is used when the context of its parent's field is SFWorldNode.

EXAMPLE 2 — **AcousticScene** is identified by a 3-bit code 0b010, when the context of its parent field is SF3DNode in the group 2 node data types in node coding tables in electronic attachment. Since that NDT exists in node coding tables in electronic attachment. (where the nodes of that data type are encoded with six bits), this node is completely encoded with 9 bits as: 0b000000010.

8.2.3 Node Coding Tables and field indexing

The syntactic description of fields is context-dependent. For a given node, its fields are indexed using a code called a `fieldID`. This `fieldID` is not unique for each field of a node but varies according to the "mode" in which the field is referenced. There are five modes in which a field may be referenced and, thus, five types of `fieldID`. For each field of each node, the binary values of the `fieldIDs` for each mode are defined in the node coding tables.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

defID
The defIDs refer to the fieldIDs for those fields that may have a value when nodes are declared. They refer to fields of type exposedField and field. This indexing scheme is further referred to as the “def” mode.

inID
The inIDs refer to the fieldIDs for those events and fields that can be modified from outside the node. They refer to fields of type exposedField and eventIn types. This indexing scheme is further referred to as the “in” mode.

outID
The outIDs refer to the fieldIDs for those events and fields that can be output from the node. They refer to fields of type exposedField and eventOut types. This indexing scheme is further referred to as the “out” mode.

dynID
The dynIDs refer to the fieldIDs for those fields that can be animated using the BIFS-Anim scheme. They refer to a subset of the fields designated by inIDs. This indexing scheme is further referred to as the “dyn” mode.

allID
The allIDs refer to all events and fields of the node. That is, there is an allID for each field of a node. This indexing scheme is further referred to as the “all” mode.

The length of each of the fieldID types for each node depends on the number of fields of that type for the given node.

EXAMPLE — The **AnimationStream** node has four fields of type defID. Therefore, three bits are required to code the defIDs for this node. The **Appearance node**, however, has just three fields of type defID. Therefore, two bits are sufficient to code the defIDs for this node.

8.2.4 BIFSConfig

This data structure is a global data structure referred to in every BIFS access unit. The data contained in the BIFSConfiguration data structure is transmitted in either BIFSConfig or BIFSv2Config (see 8.5.2 and 8.5.3).

```
Class BIFSConfiguration{
   int nodeIDbits;
   int routeIDbit;
   int PROTOIDbits;
   boolean randomAccess;
   AnimationMask animMask;
}
```

The number of bits used to encode the nodeIDs.

The number of bits used to encode the routeIDs.

The number of bits used to encode the protoID. This value is in used only if the data for the structure was transmitted by BIFSv2Config

The randomAccess boolean is set in the BIFSConfig to distinguish between BIFS-Anim elementary streams in which support random access at any intra frame, and those where random access may not be possible at all intra frames. In the latter case, greater compression efficiency may be achieved because a given intra frame may re-use quantization settings and statistics from the previous intra frame.

The AnimationMask used for BIFS-Anim

8.2.5 AnimationMask

The AnimationMask structure contains all the relevant information to describe a BIFS-Anim session. It is constructed, upon receipt of the BIFSConfig or BIFSv2Config syntax element, during the configuration of the BIFS decoder, and updated for every received AnimationFrame.

```
Class AnimationMask {
   int numNodes;
   NodeData animNode[numNodes];
   boolean isIntra;
   boolean isActive[numNodes];
}
```

The number of nodes to be animated

The array of animated nodes.

The status of the current frame: intra if isIntra is true, predictive otherwise.

The mask of active animated node for the current frame. If the node is not animated in the current frame, the boolean shall be false.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.6 NodeData

This data structure is built to decode the relevant information for one node. It is created from the node coding tables in electronic attachment. The following functions support relevant operations on this data structure:

`NodeData MakeNode(nodeGroup, nodeDataType, int nodeType)`

This function creates a `NodeData` structure from the node coding table in the node grouping nodeGroup for the given nodeDataType matching the given `nodeType`.

`NodeData MakePROTO(PROTOinterfaceDefinition interface)`

This function creates a NodeData structure from the given PROTOinterfaceDefinition. This NodeData structure is recursively forwarded to all the nodes in the PROTOcode to encode the ISed fields.

`NodeData GetNodeFromID (int nodeID)`

This function returns the `NodeData` structure matching the given `nodeID`.

`class NodeData {`

`int nodeType;`	The `nodeType` of the node.
`FieldData field[];`	The fields of this node whose construction is described below. This array is indexed in “all” mode.
`boolean isAnimField[];`	The mask of animated fields for the entire BIFS-Anim session, indexed in “dyn” mode. This array is only used in BIFS-Anim.
	The following data describes the indexing of the fields in “in”, “out”, “def”, “dyn” and “all” modes
`int nDEFbits;`	The number of bits used for “def” field codes (the width of the codewords in the 3rd column of the node coding tables).
`int nINbits;`	The number of bits used for “in” field codes (the width of the codewords in the 4th column of the node coding tables).
`int nOUTbits;`	The number of bits used for “out” field codes (the width of the codewords in the 5th column of the node coding tables).
`int numDEFfields;`	The number of “def” fields available for this node
`int numDYNfields;`	The number of “dyn” fields available for this node.
`int in2all[];`	The ids of eventIns and exposedFields in “all” mode, indexed with the ids in “in” mode.
`int def2all[];`	The ids of fields and exposedFields in “all” mode, indexed with the ids in “def” mode.
`int dyn2all[];`	The ids of dynamic fields in “all” mode, indexed with the ids in “dyn” mode.
`boolean useQuant;`	When the NodeData is used for storing a prototype, the useQuant states whether the quantization is applied on the PROTO or not.
`boolean useAnim;`	When the NodeData is used for storing a prototype, the useAnim states whether the BIFS-Anim is applied on the PROTO or not.
`NodeData proto;`	In case that a node is contained in a PROTO, its NodeData structure points to the PROTO NodeData structure in the proto field.
`int nALLbits;`	The number of bits to encode all the fields of the node. i.e. the smallest value such that $2^{nALLbits}$ > number of fields.

`}`

8.2.7 FieldData

This data structure is built to decode the relevant information for one field. It is created from the field’s entry in the relevant node coding table (see node coding tables in electronic attachment).

`Class FieldData {`

`int fieldType;`	The type of the field (e.g., `SFInt32Type`). This is given by the “Field Type” column of the node coding table for the node to which it belongs.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

`int quantType;`	The type of quantization used for the field. This is given by the "Q" column of the node coding table of the node to which it belongs. Types refer to Table 72 in 8.3.1.1.
`int animType;`	The animation method for the field. This is given by the "A" column of the node coding table. Types refer to animation type in Table 78 in 8.3.2.1.
`boolean useEfficientCoding;`	Set to true if the efficient coding is to be used. This value is FALSE by default. If there is a local **QuantizationParameter** node this value is the same as its **useEfficientCoding** field.
	The following data structures are used in the quantization process:
`FieldCodingTable fct;`	This field is determined from the node coding table as described in 8.2.9.
`AnimFieldQP aqp;`	This field is only used in BIFS-Anim. It references an `AnimFieldQP` stucture described in 8.2.10.
`QuantizationParameter lqp;`	This field points to the local **QuantizationParameter** node.
`boolean isQuantized;`	Set to true if the corresponding field is quantized, false otherwise.
`int nbBits;`	The number of bits used for the quantization of the field.
`float floatMin[];`	The minimum bounds for the quantization of vector fields. These values are obtained from the `FieldCodingTable` (described in 8.2.9) and the current **QuantizationParameter** node (for BIFS-Scene) or the `animField` (for BIFS-Anim).
`float floatMax[];`	The maximum bounds for the quantization of vector fields. These values are obtained from the `FieldCodingTable` (described in 8.2.9) and the current **QuantizationParameter** node (for BIFS-Scene) or the `animField` (for BIFS-Anim).
`int intMin[];`	The minimum bounds for integers (SFInt32 and MFInt32). These values are obtained from the `FieldCodingTable` (described in 8.2.9) and the current **QuantizationParameter** node (for BIFS-Scene) or the `animField` (for BIFS-Anim).

`}`

It is assumed that the following functions are available:

`int isSF(FieldData field)`

This function returns TRUE if the field's `fieldType` corresponds to a single field and FALSE otherwise.

`int isDEF(FieldData field)`

This function returns TRUE if the field has a DEF ID (i.e. if the field is of scope `exposedField` or `eventIn`).

`FieldData MakeField(int fieldType)`

This function creates a `FieldData` structure from the given `fieldType`.

`int getNbComp(FieldData field)`

Returns the number of quantized components for the field as given below:

Table 70 — Return values of `getNbComp`

fieldType	quantType	animType	value returned
SFFloat SFInt32	any	6,7,8 13	1
SFVec2f SFVec3f	any 9	2,12 9	2
SFVec3f SFRotation	!=9 any	1,4,11 10	3

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

The number of quantized components is the same as the natural number of components (three for SFVec3f, two for SFVec2f, and so on) except for normals (2) and rotations (3) because of the quantization process (see 8.3.3).

8.2.8 Node Data Type Table Parameters

The following functions provide access to the node data type tables (described in node coding tables in electronic attachment):

Nodes are organized into groups where new nodes and node types, may be added as a new group in a future revisions of this standard. An extension mechanism is defined. See 8.2.2 and 8.7.3.2, for a discussion of how new groups are added.

The Groups are defined in node coding tables in electronic attachment.

The nodeGroup index starts at 1 for Basic Nodes, and increments for each new group added.

```
int GetNodeType(int nodeGroup, int nodeDataType, int localNodeType)
```

Returns the nodeType of the node indexed by localNodeType in the node data type table for the group of nodes specified by nodeGroup. The nodeType of a node is its index in the SFWorldNode NDT Table for that group.

```
int GetNDTnbBits(int nodeGroup, int nodeDataType)
```

Returns the number of bits used to index the nodes of the matching node data type table for that nodeGroup (this number is indicated in the last column of the first row of the node data type table).

```
int GetNDTFromID(int id)
```

Returns the `nodeDataType` for the **children** field of the node identified by the `nodeID`, `id`. Nodes having a **children** field may have restrictions on the types of node that may occupy the field. These node types are indicated in the node semantics (see 1 and ISO/IEC 14772-1:1998 , Table 4.3).

8.2.9 Field Coding Table

This data structure contains parameters relating to the quantization of the field. It is created from the field's entry in the relevant node coding tables in electronic attachment.

```
Class FieldCodingTable {
```

`float floatMin[];`	The minimum default bounds for fields of type SFFloat, SFVec2f, SFColor and SFVec3f. These values are obtained from the "[m, M]" column of the node coding table or the `InterfaceCodingParameters` in the case of a field in a PROTO.
`float floatMax[];`	The maximum default bounds for fields of type SFFloat, SFVec2f, SFColor and SFVec3f. These values are obtained from the "[m, M]" column of the node coding table or the `InterfaceCodingParameters` in the case of a field in a PROTO.
`float intMin[];`	The minimum default bounds for fields of type SFInt32. These values are obtained from the "[m, M]" column of the node coding table.
`float intMax[];`	The maximum default bounds for fields of type SFInt32. These values are obtained from the "[m, M]" column of the node coding table.
`int defaultNbBits;`	The number of bits used by default for each field. Only used when the quantization category of the field is 13. For quantization category 13, the number of bits used for coding is also specified in the node coding (e.g "13 16" in the node coding table means category 13 with 16 bits).

```
}
```

8.2.10 AnimFieldQP

This data structure contains the necessary quantization parameters and information for the animation of a field. It is updated throughout the BIFS-Anim session.

```
class AnimFieldQP {
```

`int animType;`	The animation method for the field. This is given by the "A" column of the node coding table for each node. Types refer to animation type in Table 72 in 8.3.2.1.
`boolean useDefault;`	If this bit is set to TRUE, then the bounds used in intra mode are those specified in the "[m, M]" column of the node coding table. The default value is FALSE.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

`boolean isTotal;`	If the field is a multiple field and if this boolean is set to TRUE, all the components of the multiple field are animated.
`int numElement;`	The number of elements being animated in the field. This is 1 for all single fields, and equal to or greater than 1 for multiple fields.
`int indexList[];`	If the field is a multiple field and if `isTotal` is false, this is the list of the indices of the animated `SFFields`. For instance, if the field is an `MFField` with elements 3,4 and 7 being animated, the valuse of `indexList` will be {3,4,7}.
`float[] Imin;`	The minimum values for bounds of the field in intra mode. This value is obtained from the "[m, M]" column of the node coding table (if `useDefault` is TRUE), the `InitialAnimQP` (if `useDefault` is FALSE and the last intra did not hold any new `AnimQP`), or the `AnimQP`.
`float[] Imax;`	The maximum values for bounds of the field in intra mode. This value is obtained from the "[m, M]" column of the semantics table (if `useDefault` is TRUE), the `InitialAnimQP` (if `useDefault` is FALSE and if the last intra did not hold any new `AnimQP`), or the `AnimQP`.
`int[] IminInt;`	The minimum value for bounds of variations of integer fields in intra mode. This value is obtained from the `InitialAnimQP` (if the last intra did not hold any new `AnimQP`) or `AnimQP` structure.
`int[] Pmin;`	The minimum value for bounds of variations of the field in predictive mode. This value is obtained from the `InitialAnimQP` (if the last intra did not hold any new `AnimQP`) or `AnimQP`.
`int INbBits;`	The number of bits used in intra mode for the field. This value is obtained from the `InitialAnimQP` or `AnimQP`.
`int PNbBits;`	The number of bits used in predictive mode for the field. This value is obtained from the `InitialAnimQP` (if the last intra did not hold any new `AnimQP`) or `AnimQP` structure.

```
}
```

It is assumed that the following function is available :

`int getNbBounds(int type)`

Returns the number of set of bounds matching the quantization type or animation type (see 8.2.7), as follows :

Table 71 — Return values of getNbBounds

type	value returned
type>2	1
2	2
1	3

Note that only `Position2D` and `Position3D` have specific sets of bounds for each of their components. The number of bounds is also the number of independent models used in predictive mode during the BIFS-Anim session.

8.3 Quantization

In BIFS scenes, the values of the fields may be quantized. BIFS-Anim data is always quantized. This subclause describes this quantization process. A number of parameters control the quantization of a field. Here, these parameters are used to construct a notational data structure called `FieldData`. In this subclause, the semantics of how to determine these parameters for BIFS scenes and BIFS-Anim are first described, followed by a description of the actual quantization process.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.1 Quantization of BIFS scenes

8.3.1.1 Quantization categories

Single fields are coded according to the type of the field. The fields have a default syntax that specifies a non-quantized encoding. When quantization is used, the quantization parameters are obtained from a special node called **QuantizationParameter**. The following quantization categories are specified, providing suitable quantization procedures for the various types of quantities represented by the various fields of the BIFS nodes.

Table 72 — Quantization Categories

Category	Description
0	None
1	3D position
2	2D positions
3	Drawing order
4	SFColor
5	Texture Coordinate
6	Angle
7	Scale
8	Interpolator keys
9	Normals
10	Rotations
11	Object Size 3D (1)
12	Object Size 2D (2)
13	Linear Scalar Quantization
14	CoordIndex
15	SFVec4f
16	Reserved

Each field that may be quantized is assigned to one of the quantization categories (see node coding tables in electronic attachment). Along with quantization parameters, minimum and maximum values are specified for each field of each node.

To quantize an SFVec4f field of value (x, y, z, w), the **position3D** and **scale** quantizers of the QuantizationParameter node are used at the same time. Specifically, **position3DQuant** and **scaleQuant** should be set to TRUE. **position3DMin**, **position3DMax**, and **position3DNbBits** are the quantization parameters used to quantize the 3D part of the vector (*x*, *y*, *z*) and **scaleMin**, **scaleMax**, and **scaleNbBits** are the quantization parameters used to quantize the scaling factor *w*.

As two quantizers are used, they also apply to fields using position3D and scale quantizers. As a result, if a node using SFVec4f quantization is followed by a node using SFVec3f and scale fields, these two types of fields will be quantized using the quantization parameters for the node using SFVec4f fields. Otherwise, two QuantizationParameter nodes should be used: one for the node using SFVec4f fields and one for the node using SFVec3f and scale fields.

BIFS-Anim uses quantization of SFVec4f fields the same way it is described above for BIFS-Commands.

SFVec4f fields are only used in solid modeling detailed later in this specification that needs 4D geometry. These fields are not used in previous versions of MPEG-4 (up to amendment 3). As a result, BIFS decoders not supporting amendment 4 and above do not need to implement SFVec4f decoding.

8.3.1.2 Determining the quantization parameters for a given field

The scope of quantization is constrained to a single BIFS access unit. A field is quantized when:

The field is of type SFInt32, SFFloat, SFRotation, SFColor, SFVec2f, SFVec3f, MFInt32, MFFloat, SFRotation, MFColor, MFVec2f, or MFVec3f.

The quantization category of the field is not 0.

The node to which the field belongs has a **QuantizationParameter** (see 7.2.2.108) node in its context

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The quantization for this type of field is activated (by setting the corresponding boolean to TRUE in the **QuantizationParameter** node.

The `isQuantized`, `nbBits`, `floatMin`, `floatMax` and `intMin` fields of the `FieldData` structure pertain to the quantization of the field. The values of these fields are determined from the local **QuantizationParameter** (`lqp`) and the `FieldCodingTable` (`fct`) stored in the `FieldData`. This is done in the following way:

isQuantized

`isQuantized` is set to true when the three following conditions are met :

`lqp!=0` (there is a **QuantizationParameter** node in the scope of the field)

`quantType !=0` (the field value is of a type that may be quantized), and

the following condition is met for the relevant quantization type:

Table 73 — Condition for setting `isQuantized` to true

quantType	Condition
1	lqp.position3DQuant == TRUE
2	lqp.position2DQuant == TRUE
3	lqp.drawOrderQuant == TRUE
4	lqp.colorQuant == TRUE
5	lqp.textureCoordinateQuant == TRUE
6	lqp.angleQuant == TRUE
7	lqp.scaleQuant == TRUE
8	lqp.keyQuant == TRUE
9	lqp.normalQuant == TRUE
10	lqp.normalQuant == TRUE
11	lqp.sizeQuant == TRUE
12	lqp.sizeQuant == TRUE
13	Always TRUE
14	Always TRUE
15	lqp. position3Dquant = TRUE lqp.scaleQuant = TRUE
16	Always TRUE

nbBits

In the BIFS scene quantization process, `nbBits` is set in the following way :

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 74 — Value of `nbBits` depending on `quantType`

quantType	**nbBits**
1	lqp.position3DNbBits
2	lqp.position2DNbBits
3	lqp.drawOrderNbBits
4	lqp.colorNbBits
5	lqp.textureCoordinateNbBits
6	lqp.angleNbBits
7	lqp.scaleNbBits
8	lqp.keyNbBits
9,10	lqp.normalNbBits
11,12	lqp.sizeNbBits
13	fct.defaultNbBits
14	This value is set according to the number of points received in the last received coord field of the node. Let N that number, then: $\mathrm{nbBits} = \mathrm{Ceil}(\log_2(N+1))$ where the function Ceil returns the smallest integer greater than its argument
15	lqp.position3DnbBits lqp.scaleNbBits
16	0

`floatMin[]`

In the BIFS scene quantization process, `floatMin` is set in the following way:

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 75 — Value of `floatMin`, depending on `quantType` and `fieldType`

quantType	FieldType	floatMin
1	SFVec3fType	max(lqp.position3Dmin[0],fct.floatMin[0]), max(lqp.position3Dmin[1],fct.floatMin[1]), max(lqp.position3Dmin[2],fct.floatMin[2])
2	SFVec2fType	max(lqp.position2Dmin[0],fct.floatMin[0]), max(lqp.position2Dmin[1],fct.floatMin[1])
3	SFFloatType	max(fct.floatMin[0],lqp.drawOrderMin)
4	SFFloatType	max(lqp.colorMin, fct.floatMin[0])
	SFColorType	max(lqp.colorMin, fct.floatMin[0]), max(lqp.colorMin, fct.floatMin[0]), max(lqp.colorMin, fct.floatMin[0])
5	SFVec2fType	max(fct.floatMin[0],lqp.textureCoordinateMin)
6	SFFloatType	Max(fct.floatMin[0],lqp.angleMin)
7	SFFloatType	max(fct.floatMin[0],lqp.scaleMin)
	SFVec2fType	max(fct.floatMin[0],lqp.scaleMin), max(fct.floatMin[0],lqp.scaleMin)
	SFVec3fType	max(fct.floatMin[0],lqp.scaleMin), max(fct.floatMin[0],lqp.scaleMin), max(fct.floatMin[0],lqp.scaleMin)
8	SFFloatType	Max(fct.floatMin[0],lqp.keyMin)
9	SFVec3fType	0.0
10	SFRotationType	0.0
11,12	SFFloatType	max(fct.floatMin[0],lqp.sizeMin)
	SFVec2fType	max(fct.floatMin[0],lqp.sizeMin), max(fct.floatMin[0],lqp.sizeMin)
	SFVec3fType	max(fct.floatMin[0],lqp.sizeMin), max(fct.floatMin[0],lqp.sizeMin), max(fct.floatMin[0],lqp.sizeMin)
13,14		NULL
15	SFVec4fType	lqp.position3Dmin lqp.scaleMin
16		NULL

`floatMax[]`

In the BIFS scene quantization process, `floatMax` is set in the following way:

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 76 — Value of `floatMax`, depending on `quantType` and `fieldType`

quantType	fieldType	FloatMax
1	SFVec3fType	min(lqp.position3Dmax[0],fct.floatMax[0]), min(lqp.position3Dmax[1],fct.floatMax[1]), min(lqp.position3Dmax[2],fct.floatMax[2])
2	SFVec2fType	min(lqp.position2Dmax[0],fct.floatMax[0]), min(lqp.position2Dmax[1],fct.floatMax[1])
3	SFFloatType	min(fct.floatMax[0],lqp.drawOrderMax)
4	SFFloatType	min(lqp.colorMax, fct.floatMax[0])
	SFColorType	min(lqp.colorMax, fct.floatMax[0]), min(lqp.colorMax, fct.floatMax[0]), min(lqp.colorMax, fct.floatMax[0])
5	SFVec2fType	min(fct.floatMax[0],lqp.textureCoordinateMax)
6	SFFloatType	Min(fct.floatMax[0],lqp.angleMax)
7	SFFloatType	min(fct.floatMax[0],lqp.scaleMax)
	SFVec2fType	min(fct.floatMax[0],lqp.scaleMax), min(fct.floatMax[0],lqp.scaleMax)
	SFVec3fType	min(fct.floatMax[0],lqp.scaleMax), min(fct.floatMax[0],lqp.scaleMax), min(fct.floatMax[0],lqp.scaleMax)
8	SFFloatType	Min(fct.floatMax[0],lqp.keyMax)
9	SFVec3fType	0.0
10	SFRotationType	0.0
11,12	SFFloatType	min(fct.floatMax[0],lqp.sizeMax)
	SFVec2fType	min(fct.floatMax[0],lqp.sizeMax), min(fct.floatMax[0],lqp.sizeMax)
	SFVec3fType	min(fct.floatMax[0],lqp.sizeMax), min(fct.floatMax[0],lqp.sizeMax), min(fct.floatMax[0],lqp.sizeMax)
13,14		NULL
15	SFVec4fType	lqp.position3Dmax lqp.scaleMax
16		NULL

`intMin[]`

In the BIFS scene quantization process, `intMin` is set in the following way:

Table 77 — Value of `intMin`, depending on `quantType`

quantType	intMin
1,2,3,4,5,6,7,8 9,10,11,12	NULL
13,14	fct.intMin[0]
15	NULL

8.3.2 Quantization of BIFS-Anim

8.3.2.1 Animation Categories

The fields are grouped in the following categories for animation:

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 78 — Animation Categories

Category	Description
0	None
1	Position 3D
2	Positions 2D
3	Reserved
4	Color
5	Reserved
6	Angle
7	Float
8	BoundFloat
9	Normals
10	Rotation
11	Size 3D
12	Size 2D
13	Integer
14	Reserved
15	Reserved

8.3.2.2 Determining the quantization parameters for a given field

The `isQuantized`, `nbBits`, `floatMin`, `floatMax` and `intMin` fields of the `FieldData` structure pertain to the quantization of the field. The values of these fields are determined from the local `AnimFieldQP` (`aqp`) and the `FieldCodingTable` (`fct`) stored in the `FieldData`. This is done in the following way:

`isQuantized`

In the BIFS-Anim quantization process, `isQuantized` is always TRUE.

`nbBits`

In the BIFS-Anim quantization process, `nbBits` is set in the following way :

Table 79 — Value of `nbBits`, depending on `animType`

`animType`	**`nbBits`**
`1,2,4,6,7,8,9 10,11,12,13`	`animType.INbBits`

`floatMin[]`

In the BIFS-Anim quantization process, `floatMin` is set in the following way:

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 80 — Value of `floatMin`, depending on `animType`

animType		**aqp.useDefault**	**floatMin**
4	Color	true	fct.floatMin[0], fct.floatMin[0], fct.floatMin[0]
		false	aqp.IMin[0], aqp.IMin[0], aqp.IMin[0]
8	BoundFloat	true	fct.floatMin[0]
		false	aqp.IMin[0]
1	Position 3D	false	aqp.Imin
2	Position 2D	false	aqp.Imin
11	Size 3D	false	aqp.IMin[0], aqp.IMin[0], aqp.IMin[0]
12	Size 2D	false	aqp.IMin[0], aqp.IMin[0]
7	Float	false	aqp.IMin[0]
6 9 10	Angle Normal Rotation	false	0.0
13	Integer	false	NULL
14,15	Reseved		NULL

`floatMax[]`

In the BIFS-Anim quantization process, `floatMax` is set in the following way:

Table 81 — Value of `floatMax`, depending on `animType`

animType		**aqp.useDefault**	**FloatMax**
4	Color	true	fct.floatMax[0], fct.floatMax[0], fct.floatMax[0]
		false	aqp.IMax[0], aqp.IMax[0], aqp.IMax[0]
8	BoundFloat	true	fct.max[0]
		false	aqp.IMax[0]
1	Position 3D	false	aqp.Imax
2	Position 2D	false	aqp.Imax
11	Size 3D	false	aqp.IMax[0],aqp.IMax[0] ,aqp.IMax[0]
12	Size 2D	false	aqp.IMax[0], aqp.IMax[0]
7	Float	false	aqp.IMax[0]
6	Angle	false	2*Pi
9 10	Normal Rotation	false	1.0
13	Integer	false	NULL
14,15	Reseved		NULL

`intMin[]`

In the BIFS-Anim quantization process, `intMax` is set in the following way:

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 82 — Value of `intMin`, depending on `animType`

animType	IntMin
1,2,4,6,7,8 9,10,11,12	NULL
13	aqp.IminInt[0]
14,15	NULL

8.3.3 Quantization process

Let $v_q(t)$ be the value decoded from the bitstream at an instant *t*. Then, the inverse-quantized value at time *t* is:

$$v(t) = \mathrm{InvQuant}\left(v_q(t)\right)$$

The linear quantization and inverse quantization are:

```
int quantize (float Vmin, float Vmax, float v, int Nb)
```

which returns $v_q = \dfrac{v - V_{\min}}{V_{\max} - V_{\min}}(2^{Nb} - 1)$

```
float invQuantize (float Vmin,float Vmax,int vq, int Nb)
```

which returns $\hat{v} = V_{\min} + v_q \dfrac{V_{\max} - V_{\min}}{2^{\max(Nb,1)} - 1}$

If `isQuantized` is true, the quantization/inverse quantization process is the following :

Table 83 — Quantization and inverse quantization process

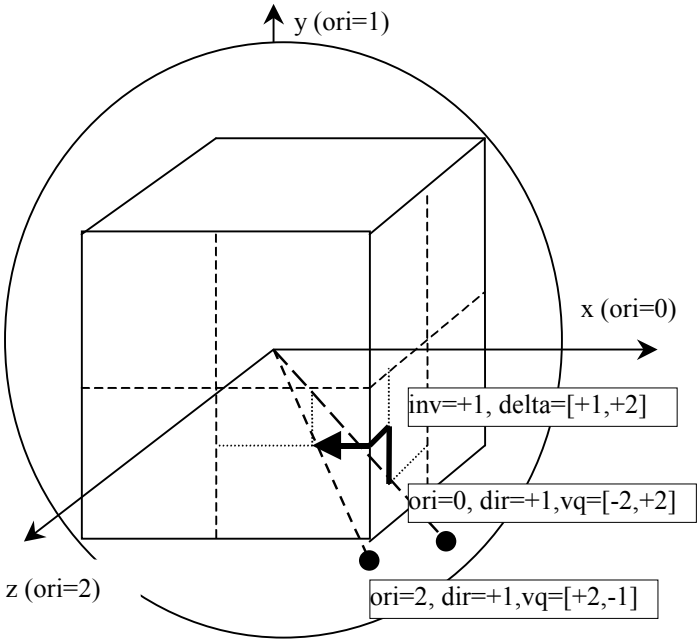
QuantType	animType	Quantization/Inverse Quantization Process
1,2,3,4,5 6,7,8 11,12	1,2,4 6,7,8 11,12	For each component of the vector, the float quantization is applied: $v_q[i] = \mathrm{quantize}(\mathrm{floatMin}[i], \mathrm{floatMax}[i], \mathrm{v}[i], \mathrm{nbBits})$ For the inverse quantization: $\hat{v}[i] = \mathrm{invQ\,uantize}(\mathrm{floatMin}[i], \mathrm{floatMax}[i], \mathrm{v_q}[i], \mathrm{nbBits})$
9,10	9,10	For normals and rotations, the quantization method is as follows. Normals are first renormalized : $v[0] = \dfrac{n_x}{\sqrt{{n_x}^2 + {n_u}^2 + {n_z}^2}}, \quad v[1] = \dfrac{n_y}{\sqrt{{n_x}^2 + {n_u}^2 + {n_z}^2}}, \quad v[2] = \dfrac{n_z}{\sqrt{{n_x}^2 + {n_u}^2 + {n_z}^2}}$ Rotations (axis $\vec{n}$, angle α) are first written as quaternions : $v[0] = \cos(\dfrac{\alpha}{2}) \quad v[1] = \dfrac{n_x}{\|\vec{n}\|}.\sin(\dfrac{\alpha}{2}) \quad v[2] = \dfrac{n_y}{\|\vec{n}\|}.\sin(\dfrac{\alpha}{2}) \quad v[3] = \dfrac{n_z}{\|\vec{n}\|}.\sin(\dfrac{\alpha}{2})$ The number of reduced components is defined to be N: 2 for normals, and 3 for rotations. Note that v is then of dimension N+1. The compression and quantization process is the same for both :

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

QuantType	animType	Quantization/Inverse Quantization Process
		The orientation k of the unit vector v is determined by the largest component in absolute value: $k = \text{argMax}(\lvert v[i] \rvert)$. This is an integer between 0 and N that is encoded using two bits. The direction of the unit vector v is 1 or –1 and is determined by the sign of the component $v[k]$ .Note that this value is not written for rotations (because of the properties of quaternions). The *N* components of the compressed vector are computed by mapping the square on the unit sphere $\left\{ v \quad \left\lVert \frac{v[i]}{v[k]} \right\rVert \le 1 \right\}$ into a N dimensional square : $v_c[i] = \frac{4}{\pi} \tan^{-1}\left(\frac{v[(i+k+1) \bmod (N+1)]}{v[k]} \right) \quad i = 0,\ldots,N$ If nbBits=0, the process is complete. Otherwise, each component of v_c (which lies between –1 and 1) is quantized as a signed integer as follows : $v_q[i] = 2^{\text{nbBits}-1} + \text{quantize}(\text{floatMin}[0], \text{floatMax}[0], v_c[i], \text{nbBits}-1)$ The value is encoded in the bitstream as $v_q[i]$ The decoding process is the following : The value decoded from the stream is converted to a signed value $v_q[i] = v_{decoded} - 2^{.\text{nbBits}-1}$ The inverse quantization is performed $v_c[i] = \text{invQ uantize}(\text{floatMin}[0], \text{floatMin}[0], v_q[i], \text{nbBits}-1)$ After extracting the orientation (k) and direction (dir) , the inverse mapping can be performed : $\hat{v}[k] = \text{dir}. \frac{1}{\sqrt{1 + \sum_{i=0}^{i<N} \tan^2 \frac{\pi . v_c[i]}{4}}}$

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

QuantType	animType	Quantization/Inverse Quantization Process
		$\hat{v}[(i+k+1) \bmod (N+1)] = \tan\left(\frac{\pi . v_c[i]}{4}\right) . \hat{v}[k] \quad i = 0,..., N$ If the object is a rotation, v can be either used directly or converted back from a quaternion to a SFRotation : $\alpha = 2.\cos^{-1}(\hat{v}[0]) \quad n_x = \frac{\hat{v}[1]}{\sin(\alpha/2)} \quad n_y = \frac{\hat{v}[2]}{\sin(\alpha/2)} \quad n_z = \frac{\hat{v}[3]}{\sin(\alpha/2)}$ The entire compression process therefore consists in projecting a vector of the unit sphere onto the face of a cube inscribed inside the sphere, and transmitting separately the face's index (orientation: x, y or z – and direction : + or -) and the coordinates on the face. EXAMPLE — How two different normals are encoded in the case nbBits=3. The compensation process (described in 8.4) is also illustrated.  Note that two quaternions that lie in opposite directions on the unit sphere actually represent the same rotation. This is the reason why the direction is not coded for rotations.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

QuantType	animType	Quantization/Inverse Quantization Process
13,14	13	For integers, the quantized value is the integer shifted to fit the interval [0, 2^{nbBits} -1]. $v_q = v - \text{intMin}$ The inverse quantization process in then : $\hat{v} = \text{intMin} + v_q$
fieldType SFImage		For SFImage types, the width and height of the image are sent. **numComponents** defines the image type. The following four types are enabled: If the value is '00', then a grey scale image is defined. If the value is '01', a grey scale with alpha channel is used. If the value is '10', then an RGB image is used. If the value is '11', then an RGB image with alpha channel is used.

8.4 Compensation process

This subclause describes the mechanism used to compensate a quantized value for a given `FieldData` structure. In other words, how to add a delta to a quantized value to yield the result of addition, which is another quantized value. For vectorial types, this is simply an addition component by component, but for normals and rotations special care has to be taken when performing this addition. This process is used in predictive mode in BIFS-Anim sessions.

Let v_q^1 be the initial quantized value, v^δ be the delta value and v_q^2 be the quantized value resulting from the addition. The general inverse compensation process is :

$$v_q^2 = \text{AddDelta}(v_q^1, v^\delta)$$

v_q^1 and v^δ are interpreted as follows:

A quantized value v_q contains an array of integers vq[]. Additionally, for normals and rotations, v_q^1 contains an orientation and, for normals only, a direction (see 8.3.3).

A delta value v^δ contains an array of integers vDelta[]. Additionnally, for normals, it contains an integer inverse whose value is –1 or 1.

The size of these arrays is that returned by the function `getNbComp(field)`, as described in Table 70.

The result v_q^2 is then computed in the following way :

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 84 — Compensation process for multiple fields and BIFS-Anim

<table>
<tr><th>quantType</th><th>animType</th><th colspan="2">Compensation Process</th></tr>
<tr><td>1,2,4,6,7,8,
9 (other than SFVec3fType),
10 (other than SFRotationType),
11,12,13</td><td>1,2,4,6,7,8
11,12,13</td><td colspan="2">The components of v_q^2 are:

vq2[i] = vq1[i] + vDelta[i]</td></tr>
<tr><td>9 (SFVec3fType),
10 (SFRotation)</td><td>9,10</td><td colspan="2">The addition is first performed component by component and stored in a temporary array:
vqTemp[i] = vq1[i] + vDelta[i].
Let scale = $2^{\max(0,nbBits-1)} - 1$.
Let N the number of reduced components (2 for normals, 3 for rotations)
There are then three cases are to be considered:</td></tr>
<tr><td></td><td></td><td>For every index I,
$|\text{vqTemp[i]}| \le \text{scale}$</td><td>$v_q^2$ is defined by,

vq2[i] = vqTemp[i]
orientation2= orientation1
direction2 = direction1 * inverse</td></tr>
<tr><td></td><td></td><td>There is one and only one index k such that
$|\text{vqTemp[k]}| > \text{scale}$</td><td>$v_q^2$ is rescaled as if gliding on the faces of the mapping cube.

Let inv = 1 if vqTemp[k]>=0 and –1 else
Let dOri = k+1

The components of vq2 are computed as follows
$0 \le i < N - dOri$: vq2[i] = inv*vqTemp[(i+dOri) mod N]
$i = N - dOri$: vq2[i] = inv*2*scale-vqTemp[dOri-1]
$N - dOri < i < N$: vq2[i] = inv*vqTemp[(i+dOri-1) mod N]

orientation2 = (orientation1 + dOri) mod (N+1)
direction2 = direction1 * inverse * inv</td></tr>
<tr><td></td><td></td><td>There are several indices k such that
$|\text{vqTemp[k]}| > \text{scale}$</td><td>The result is undefined</td></tr>
</table>

Note: The BIFS-Anim process is identical to the process applied for optimal encoding of BIFS multiple fields.

8.5 BIFS Configuration

8.5.1 Overview

This subclause describes the terminal configuration for the BIFS elementary stream. It is encapsulated within the `specificInfo` fields of the general `DecoderSpecificInfo` structure (see 7.2.6.7, ISO/IEC 14496-1), which is contained in the `DecoderConfigDescriptor` that is carried in `ES_Descriptors`. If the session is a BIFS-Anim session, the BIFS configuration contains some specific information to describe the animation mask, which specifies the elements of the scene to be animated.

Two terminal configurations are defined in 8.5.2 and 8.5.3, the first being deprecated. The BIFS version of a specific scene description stream is determined by the objectTypeIndication field of the DecoderConfigDescriptor contained in the ES_Descriptor that describes this stream.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.5.2 BIFSConfig

8.5.2.1 Syntax

```
class BIFSConfig extends DecoderSpecificInfo : bit(8) tag=DecSpecificInfoTag {
   unsigned int(5) nodeIDbits;
   unsigned int(5) routeIDbits;
   bit(1) isCommandStream;
   if(isCommandStream) {
      bit(1) pixelMetric;
      bit(1) hasSize;
      if(hasSize) {
         unsigned int(16) pixelWidth;
         unsigned int(16) pixelHeight;
      }
   }
   else {
      bit(1)       randomAccess;
      AnimationMask   animMask();
   }
}
```

8.5.2.2 Semantics

`BIFSConfig` is the terminal configuration for the BIFS elementary streams having their object type indication set to 0x01 and their StreamType set to 0x03. It is encapsulated within the `specificInfo` fields of the general `DecoderSpecificInfo` structure (see 7.2.6.7, ISO/IEC 14496-1), which is contained in the `DecoderConfigDescriptor` that is carried in `ES_Descriptors`.

The parameter `nodeIDbits` sets the number of bits used to represent `nodeIDs`. Similarly, `routeIDbits` sets the number of bits used to represent `ROUTEIDs`.

The boolean `isCommandStream` identifies whether the BIFS stream is a BIFS-Command stream or a BIFS-Anim stream. If the BIFS-Command stream is selected (`isCommandStream` set to TRUE), the following parameters are contained in `BIFSConfig`:

The boolean `isPixelMetric` indicates whether pixel metrics or meter metrics are used.

The boolean `hasSize` indicates whether a desired scene size (in pixels) is specified. If `hasSize` is set to true, `pixelWidth` and `pixelHeight` provide to the receiving terminal the desired horizontal and vertical dimensions (in pixels) of the scene.

If isCommandStream is false, the following information is contained in `BIFSConfig`:

The `randomAccess` boolean signals the mode of the BIFS-Anim stream. If the bit is set to TRUE, it is possible to perform random access in the BIFS-Anim stream at any intra frame. At each intra frame, the statistics of the arithmetic decoder shall be reset. New quantization parameters shall be coded in the bistream or the default parameters sent in the BIFS-Anim mask are used. In this case, the `randomAccessPointFlag` of the BIFS Access Unit shall be set to 1 (see 7.1.1.1.3). If the `randomAccess` bit is set to FALSE, compression may be more efficient, but random access may not be possible at each intra frame. See 7.1.1.3.3 for detailed semantics.

The `AnimationMask` specifies the animation parameters of the BIFS-Anim elementary stream.

8.5.3 BIFSv2Config

8.5.3.1 Syntax

```
class BIFSv2Config extends DecoderSpecificInfo : bit(8) tag=DecSpecificInfoTag {
   bit(1) use3DMeshCoding;
   bit(1) usePredictiveMFField;
   bit(5) nodeIDbits;
   bit(5) routeIDbits;
   bit(5) PROTOIDbits;
   bit(1) isCommandStream;
   if(isCommandStream) {
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      bit(1) pixelMetric;
      bit(1) hasSize;
      if(hasSize) {
         int(16) pixelWidth;
         int(16) pixelHeight;
      }
   }
   else {
      bit(1)       randomAccess;
      AnimationMask   animMask();
   }
}
```

8.5.3.2 Semantics

`BIFSv2Config` is the terminal configuration for elementary streams having their object type indication set to 0x02 and their StreamType set to 0x03. It is encapsulated within the `decSpecificInfo` field of the `DecoderConfigDescriptor` structure that is carried in `ES_Descriptors`. It extends the general `DecoderSpecificInfo` structure (see 7.2.6.7, ISO/IEC 14496-1) .

The `use3DmeshCoding` flag is used to signal that the syntax of 3D Mesh as sepecified by ISO/IEC 14496-2:2004 is used to encode IndexedFaceSet nodes. The `usePredictiveMFField` flag is used to signal that the syntax for predictive MFField, instead of the non-predictive mode, is used to encode IndexedFaceSet nodes. This flag is used for terminals supporting this tool.

Parameters `nodeIDbits` and `routeIDbits` are used similarly as in `BIFSConfig`.

Boolean variables `isCommandStream`, `isPixelMetric`, `hasSize`, `pixelWidth`, and `pixelHeight` are used similarly as in `BIFSConfig`. If the BIFS-Command stream is selected (`isCommandStream` set to TRUE), a `PROTOIDbits field is additionally contained in BIFSv2Config to determine the number of` bits necessary to encode the PROTOs.

If isCommandStream is false, `randomAccess`, and `AnimationMask` are contained and used in `BIFSv2Config similarly as in BIFSConfig`.

8.5.4 AnimationMask

8.5.4.1 Syntax

```
class AnimationMask() {
   int numNodes = 0;
   do {
      ElementaryMask elemMask();
      numNodes++;
       bit(1) moreMasks;
   } while (moreMasks);
}
```

8.5.4.2 Semantics

The `AnimationMask` describes the nodes and fields to be animated, along with the quantization parameters to help decode their values. It consists of a list of `ElementaryMasks`.

If the boolean `moreMasks` is TRUE, another `ElementaryMask` shall be present.

8.5.5 Elementary mask

8.5.5.1 Syntax

```
Class ElementaryMask() {
   bit(BIFSConfiguration.nodeIDbits) nodeID;
      NodeUpdateField node = GetNodeFromID(nodeID);
   switch (node.nodeType) {
      case FaceType:
      break;
      case BodyType:
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      break;
      case IndexedFaceSet2DType:
      break;
      default:
      InitialFieldsMask initMask(node);
   }
}
```

8.5.5.2 Semantics

The `ElementaryMask` describes how to animate the elements of a node.

The integer `nodeID` identifies the animated node.

If the node's `nodeType` is **FDP**, **BDP** or **IndexedFaceSet2D**, no further information is expected.

If any other case, an `InitialFieldsMask` shall be present.

8.5.6 InitialFieldsMask

8.5.6.1 Syntax

```
class InitialFieldsMask(NodeUpdateField node) {
   for(i=0; i<node.numDYNfields; i++)
      bit(1) node.isAnimField[i];
   int i;
   for(i=0; i<node.numDYNfields; i++) {
      if (node.isAnimField[i]) {
         FieldData field = node.field[node.dyn2all[i]];
         AnimFieldQP aqp = field.aqp;
         if (!isSF(field) {
            bit(1) aqp.isTotal;
            if (!aqp.isTotal) {
               unsigned int(5) nbBits;
               do {
                  int(nbBits) aqp.indexList[aqp.numElement++];
                  bit(1) moreIndices;
               } while (moreIndices);
         }
         InitialAnimQP QP[i](field.aqp);
      }
   }
}
```

8.5.6.2 Semantics

The `InitialFieldsMask` specifies which fields of a given node are animated.

The array of booleans `isAnimField` describes whether the fields (indexed with `dynIDs`) are animated.

If a multiple field is animated and if the boolean `isTotal` is TRUE, all the of the field's individual elements are animated.

If a multiple field is animated and if the boolean `isTotal` is FALSE, the indices of the animated individual field are sent and stored in `aqp.indexList[]`. The number of bits used to encode them is specified by `nbBits`. If the boolean `moreIndices` is TRUE, another index shall be present.

An `InitialAnimQP` shall then be expected.

8.5.7 InitialAnimQP

8.5.7.1 Syntax

```
InitialAnimQP(animFieldQP aqp) {

   aqp.useDefault=FALSE;
   uint(4) type;
   aqp.animType = type;
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   switch(aqp.animType) {

         case  4:          // Color
         case  8:          // BoundFloats
            bit(1)     aqp.useDefault
         case  1:          // Position 3D
         case  2:          // Position 2D
         case 15:          // Position 4D
         case 11:          // Size 3D
         case 12:          // Size 2D
         case  7:          // Floats
            if (!aqp.useDefault) {
               for (i=0;i<getNbBounds(aqp.animType);i++) {
                  bit(1)       useEfficientCoding
                  GenericFloat aqp.Imin[i](useEfficientCoding);
               }
               for (i=0;i<getNbBounds(aqp.animType);i++) {
                  bit(1)       useEfficientCoding
                  GenericFloat aqp.Imax[i](useEfficientCoding);
               }
            }
            break;

         case 13:          // Integers
            int(32)    aqp.IminInt[0];
         break;
   }
   unsigned int(5)      aqp.INbBits;
    for (i=0;i<getNbBounds(aqp.animType);i++) {
         int(INbBits+1) vq
         aqp.Pmin[i] = vq-2^aqp.INbBits;
   }
   unsigned int(4)      aqp.PNbBits;
}
```

8.5.7.2 Semantics

The `InitialAnimQP` specifies the field's default quantization parameters.

The quantization bounds are first coded. For `animTypes` that have default finite bounds (`Colors`, `BoundFloats`), the default bounds of the field coding tables data structures can optionally be used by setting `aqp.useDefault` to TRUE. For all other `animTypes`, this boolean is set to FALSE. For all vectorial `animTypes` (`Position3D`, `Position2D`, `Size3D`, `Size2D`, `Float`, `BoundFloat`, `Color`), if `aqp.useDefault` is FALSE, the quantization bounds `aqp.Imin[]` and `aqp.Imax[]` are coded. Depending on the value of `useEfficientCoding`, these bounds are coded using `GenericFloat` as floats of 32 bits or less. For the `animTypes` `Angle`, `Normal` and `Rotation`, no quantization bounds are coded.

The number of bits used in the quantization process, `aqp.INbBits`, is then coded. The quantization process (see 8.3.3) is used in intra mode only.

The minimal bounds used to offset the values obtained from the compensatiation process in predictive mode, `Pmin[]`, are then coded. `Pmins` may have values in the range $-2^{INbBits}$ to $2^{INbBits}-1$. The value is coded as an unsigned integer using `INbBits`+1 bits and has the value `PMin`+$2^{INbBits}$.

The number of bits used for the predictive values, `aqp.PNbBits`, is then coded. The compensation process (see 8.4) is used in predictive mode only.

8.6 BIFS Command Syntax

8.6.1 Overview

This subclause describes the commands that can be sent to act on the scene. They allow insertion, modification, and deletion of elements of the scene (new scenes, nodes, fields). All BIFS information is encapsulated in BIFS command frames. Each frame may contain commands that perform a number of operations, such as insertion, deletion, or modification of scene nodes, their fields, or routes. When a BIFS command is invalid, for example if it references missing

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

nodes or ROUTEs, the terminal shall ignore the command. If the command cannot be parsed, then the command and the rest of the access unit shall be ignored.

8.6.2 Command Frame

8.6.2.1 Syntax

```
class CommandFrame() {
   do {
      Command command();
      bit(1) continue;
   } while (continue);
}
```

8.6.2.2 Semantics

A `CommandFrame` is a collection of BIFS-Commands, and corresponds to one access unit. A sequence of commands may be sent. The boolean value `continue`, when TRUE, indicates that another command follows the current one.

8.6.3 Command

8.6.3.1 Syntax

```
class Command() {
   bit(2) code;
   switch (code) {
   case 0:
      InsertionCommand insert();
      break;
   case 1:
      DeletionCommand delete();
      break;
   case 2:
      ReplacementCommand replace();
      break;
   case 3:
      SceneReplaceCommand sceneReplace();
      break;
}
```

8.6.3.2 Semantics

For each `Command`, the 2-bit flag, `code`, signals one of the four basic commands: insertion, deletion, replacement, and scene replacement.

8.6.4 Insertion Command

8.6.4.1 Syntax

```
class InsertionCommand() {
   bit(2) parameterType ;
   switch parameterType {
   case 0:
      NodeInsertion nodeInsert();
      break;
   case 1:
      ExtendedUpdate extendedUpdate();
      break;
   case 2:
      IndexedValueInsertion idxInsert();
      break;
   case 3:
      ROUTEInsertion ROUTEInsert();
      break ;
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   }
}
```

8.6.4.2 Semantics

There are four basic insertion commands, signaled by the 2-bit flag `parameterType`.

If `parameterType` is 0, a `NodeInsertion` is expected.

If `parameterType` is 1, an `ExtendedUpdate` is expected.

If `parameterType` is 2, an `IndexedValueInsertion` is expected.

If `parameterType` is 3, a `ROUTEInsertion` is expected.

8.6.5 Node Insertion

8.6.5.1 Syntax

```
class NodeInsertion() {
   bit(BIFSConfiguration.nodeIDbits) nodeID ;
      int ndt=GetNDTFromID(nodeID);
   bit(2) insertionPosition;
   switch (insertionPosition) {
   case 0:    // insertion at a specified position
      bit (8) position;
      SFNode node(ndt);
      break;
   case 2:      // insertion at the beginning of the field
      SFNode node(ndt);
      break;
   case 3:    // insertion at the end of the field
      SFNode node(ndt);
      break;
   }
}
```

8.6.5.2 Semantics

The insertion of a node may be performed on a node that has an MFNode children field. Inserting a node adds the node at the desired position in the children multiple field. The command is thus valid only if the node referred to by `nodeID` contains a children field of type MFNode.

A node may be inserted in the children field of a grouping node. The `nodeID` of this grouping node is first coded.

The NDT of the inserted node can be determined from the NDT of the children field in which the node is inserted.

The position in the children field where the node shall be inserted, insertionPosition is then coded on two bits :

If the `insertionPosition` is 0, the node is inserted at a specified position coded on 8 bits.

If the `insertionPosition` is 2, the node is inserted at the beginning of the field.

If the `insertionPosition` is 3, the node is inserted at the end of the field.

The node is then coded.

8.6.6 ExtendedUpdate

8.6.6.1 Syntax

```
class ExtendedUpdate() {
  bit(8) updateType ;
  switch (updateType) {
  case 0:
    PROTOlist protos();
    break;
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  case 1:
    PROTOlistDeletion listDelete();
    break;
  case 2:
    // remove all existing PROTOs
    break ;
  case 3:
    MultipleIndexedFieldReplacement mifReplacement();
    break;
  case 4:
    MultipleFieldReplacement mfReplacement();
    break;
  case 5:
    GlobalQuantizationConfiguration globalQuantizer();
    break;
  case 6:
    NodeDeletionEx() nodeDeleteEx()
    break;
  }
}
```

8.6.6.2 Semantics

There can be up to 256 extended update types.

If updateType is 0, the insertion of a list of PROTOs is signalled.

If updateType is 1, the deletion of a list of PROTOs is signalled.

If updateType is 2, the deletion of all previously defined PROTOs is signalled.

If updateType is 3, a MultipleIndexedFieldReplacement is signalled.

If updateType is 4, a MultipleFieldReplacement is signalled.

If updateType is 5, a GlobalQuantizationConfiguration is signalled.

8.6.7 PROTOlistInsertion

8.6.7.1 Syntax

```
class PROTOlistInsertion () {
   PROTOList list();
}
```

8.6.7.2 Semantics

This element signals the insertion of a list of PROTOs.

Inserting a new PROTO with id=PID means that, from that point in time on, there can be a PROTO instance referring to a PROTO declaration with id=PID.

When an existing PROTO already has the same id as a newly inserted PROTO, it means that new PROTO instances refer to the new definition, while previous PROTO instances are unchanged.

8.6.8 PROTOlistDeletion

8.6.8.1 Syntax

```
class PROTOlistDeletion () {
  bit(1) listDescription;
  if (listDescription) {
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    bit(1) morePROTOs;
    while (morePROTOs) {
      bit(BIFSConfiguration.protoIDbits) protoID;
      bit(1) morePROTOs;
    }
  } else { // vector
    bit(5) len;
    bit(len) numPROTOs;
    for (i=0; i < numPROTOs; i++)
      bit(BIFSConfiguration.protoIDbits) protoID;
  }
}
```

8.6.8.2 Semantics

This command can delete a selection of the existing PROTO declarations. This only deletes the definition, i.e. existing instances are not influenced by the deletion, but no new instance of the deleted PROTOs can be created.

8.6.9 MultipleFieldReplacement

8.6.9.1 Syntax

```
class MultipleFieldReplacement () {
  bit(BIFSConfiguration.nodeIDbits) nodeID;
  nodeData = getNodeFromID(nodeID);
  bit(1) maskAccess;
  if (maskAccess)
    MaskNodeDescription mnode(nodeData);
  else
    ListNodeDescription lnode(nodeData);
}
```

8.6.9.2 Semantics

This element allows the modification of some fields of a node. This element is meant to replace a set of FieldReplacement commands operating on the same node. The fact that the nodeID is not repeated allows for better compression.

8.6.10 MultipleIndexedFieldReplacement

8.6.10.1 Syntax

```
class MultipleIndexedFieldReplacement () {
   bit(BIFSConfiguration.nodeIDbits) nodeID;
      NodeData node = GetNodeFromID(nodeID);
   int(node.nINbits) inID;
   bit (5) lenPosition;
   bit (5) lenNum;
   bit (lenNum) numPositions;
   for (i=0; i < numPositions; i++) {
      bit(lenPosition) position;
      SFField value(node.field[node.in2all[inID]]);
   }
}
```

8.6.10.2 Semantics

This element allows the modification of some values of a multiple field. This element is meant to replace a set of IndexedValueReplacement commands operating on the same field. The fact that the nodeID and inID are not repeated allows for better compression.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.6.11 GlobalQuantizationConfiguration

8.6.11.1 Syntax

```
class GlobalQuantizationConfiguration () {
   SFNode gqp();
}
```

8.6.11.2 Semantics

This element allows the setting of the global quantizer defined in 7.2.2.108.2. The use of the SFNode structure allows to give this global quantizer a DEF ID, or to reset global quantization by setting it to a NULL node, or to use a QuantizationParameter node already present in the scene.

In the context of this command, a node of any other type than QuantizationParameter shall be treated as a NULL node, resetting global quantization. The isLocal field of QuantizationParameter is meaningless in this context.

8.6.12 NodeDeletionEx

8.6.12.1 Syntax

```
class NodeDeletionEx () {
   bit (BIFSConfiguration.nodeIDbits) nodeID;
}
```

8.6.12.2 Semantics

The NodeDeletionEx has semantics of NodeDeletion, see 8.6.16.2, except when deleting a child of an OrderedGroup from the children field the corresponding entry for the order, if it exists, is also deleted from the order field.

8.6.13 IndexedValue Insertion

8.6.13.1 Syntax

```
class IndexedValueInsertion() {
   bit(BIFSConfiguration.nodeIDbits) nodeID;
   NodeUpdateField node=GetNodeFromID(nodeID);
   int(node.nINbits) inID;
   bit(2) insertionPosition;
   switch (insertionPosition) {
   case 0: // insertion at a specified position
      bit (16) position;
      SFField value(node.field[node.in2all[inID]]);
      break;
   case 2:   // insertion at the beginning of the field
      SFField value(node.field[node.in2all[inID]]);
      break;
   case 3:   // insertion at the end of the field
      SFField value(node.field[node.in2all[inID]]);
      break;
   }
}
```

8.6.13.2 Semantics

The `IndexedValueInsertion` syntax allows the insertion of a new value in a multiple field at the desired position.

The `nodeID` of the node in whose field the value is to be inserted is first coded.

The field in which the value is inserted must be a multiple field type. The field is signaled with an `inID`. The `inID` is parsed using the table for the node type of the node in which the value is inserted. The node type may be determined from the `nodeID`

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The position in the children field where the node shall be inserted, `insertionPosition`, is then coded:

If the `insertionPosition` is 0, the node is inserted at a specified position coded using 16 bits.

If the `insertionPosition` is 2, the node is inserted at the beginning of the field.

If the `insertionPosition` is 3, the node is inserted at the end of the field.

The node is then coded.

8.6.14 ROUTE Insertion

8.6.14.1 Syntax

```
class ROUTEInsertion() {
   bit(1) isUpdatable;
   if (isUpdatable) {
      bit(BIFSConfiguration.routeIDbits) routeID;
   }
   bit(BIFSConfiguration.nodeIDbits) departureNodeID;
   NodeData nodeOUT=GetNodeFromID(departureNodeID);
   int(nodeOUT.nOUTbits) departureID;
   bit(BIFSConfiguration.nodeIDbits) arrivalNodeID;
   NodeData nodeIN=GetNodeFromID(arrivalNodeID);
   int(nodeIN.nINbits) arrivalID;
}
```

8.6.14.2 Semantics

The ROUTE insertion syntax permits the addition of a new ROUTE in the list of ROUTEs for the current scene.

A ROUTE is inserted in the list of ROUTEs by specifying a new ROUTE.

If the boolean `isUpdatable` is TRUE, a `routeID` is coded to allow the ROUTE to be referenced.

The `nodeID` of the route's departure, `departureNodeID`, is first coded.

The `outID` of the departure field in the departure node, `departureID`,is then coded.

The `nodeID` of the route's arrival, `arrivalNodeID`, is then coded.

The `inID` of the arrival field in the arrival node, `arrivalID`, is then coded.

8.6.15 Deletion Command

8.6.15.1 Syntax

```
class DeletionCommand() {
   bit(2) parameterType ;
   switch (parameterType) {
   case 0:
      NodeDeletion nodeDelete();
      break ;
   case 2:
      IndexedValueDeletion idxDelete();
      break ;
   case 3:
      ROUTEDeletion ROUTEDelete();
      break ;
   }
}
```

8.6.15.2 Semantics

There are three types of deletion commands, signalled by the 2-bit flag `parameterType`.

If `parameterType` is 0, a `NodeDeletion` is expected.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If parameterType is 2, an IndexedValueDeletion is expected.

If parameterType is 3, a ROUTEDeletion is expected.

8.6.16 Node Deletion

8.6.16.1 Syntax

```
class NodeDeletion() {
   bit(BIFSConfiguration.nodeIDbits) nodeID;
}
```

8.6.16.2 Semantics

The NodeDeletion syntax permits the deletion of a node with a specific nodeID. The node deletion deletes the node and all its instances, if it was referenced elsewhere in the scene with a USE statement.

The node deletion is signalled by the nodeID of the node to be deleted. When deleting a node, all fields shall also deleted, as well as all ROUTEs related to the node or its fields.

8.6.17 IndexedValue Deletion

8.6.17.1 Syntax

```
class IndexedValueDeletion() {
   bit(BIFSConfiguration.nodeIDbits) nodeID;
      NodeData node=GetNodeFromID(nodeID);
   int(node.nINbits) inID;
   bit(2) deletionPosition;
   switch (deletionPosition) {
   case 0: // deletion at a specified position
      bit(16) position;
      break;
   case 2:   // deletion at the beginning of the field
      break;
   case 3:   // deletion at the end of the field
      break;
   }
}
```

8.6.17.2 Semantics

The IndexedValueDeletion syntax permits the deletion of an element of a multiple value field.

The nodeID of the node to be deleted is first coded.

The inID of the field to be deleted is then coded.

The position in the children field from where the value shall be deleted, deletionPosition, is then coded:

If the insertionPosition is 0, the value at specified position, coded using 16 bits, shall be deleted.

If the insertionPosition is 2, the value at the beginning of the field shall be deleted.

If the insertionPosition is 3, the value at the end of the field shall be deleted.

The IndexedValueDeletion operation on an MFNode element that is “DEF”ed or “USE”ed removes *only* that reference to the node and leaves any others intact. The implementation should behave as if the deleted node’s usage is reference counted and decrement the reference count; when the reference count indicates that there will be no more instances the node definition itself is removed.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.6.18 ROUTE Deletion

8.6.18.1 Syntax

```
class ROUTEDeletion() {
   bit(BIFSConfiguration.routeIDbits) routeID;
}
```

8.6.18.2 Semantics

The `ROUTEDeletion` syntax permits the deletion of a ROUTE with a given `routeID` from the list of active ROUTEs.

Deleting a ROUTE is performed by specifying its `routeID`. This is similar to the deletion of a node.

8.6.19 Replacement Command

8.6.19.1 Syntax

```
class ReplacementCommand() {
   bit(2) parameterType ;
   switch (parameterType) {
   case 0:
      NodeReplacement nodeReplace();
      break;
   case 1:
      FieldReplacement fieldReplace();
      break;
   case 2:
      IndexedValueReplacement idxReplace();
      break;
   case 3:
      ROUTEReplacement ROUTEReplace();
      break;
   }
}
```

8.6.19.2 Semantics

There are 4 replacement commands, signalled by the 2-bit flag `parameterType`.

If `parameterType` is 0, a `NodeReplacement` is expected.

If `parameterType` is 1, a `FieldReplacement` is expected.

If `parameterType` is 2, a `IndexedValueReplacement` is expected.

If `parameterType` is 3, a `ROUTEReplacement` is expected.

8.6.20 Node Replacement

8.6.20.1 Syntax

```
class NodeReplacement() {
   bit(BIFSConfiguration.nodeIDbits) nodeID;
   SFNode node(SFWorldNode);
}
```

8.6.20.2 Semantics

The `NodeReplacement` syntax permits the deletion of an existing node and its replacement with a new node. All ROUTEs pointing to the deleted node as well as any instances of the node created through the USE mechanism shall be deleted.

The node to be replaced is signalled by its `nodeID`. The new node is encoded with the `SFWorldNode` node data type, which is valid for all BIFS nodes, in order to avoid necessitating the NDT of the replaced node to be established.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.6.21 Field Replacement

8.6.21.1 Syntax

```
class FieldReplacement() {
   bit(BIFSConfiguration.nodeIDbits) nodeID ;
   NodeData node = GetNodeFromID(nodeID);
   int(node.nINbits) inID;
   Field value(node.field[node.in2all[inID]]);
}
```

8.6.21.2 Semantics

This `FieldReplacement` syntax permits the modification of the value of a field of an existing node. The existing value shall be deleted and replaced with the new value.

The `nodeID` of the node whose field is to be modified is first coded

The `inID` of the field to be modified is then coded

The new field is then coded

8.6.22 IndexedValueReplacement

8.6.22.1 Syntax

```
class IndexedValueReplacement() {
   bit(BIFSConfiguration.nodeIDbits) nodeID;
      NodeData node = GetNodeFromID(nodeID);
   int(node.nINbits) inID;
   bit(2) replacementPosition;
   switch (replacementPosition) {
   case 0: // replacement at a specified position
      bit (16) position;
      SFField value(node.field[node.in2all[inID]]);
      break;
   case 2: // replacement at the beginning of the field
      SFField value(node.field[node.in2all[inID]]);
      Break;
   case 3: // replacement at the end of the field
      SFField value(node.field[node.in2all[inID]]);
      break;
   }
}
```

8.6.22.2 Semantics

The `IndexedValueReplacement` syntax permits the modification of the value of an element of a multiple field. As for any multiple field access, it is possible to replace at the beginning, the end or at a specified position in the multiple field.

The `nodeID` of the node whose field is to be modified is first coded

The `inID` of the field whose value is to be modified is then coded

The position in the children field where value has to be modified, `replacementPosition`, is then coded:

If the `insertionPosition` is 0, the value at specified position, coded using 16 bits, is modified.

If the `insertionPosition` is 2, the value at the beginning of the field is modified.

If the `insertionPosition` is 3, the value at the end of the field is modified.

The new value is then coded as a SFField.

The `IndexedValueReplacement` operation replaces an MFNode element. If the replaced element is “DEF”ed or “USE”ed, *only* that reference to the node is replaced and all others are left intact. The implementation should behave as if

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

the replaced node's usage is reference counted and decrement the reference count; when the reference count indicates that there are no more instances, the node definition itself is replaced.

8.6.23 ROUTE Replacement

8.6.23.1 Syntax

```
class ROUTEReplacement() {
   bit(BIFSConfiguration.routeIDbits) routeID;
   bit(BIFSConfiguration.nodeIDbits) departureNodeID;
   NodeData nodeOUT = GetNodeFromID(nodeID);
   int(nodeOUT.nOUTbits) departureID;
   bit(BIFSConfiguration.nodeIDbits) arrivalNodeID;
   NodeData nodeIN = GetNodeFromID(nodeID);
   int(nodeIN.nINbits) arrivalID;
}
```

8.6.23.2 Semantics

Replacing a ROUTE deletes the replaced ROUTE and replaces it with the new ROUTE.

The `routeID` of the ROUTE to be replaced is first coded.

The `nodeID` of the new route's departure, `departureNodeID`, is then coded.

The `outID` of the departure field in the departure node, `departureID`, is then coded.

The `nodeID` of the route's arrival, `arrivalNodeID`, is then coded.

The `inID` of the arrival field in the arrival node, `arrivalID`, is then coded.

8.6.24 Scene ReplaceCommand

8.6.24.1 Syntax

```
class SceneReplaceCommand() {
   BIFSScene scene();
}
```

8.6.24.2 Semantics

Replacing a scene results in the entire BIFS scene being replaced with a new `BIFSScene` scene. When used in the context of an **Inline** node, this corresponds to replacement of the sub-scene (previously assumed to be empty). In a BIFS elementary stream, the `SceneReplacement` commands are the only random access points.

8.7 BIFS Scene

8.7.1 BIFSScene

8.7.1.1 Syntax

```
class BIFSScene() {
   bit(6) reserved;
   bit(1) USENAMES;
   PROTOlist protos();
   SFNode nodes(SFTopNode);
   bit(1) hasROUTEs;
   if (hasROUTEs) {
      ROUTEs routes();
   }
}
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.7.1.2 Semantics

The integer `reserved` may be used in future extensions. It shall be set to 0.

The `BIFSScene` structure represents the global scene. A `BIFSScene` is always associated to a `ReplaceScene` BIFS-Command message. The `BIFSScene` is structured in the following way:

The nodes of the scene are described first as an SFNode. The first node in the scene shall be of type SFTopNode (see node coding tables in electronic attachment).

A boolean value, `USENAMES`, sets a global flag that indicates whether PROTOs, SFNodes, and ROUTEs store their field names and IDs as strings, as well as integer values. (This is needed for MPEG-J and Scripts, which refer to fields, by their explicit string name).

A list of PROTOs associated with the scene is stored in `protos`.

ROUTEs are described after all nodes

All BIFS scenes shall begn with a node of type `SFTopNode`. This implies that the top node may be one of **Layer2D**, **OrderedGroup**, **Group** or **Layer3D**.

8.7.2 Encoding of PROTOs

This subclause describes how PROTOs, a mechanism that allows scene components to be reused, are encoded. The encoding of PROTOs allows specification of quantization and animation categories for the PROTO parameters, so that PROTOs can take advantage of BIFS compression capabilities just like any other (predefined) node in the node coding tables in electronic attachment. A `PROTOlist` is stored in a `BIFSScene` and contains a list of PROTOs that are associated with that scene.

8.7.2.1 PROTOlist

8.7.2.1.1 Syntax

```
class PROTOlist() {
   bit(1) morePROTOs;
   while (morePROTOs) {
      PROTOdeclaration proto();
      bit(1) morePROTOs;
   }
}
```

8.7.2.1.2 Semantics

The PROTOlist stores a list of PROTOs. A one-bit flag `morePROTOs` signals the fact that more PROTOs are being declared.

8.7.2.2 PROTOdeclaration

8.7.2.2.1 Syntax

```
PROTOdeclaration() {
   PROTOinterfaceDefintion interface();
   NodeData protoData = MakePROTOdata(interface);
   PROTOcode code(protoData);
   PROTOcodingTable table(protoData);
}
```

8.7.2.2.2 Semantics

The PROTO declaration is made of the `PROTOinterface` definition, the PROTO implementation in terms of nodes, and the PROTO coding table. The PROTO coding table codes the equivalent of the Node Coding table for the PROTO. This makes it possible to animate, quantize and update the PROTO instantiations using the identical mechanisms used for the pre-defined nodes.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.7.2.3 PROTOinterfaceDefinition

8.7.2.3.1 Syntax

```
class PROTOinterfaceDefinition {

   bit(protoIDbits) protoID;
   if (USENAMES) {
      String PROTOname;
   }
   bit(1) moreFields;
   while (moreFields) {
      bit(2) eventType;
      bit(6) fieldType;
      if (USENAMES) {
         String fieldName;
      }
      if ((eventType == 0b00) || (eventType == 0b01)) {
         FieldData fieldData = MakeField(fieldType);
         Field defaultValue(fieldData, null);
      }
      bit(1) moreFields;
   }
}
```

8.7.2.3.2 Semantics

A protoID is given to the PROTO in order to be able to refer to it. The `protoIDbits` is obtained from the `BIFSConfiguration` and encodes the ID of the PROTO in the PROTO table. The PROTO interface contains a one bit `moreFields` field that specifies if more PROTO fields are encoded. Then for each field, the event type (exposedField, field, eventIn, eventOut) and the fieldType is given (SFBool, SFFloat, etc). The `eventType` is coded using 2 bits according to Table 85. The `fieldType` is coded using 6 bits according to Table 86. When the field type is a node, it is coded as an `SFWorldNode` or `MFWorldNode`. The `USENAMES` is a static constant set at the `BIFSScene` level, which selects the fact that node and field names are encoded as Strings as well as IDs.

Table 85 — eventTypes.

field	0b00
exposedField	0b01
eventIn	0b10
eventOut	0b11

8.7.2.4 PROTOcode

8.7.2.4.1 Syntax

```
class PROTOcode(isedNodeData protoData) {
  bit(1) isExtern;
  if (isExtern) {
    MFUrl locations;
  } else {
    PROTOlist subProtos();
    do {
      SFNode node(SFWorldNodeType, protoData);
      bit(1) moreNodes;
    } while (moreNodes);
    bit(1) hasROUTEs;
    if (hasROUTEs) {
      ROUTEs routes();
    }
  }
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
}
```

8.7.2.4.2 Semantics

First a flag signals whether the prototype is a PROTO, which then has his code included in the proto declaration, or if it is an EXTERNPROTO, in which case only an external reference is provided. The EXTERNPROTO is an authoring facility that makes possible the distribution of PROTOs in external libraries that can be reused across scenes. The EXTERNPROTO opens a BIFS-Command stream that contains a ReplaceScene command with a BIFSScene containing the PROTO definitions. The EXTERNPROTO code is found in the PROTO contained in this new scene. The url field allows to uniquely identify the EXTERNPROTO code through the following url scheme : “resource_URL#ProtoID” or “resource_URL#ProtoName”, where resource_URL is the location of the scene to open, ProtoID the binary ID of the proto in the new scene and ProtoName the name of the proto in the new scene when this scene is encoded with USENAMES. In case “#ProtoID” or “#ProtoName” is omitted in the location, the first proto in the new scene with the same PROTOinterfaceDefinition shall be used. Nodes contained in the EXTERNPROTO scene shall be ignored. Opening of the scene description stream follows the MPEG-4 content access procedure described in 7.2.7.3.8.2, ISO/IEC 14496-1.

In case of a PROTO, the PROTOcode contains a (possibly empty) list of the sub-PROTOs of this PROTO in subProtos, followed by the code to execute the PROTO. The code is specified as a set of `SFNode`s, using a standard `SFNode` definition with the additional possibility to declare an `IS` field. Moreover, the PROTO body may contain ROUTEs if the `hasROUTE` flag is set to 1.

8.7.2.5 PROTOCodingTable

8.7.2.5.1 Syntax

```
PROTOCodingTable(NodeData protoData) {
     InterfaceCodingMask mask(protoData);
     InterfaceCodingParameters icp(protoData);
}
```

8.7.2.5.2 Semantics

The PROTO coding table defines the Quant and Anim parameters and the parameters necessary to reconstruct a NCT table for the PROTO definition.

8.7.2.6 InterfaceCodingMask

8.7.2.6.1 Syntax

```
InterfaceCodingMask(NodeData protoData) {
     bit(1) protoData.useQuant;
     bit(1) protoData.useAnim;
}
```

8.7.2.6.2 Semantics

The mask encodes two Boolean values to store whether the PROTO can further be animated (using BIFS-Anim), or quantized.

8.7.2.7 InterfaceCodingParameters

8.7.2.7.1 Syntax

```
InterfaceCodingParameters(InterfaceCodingMask mask, NodeData protoData) {
   for (int i =0; i < protoData.numALLfields ; i++) {
      if (protoData.useQuant) {
         if (protoData.field[i].isDEF()) {
            bit(4) quantCategory;
            if (quantCategory == 13)
            bit(5) nbBits;
            bit(1) hasMinMax;
            if (hasMinMax) {
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            CastToSF(Field) minFieldValue(protoData.field[i]), null);
            CastToSF(Field) maxFieldValue(protoData.field[i]), null);
         }
      }
   }
   if (protoData.useAnim) {
      if (protoData.field[i].isIN()) {
         bit(1) isDyn;
         if (isDyn) {
            int(4) animCategory;
         }
      }
   }
 }
}
```

8.7.2.7.2 Semantics

The `InterfaceCodingParameters` includes all the necessary parameters to further update, quantize and animate the PROTO instantiation.

If the `useQuant` information is TRUE, and the field is of « DEF » type, the quantization category will be encoded. If the category is 13, the number of bits for this category is further needed. To quantize, it is further necessary to encode the min and max values for the field. When the field is an SFField, the functions CastToSF(field) parses an SFField, but when the field is an MFField, the function CastToSF() parses the SFType corresponding the MFType.

If the `useAnim` is TRUE and the field type is IN, then the anim category will be encoded.

8.7.3 SFNode

8.7.3.1 Syntax

```
class SFNode(int nodeDataType, NodeData protoNodeData) {
  bit(1) isReused;
  if (isReused) {
    bit(BIFSConfiguration.nodeIDbits) nodeID;
  } else {
    int nodeGroup = 0;
    do {
      nodeGroup++;
      bit(GetNDTnbBits(nodeGroup, nodeDataType)) localNodeType;
    } while (localNodeType == 0);

    if (nodeGroup == 2 && localNodeType == 1 ) {
      bit(BIFSConfiguration.PROTOIDbits) protoID;
      nodeDataType = PROTODataType;
      nodeType = GetNodeType(nodeGroup, nodeDataType, protoID);
    }
    else {
      nodeType = GetNodeType(nodeGroup, NodeDataType, localNodeType);
    }

    bit(1) isUpdateable;
    if (isUpdateable) {
      bit(BIFSConfiguration.nodeIDbits) nodeID;
      if (USENAMES) {
        String name;
       }
    }

   if (nodeGroup == 1 && nodeType == IndexedFaceSetType
    && BIFSConfiguration.use3DmeshCoding == 1) {
      Mesh3D mnode;
    } else {
      bit(1) MaskAccess;
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        NodeData nodeData = MakeNode(nodeGroup, nodeDataType, nodeType);
        nodeData.protoData = protoNodeData;
        if (MaskAccess) {
         MaskNodeDescription mnode(nodeData);
        } else {
          ListNodeDescription lnode(nodeData);
        }
      }
    }
}
```

8.7.3.2 Semantics

The `SFNode` syntax represents a generic node. The encoding depends on the context of the parent field of the node. This context is described by the parent field's node data type (NDT).

If `isReused` is TRUE then this node is a reference to another node, identified by its `nodeID`. This is equivalent to the use of the USE statement in ISO/IEC 14772-1:1998. In the special case where the **nodeID** of the node has the value (2^(BIFSConfiguration.nodeIDbits) – 1), that is all 1s, then the node shall be deemed to have NULL value. This special value shall not be used as a node DEF ID.

If `isReused` is FALSE, then a complete node is provided in the bitstream. This requires that the `nodeType` be inferred from the node data type. The node is referenced by its `localNodeType` in the node data type table. Then, this information is converted into the node's `nodeType` (e.g. its `localNodeType` in the `SFWorldNode` NDT table for the particular node group).

If the `localNodeType` is 0 this is an extension code to escape to the next node grouping (see 8.2.2 for a definition of node groups). For each extension code decoded the nodeGroup index is incremented until all extension codes are consumed.

Note clarifying extension coding: During the specification amendment process, new node data types have been added. When a new node data type is added to this specification it is always in a new node group. However, conceptually it is added to all existing groups. In many groups, and of course in those groups before it was defined, there will be no nodes of that type. Hence the only entry in the node type table for each group, except group 2, will be the extension code 0. Since it is the only entry it does not need to be explicitly coded. Only in group 2 is this not true; there the table also includes PROTO-value of 1. For example to code the BAP node from the SFBAPNode type from group 2 there is first an implicit 0 extension code to get to group 2 (there are no SFBAPNode types in group 1 hence there is only one entry, the zero extension, which does not need explicit coding) and then the BAP node itself is coded as 10.

If the nodeGroup is 2 and the `LocalNodeType` is equal to 1, this indicates that the `NodeType` is a PROTO. Then, the global node type is constructed according to the list of PROTOs declared and their ID. When a PROTO is declared, a new node type is created from the `protoID` and dynamically added to the `SFWorldNode` NDT table for the node group of index two. So, the `nodeType` can be inferred from the given `protoID`, irrespectively from the `nodeDataType`. To specify that `nodeDataType` is irrelevant in the case of PROTO its value is set to `PROTODataType`.

When a PROTO is declared, a new node type is created and added to the global node type table of supported nodes. The function `GetPROTONodeType(PROTODataType, PROTONodeType)` returns the ID for the extended global node type of a given PROTO given its PROTO type.

If a node is detected as an IndexedFaceSet node and the Mesh3D syntax is used (see 7.2.2.64), then the IndexedFaceSet node is coded as a specific visual object (see ISO/IEC 14496-2:2004).

The `isUpdatable` flag enables the assignment of a `nodeID` to the node. This is equivalent to the DEF statement of ISO/IEC 14772-1:1998.

The node definition follows using either a `MaskNodeDescription`, or a `ListNodeDescription`.

The `nodeType` is a number that represents the type of the node. This `nodeType` is coded using a variable number of bits for efficiency reasons. The exact type of node may be determined from the `nodeType` as follows:

1. The data type of the field parsed indicates the node data type. The root node is always of type `SFTopNode`.
2. From the `nodeDataType`, the `nodeGroup` expected and the total number of nodes type in the category, the number of bits representing the `nodeType` is obtained (this number is given in the NDT tables in node coding tables in electronic attachment).
3. When greater than 0 (1 if nodeGroup is 2), the `nodeType` gives the nature of the node to be parsed.

EXAMPLE — The **Shape** node has 2 fields defined as:

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   exposedField SFAppearanceNode        Appearance         NULL
   exposedField SFGeometry3DNode        geometry           NULL
```

When decoding a **Shape** node, if the first field is transmitted, a node of type `SFAppearanceNode` is expected. The only node with `SFAppearanceNode` type is the **Appearance** node, and hence the `nodeType` can be coded using 0 bits. When decoding the **Appearance** node, the following fields can be found:

```
   exposedField SFMaterialNode          Material          NULL
   exposedField SFTextureNode           texture           NULL
   exposedField SFTextureTransformNode  TextureTransform  NULL
```

8.7.4 MaskNodeDescription

8.7.4.1 Syntax

```
class MaskNodeDescription(NodeData node) {
   if (node.protoData != null) {
      for (i=0; i<node.numALLfields; i++) {
         bit(1) Mask;
         if (Mask) {
            bit(1) isedField;
            if (isedField) {
              unsigned int(node.proto.nALLbits) protoField;
            } else {
              Field value(node.field[i]], node.protoData);
            }
         }
      }
   } else { //regular list of fields - not from a PROTO
      for (i=0; i<node.numDEFfields; i++) {
         bit(1) Mask;
         if (Mask) {
            Field value(node.field[node.def2all[i]], node.protoData);
         }
      }
   }
}
```

8.7.4.2 Semantics

If the encoded node is a PROTO then all the fields are scanned. Those that have a `Mask` value of 1 either have a value read in or are ISed fields indicated by `isedField`. The ISed fields read a reference to the PROTO interface field to which they refer.

If the encoded node is not a PROTO, then in the `MaskNodeDescription`, a mask indicates, for each "def" mode field (those having a `defID`) of this node type, if the field value is specified. Fields are sent in the order indicated in node coding tables in electronic attachment . The field types are thus known and permit the field's value to be decoded.

8.7.5 ListNodeDescription

8.7.5.1 Syntax

```
class ListNodeDescription (NodeData node) {
   bit(1) endFlag;
   while (!EndFlag){
      if (node.protoData != null ) {
         bit(1) isedField;
         if (isedField){
            bit(node.nALLbits) fieldRef;
            bit(node.proto.nALLbits) protoField;
         } else {
            bit(node.nDEFbits) fieldRef;
            Field value(node.field[node.def2all[fieldRef]], node.protoData);
         }
      }
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      else {
            bit(node.nDEFbits) fieldRef;
            Field value(node.field[node.def2all[fieldRef]], node.protoData);
      }
      bit(1) endFlag;
   }
}
```

8.7.5.2 Semantics

In the `ListNodeDescription`, fields are directly addressed by their field reference, `fieldRef`. The reference is sent as a `defID` and its parsing depends on the node type (see 8.2.3). When the fields belong to a PROTO, they may be ISed fields, indicated by `isedField`. In this case, a reference to the PROTOinterface in coded in `protoField`. A field may appear several times if ISed by different fields of the `PROTOInterface`. Since all fields may be ISed, PROTO field references are encoded using `node.nALLbits`, where as normal node field references are encoded using only `node.nDEFbits`. Fields that are not Ised may have a default value assigned to them.

Non-PROTO fields always have a default value coded.

8.7.6 Field

8.7.6.1 Syntax

```
class Field(FieldData field, NodeData protoNodeData) {
   if (isSF(field))
      SFField svalue(field, protoNodeData);
   else {
      if (BIFSConfig.usePredictiveMFField == 1) {
         bit(1) usePredictive;
         if (usePredictive)
            PredictiveMFField mvalue(field);
         else
            MFField mvalue(field, protoNodeData);
      } else {
         MFField mvalue(field, protoNodeData);
      }
   }
}
```

8.7.6.2 Semantics

A field is encoded according to its type: single (SFField) or multiple (MFField). A multiple field is a collection of single fields.

8.7.7 MFField

8.7.7.1 Syntax

```
class MFField(FieldData field, NodeData protoNodeData)  {
   bit(1) reserved;
   if (!reserved) {
      bit(1) isListDescription;
      if (isListDescription)
         MFListDescription lfield(field, protoNodeData);
      else
         MFVectorDescription vfield(field, protoNodeData);
   }
}
```

8.7.7.2 Semantics

The bit `reserved` is reserved for future extension. The bit shall be set to 0.

MFField types can be encoded with a list (`MFListDescription`) or vector (`MFVectorDescription`) description.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.7.8 MFListDescription

8.7.8.1 Syntax

```
class MFListDescription(FieldData field, NodeData protoNodeData)  {
   bit(1) endFlag;
   while (!endFlag) {
      SFField field(field, protoNodeData);
      bit(1) endFlag;
   }
}
```

8.7.8.2 Semantics

The MFField type is encoded as a list of single fields.

8.7.9 MFVectorDescription

8.7.9.1 Syntax

```
class MFVectorDescription(FieldData field, NodeData protoNodeData) {
   int(5) NbBits;
   int(NbBits)  numberOfFields;
   SFField field[numberOfFields](field, protoNodeData);
}
```

8.7.9.2 Semantics

The MFField type is encoded as a vector of fields whose dimension is specified.

The number of bits, `NbBits`, used to specify the dimension of the vector is first coded. The actual dimension is then coded as an unsigned integer using `NbBits`. The fields are then coded in order.

8.7.10 PredictiveMFField

8.7.10.1 Syntax

```
class PredictiveMFField (FieldData field) {
   AnimFieldQP  aqp = new AnimFieldQP();
   aqp.useDefault = FALSE;
   field.aqp = aqp;
   ArrayHeader  header(field);
   ArrayOfValues   values(field);
}
```

8.7.10.2 Semantic

The array of data is composed of an `ArrayHeader`, and an `ArrayOfValues`. Note that the `FieldData` structure is filled as descrtibed in the BIFS-Scene quantization process (subclause 8.3.1).

The process applied for optimal encoding of BIFS multiple fields is exactly identical to the BIFS-Anim process (See Table 84):

- Compensation on the P values;
- Inverse Quantization into single field values.

The compensation process uses the quant type as well as `Pmin` and `PNbBits`, defined in the `ArrayQP` and `InitialArrayQP`, and is summarized in Table 84.

The inverse quantization process uses the values of `floatMax`, `floatMin`, and `NbBit` as defined in the BIFS quantization process and as defined by the **QuantizationParameter** node.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.7.11 ArrayHeader

8.7.11.1 Syntax

```
class ArrayHeader(FieldData field){
   uint(5) NbBits;
   int(NbBits) numberOfFields;
   bit(2) intraMode;
   InitialArrayQP  qp(intraMode,field);
}
```

8.7.11.2 Semantic

The array header contains first information to specify the number of fields (`NbBits` is the number of bits used to code the `numberOfFields`). Then the Intra/Predictive policy (`intraMode`) is specified as follows:

0 : Only one Intra value at the beginning and then only predictive coded values;

1 : An Intra every given number of predictive values;

2 : A bit for each value to determine whether the value is an Intra or predictive value.

Lastly, the `InitialArrayQP` is coded.

8.7.12 InitialArrayQP

8.7.12.1 Syntax

```
class InitialArrayQP(int intraMode, FieldData field){
   switch (intraMode) {
      case 1 :
         int(5)     NbBits;
         int(NbBits)  intraInterval;
         // no break
      case 0 :
      case 2 :
         int(5) CompNbBits;
         for (int i=0;i< getNbBound(field.quantType);i++) {
              int(field.NbBits+1) vq;
              field.aqp.Pmin[i] = vq-2^field.NbBits;
         }
         // no break
      case 3:
         break;
   }
}
```

8.7.12.2 Semantic

If `intraMode` is 1, the size of the interval between two intras is first specified. Independent of the `intraMode`, the number of Bits used in Predictive mode `CompNbBits` and the `CompMins` are coded. The function `getNbBound()` is a function that returns the number of components of the quantizing bounds, and depends on the object. For instance it returns 4 for 4D positions, 3 for 3D positions, 2 for 2D positions, and 3 for rotations. See Table 71. The values `CompNbBits` and `CompMin` are stored in the `field.aqp AnimationQP` structure and are used for the compensation process as defined in Table 84 and subclause 8.4 .

8.7.13 ArrayQP

8.7.13.1 Syntax

```
class ArrayQP(int intraMode, FieldData field){
   switch (intraMode) {
      case 1 :
         int(5) NbBits;
         int(NbBits) intraInterval;
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        // no break
      case 0 :
      case 2 :
        boolean(1) hasCompNbBits
        if (hasCompNbBits) {
          int(5) CompNbBits;
        }
        boolean(1) hasCompMin
        if (hasCompMin) {
          for (int i=0;i< getNbBound(field.quantType); ++i) {
               int(field.NbBits+1) vq;
               field.aqp.Pmin[i] = vq-2^field.NbBits;
          }
        }
      case 3:
        break;
   }
}
```

8.7.13.2 Semantic

`ArrayQP` fulfills the same purpose as `InitialArrayQP`, but in this case, the parameters are optionaly set. If they are not set in the stream, they are set by default, in reference to the `InitialArrayQP` or the latest received value of the parameter.

If `IntraMode` is 1, the size of the interval between two intras is first specified. In any case, the number of Bits used in Predictive mode (`CompNbBits`) and the `CompMins` are coded. . The function `getNbBound()` is a function that returns the number of components of the quantizing bounds, and depends on the object. For instance it returns 4 for 4D positions, 3 for 3D positions, 2 for 2D positions, and 3 for rotations. See Table 71. The values `CompNbBits` and `CompMin` are stored in the `field.aqp AnimationQP` structure, and are used for the compensation process as defined in Table 84 and subclause 8.4.

Predictive encoding of 4D values is done per component, extending the scheme for 3D values. As for BIFS-Commands and BIFS-Anim, **position3D** and **scale** quantizers parameters are used.

8.7.14 ArrayOfValues

8.7.14.1 Syntax

```
class ArrayOfValues(FieldData field) {
   ArrayIValue value[0];
   for (int i=1; i < numberOfFields;i++) {
      Switch (intraMode) {
        case  0:
          ArrayPValue value(field);
          break;
        case 1:
          if ( (i % intraInterval) == 0) {
            bit(1) hasQP;
            if (hasQP) {
              ArrayQP qp(field);
            }
            ArrayIValue value(field);
          } else {
            ArrayPvalue value(field);
          }
          break;
        case 2:
          bit (1) isIntra;
          if (isIntra) {
            bit(1) hasQP;
            if (hasQP) {
              ArrayQP qp(field);
            }
            ArrayIValue value;
          } else {
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            ArrayPvalue value;
         }
         break;
   }
}
```

8.7.14.2 Semantic

The array of values first codes a first intra value, and then, according to the `IntraMode`, codes Intra or Predictive values. In Predictive-only mode, no more Intra values are coded. In the second mode, an Intra is sent every `intraInterval` values. In the third mode, an `isIntra` bit selects between Predictive and Intra mode at each value. In that case, an `ArrayQP` can be sent for Intra values. If an `ArrayQP` is sent, the statistics of the arithmetic encoder are reset.

8.7.15 ArrayIValue

8.7.15.1 Syntax

```
class ArrayIValue(FieldData field) {
   switch (field.quantType) {
   case  9:   // Normal
      int(1) direction
   case 10:   // Rotation
      int(2) orientation
      break;
   default:
      break;
}
   for (j=0;j<getNbComp(field);j++)
      int(field.nbBits)  vq[j];
}
```

8.7.15.2 Semantic

The `ArrayIValue` represents the quantized intra value of a field. The value is coded following the quantization process described in the quantization section, and according to the type of the field. For normals the direction and orientation values specified in the quantization process are first coded. For rotations only the orientation value is coded. If the bit representing the direction is 0, the normal's direction is set to 1, if the bit is 1, the normal's direction is set to –1. The value of the orientation is coded as an unsigned integer using 2 bits. The compressed components `vq[i]` of the field's value are then coded as a sequence of unsigned integers using the number of bits specified in the field data structure. The decoding process in intra mode computes the animation values by applying the inverse quantization process.

8.7.16 ArrayPValue

8.7.16.1 Syntax

```
class ArrayPValue(FieldData field) {
   switch (field.quantType) {
      case  9:   // Normal
         int(1)     inverse
         break;
      default:
         break;
   }
   for (j=0;j<getNbComp(field);j++) {
      int(aacNbBits)  vqDelta[j];
   }
}
```

8.7.16.2 Semantic

The `ArrayPValue` represents the difference between the previously received quantized value and the current quantized value of a field. The value is coded using the compensation process as described above.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The values are decoded from the adaptive arithmetic coder bitstream with the procedure `v_aac = aa_decode(model)`. The model is updated with the procedure `model_update(model, v_aac)`. For normals the inverse value is decoded through the adaptive arithmetic coder with a uniform, non-updated model. The compensation values `vqDelta[i]` are then decoded one by one. Let `vq(t-1)` the quantized value decoded at the previous frame and `v_aac(t)` the value decoded by the frame's Adaptive Arithmetic Decoder at instant `t` with the field's models. The value a time `t` is obtained from the previous value as follows :

```
vDelta(t) = v_acc(t) + Pmin
vq(t) = AddDelta(vq(t-1), vDelta(t))
v(t) = InvQuant(vq(t))
```

The field's models are updated each time a value is decoded through the adaptive arithmetic coder. If the animType is 1 (Position3D) or 2 (Position2D), each component of the field's value is using its own model and offset `PMin[i]`. In all other cases the same model and offset `PMin[0]` is used for all the components.

8.7.17 SFField

8.7.17.1 Syntax

```
class SFField(FieldData field, NodeData protoNodeData) {
   switch (field.fieldType) {

      case SFNodeType:
         SFNode nValue(field.fieldType, protoNodeData);
         break;

      case SFBoolType:
         SFBool bValue;
         break;

      case SFColorType:
         SFColor cValue(field);
         break;

      case SFFloatType:
         SFFloat fValue(field);
         break;

      case SFInt32Type:
         SFInt32 iValue(field);
         break;

      case SFRotationType:
         SFRotation rValue(field);
         break;

      case SFStringType:
         SFString sValue;
         break;

      case SFTimeType:
         SFTime tValue;
         break;

      case SFUrlType:
         SFUrl uValue;
         break;

      case SFVec2fType:
         SFVec2f v2Value(field);
         break;

      case SFVec3fType:
         SFVec3f v3Value(field);
         break;
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      case SFVec4fType:
         SFVec4f v4Value(field);
         break;

      case SFImageType:
         SFImage imageValue(field);
         break;

      case SFCommandBufferType:
         SFCommandBuffer commandValue(field);
         break;

      case SFScriptType:
         SFScript scriptValue(protoNodeData);
         break;
   }
}
```

8.7.17.2 Semantics

Each field is encoded according to its `fieldType`.

8.7.18 GenericFloat

8.7.18.1 Syntax

```
class GenericFloat(boolean useEfficientCoding) {
   if (!useEfficientCoding) {
      float(32) value;
   } else {
      EfficientFloat  value;
   }
}
```

8.7.18.2 Semantics

If the parameter `useEfficientCoding` is true, the float is coded using the `EfficientFloat` scheme. Otherwise, the IEEE 32 bit format for float coding is used.

8.7.19 EfficientFloat

8.7.19.1 Syntax

```
class EfficientFloat {
   unsigned int(4) mantissaLength;
   if (mantissaLength != 0) {
      int(3) exponentLength;
      int(1) mantissaSign;
      int(mantissaLength-1) mantissa;
      if (exponentLength != 0) {
         int(1) exponentSign;
         int(exponentLength-1) exponent;
      }
   }
}
```

8.7.19.2 Semantics

For floating point values it is possible to use a more economical representation than the standard 32-bit format, as specified in the `EfficientFloat` structure. This representation separately encodes the size of the exponent (base 2) and mantissa of the number.

If the `mantissaLength` is 0, the decoded value is 0 and further parameters are not coded.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If the mantissaLength is not 0, the exponentLength, mantissaSign and mantissa are coded. The mantissa sign is 1 when the mantissa is negative, otherwise it is 0.

The mantissa syntax element contains the actual mantissa with the leading 1 removed, hence only (mantissaLength-1) bits are needed to encode it.

If the exponentLength is 0 then exponent is not parsed, and the decoded exponent is set, by default, to 0. Otherwise, the sign is read, with exponentSign=1 used to denote a negative exponent. The leading 1 of the exponent is not coded, so that exponent can be encoded using exponentLength-1 bits.

The actual mantissa and exponent are, respectively, ($2^{\text{mantissaLength-1}}$ + mantissa) and

($2^{\text{exponentLength-1}}$ + exponent), thus in all other cases the decoded value shall be:

$$(1-2.\text{mantissaSign}).(2^{\text{mantissaLength-1}}+\text{mantissa}).2^{(1-2.\text{exponentSign}).(2^{\text{exponentLength}-1}+\text{exponent})}$$

8.7.20 SFBool

8.7.20.1 Syntax

```
class SFBool {
   bit(1) value;
}
```

8.7.20.2 Semantics

If value is 1 the decoded boolean is set to TRUE. If value is 0, the decoded boolean is set to FALSE.

8.7.21 SFColor

8.7.21.1 Syntax

```
class SFColor(FieldData field) {
   if (field.isQuantized) {
      QuantizedField qvalue(field);
   } else {
      GenericFloat rValue(field.useEfficientCoding);
      GenericFloat gValue(field.useEfficientCoding);
      GenericFloat bValue(field.useEfficientCoding);
   }
}
```

8.7.21.2 Semantics

If the field's isQuantized bit is TRUE, the QuantizedField scheme shall be used. Otherwise each component of the SFColor is coded using the GenericFloat scheme.

8.7.22 SFFloat

8.7.22.1 Syntax

```
class SFFloat(FieldData field) {
   if (field.isQuantized) {
      QuantizedField qvalue(field);
   } else{
      GenericFloat value(field.useEfficientCoding);
   }
}
```

8.7.22.2 Semantics

If the field's isQuantized bit is TRUE, the QuantizedField scheme shall be used. Otherwise the SFFloat is coded using the GenericFloat scheme.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.7.23 SFInt32

8.7.23.1 Syntax

```
class SFInt32(FieldData field) {
   if (field.isQuantized) {
      QuantizedField qvalue(field);
   } else {
      int(32) value;
   }
}
```

8.7.23.2 Semantics

If the field's `isQuantized` bit is TRUE, the `QuantizedField` scheme shall be used. Otherwise the SFInt32 is coded as a signed value using 32 bits.

8.7.24 SFRotation

8.7.24.1 Syntax

```
class SFRotation(FieldData field) {
   if (field.isQuantized) {
      QuantizedField qvalue(field);
   } else {
      GenericFloat xAxis(field.useEfficientCoding);
      GenericFloat yAxis(field.useEfficientCoding);
      GenericFloat zAxis(field.useEfficientCoding);
      GenericFloat angle(field.useEfficientCoding);
   }
}
```

8.7.24.2 Semantics

If the field's `isQuantized` bit is TRUE, the `QuantizedField` scheme shall be used. Otherwise each component of the SFRotation is coded indepedently using the `GenericFloat` scheme.

8.7.25 SFString

8.7.25.1 Syntax

```
class SFString {
   unsigned int(5) lengthBits;
   unsigned int(lengthBits) length;
   char(8) value[length];
}
```

8.7.25.2 Semantics

The `SFString` is coded as an array of characters whose length is first specified.

`lengthBits` is the number of bits used to encode the string length.

`length` is the length of the string coded using `lengthBits`.

All characters are coded using the UTF-8 character encoding (ISO/IEC 10646-1).

8.7.26 SFTime

8.7.26.1 Syntax

```
class SFTime {
   double(64) value;
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
}
```

8.7.26.2 Semantics

The SFTime value is coded as a 64-bit double.

8.7.27 SFUrl

8.7.27.1 Syntax

```
class SFUrl {
   bit(1) isOD;
   if (isOD) {
      bit(10) ODid;
   } else {
      SFString urlValue;
   }
}
```

8.7.27.2 Semantics

The “od:” URL scheme is used in an **url** field of a BIFS node to refer to an object descriptor. The integer immediately following the “od:” prefix identifies the `ObjectDescriptorID`. For example, “od:12” refers to object descriptor number 12.

If the `SFUrl` refers to an object descriptor, the `ObjectDescriptorID` is coded as a 10-bit integer. If the `SFUrl` refers to a segment of a media stream (“od:12#<segmentName>”) and in all other cases the URL is sent as an `SFString`.

8.7.28 SFVec2f

8.7.28.1 Syntax

```
class SFVec2f(FieldData field) {
   if (field.isQuantized) {
      QuantizedField qvalue(field);
   } else {
      GenericFloat value1;
      GenericFloat value2;
   }
}
```

8.7.28.2 Semantics

If the field’s `isQuantized` bit is TRUE, the `QuantizedField` scheme shall be used. Otherwise each component of the SFVec2f is coded using the `GenericFloat` scheme.

8.7.29 SFVec3f

8.7.29.1 Syntax

```
class SFVec3f(FieldData field) {
   if (field.isQuantized) {
      QuantizedField qvalue(field);
   } else {
      GenericFloat value1(field.useEfficientCoding);
      GenericFloat value2(field.useEfficientCoding);
      GenericFloat value3(field.useEfficientCoding);
   }
}
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.7.29.2 Semantics

If the field's `isQuantized` bit is TRUE, the `QuantizedField` scheme shall be used. Otherwise each component of the SFVec3f is coded using the `GenericFloat` scheme.

8.7.30 SFVec4f

8.7.30.1 Syntax

```
class SFVec4f(FieldData field) {
   if (field.isQuantized)
      QuantizedField qvalue(field);
   else {
      GenericFloat value1(field.useEfficientCoding);
      GenericFloat value2(field.useEfficientCoding);
      GenericFloat value3(field.useEfficientCoding);
      GenericFloat value4(field.useEfficientCoding);
   }
}
```

8.7.30.2 Semantics

An SFVec4f field typically holds of a 4-Dimensional vector that consists of 4 values (x, y, z, w), where (x ,y ,z) are the 3D coordinates of the vector and w is a scaling factor to represent the infinity.

If the field's `isQuantized` bit is TRUE, the `QuantizedField` scheme shall be used. Otherwise each component of the SFVec4f is coded using the `GenericFloat` scheme.

8.7.31 SFImage

8.7.31.1 Syntax

```
class SFImage {
   unsigned int(12)   width;
   unsigned int(12)   height;
   bit(2)  numComponents;
   bit(8)  pixels[(numComponents+1)*width*height];
}
```

8.7.31.2 Semantics

The `width` and `height` in pixels of the image are coded as 12-bit unsigned integers.

`numComponents` defines the image type. The following types are permitted:

If the value is '00', then a grey scale image shall be decoded.

If the value is '01', then a grey scale with alpha channel shall be decoded.

If the value is '10', then an RGB image shall be decoded.

If the value is '11', then an RGB image with alpha channel shall be decoded.

The pixels array stores the pixel values in row order from left to right and bottom to top. Each pixel being stored in network byte order. Pixels shall be decoded as unsigned char, 8-bit encoded values for each of its components. Components are coded in the orders of grey, grey alpha, R G B or R G B alpha for the four numComponent image type values 0 through 3 respectively.

8.7.32 SFCommandBuffer

8.7.32.1 Syntax

```
class SFCommandBuffer {
   unsigned int(5) lengthBits;
   unsigned int(lengthBits) length;
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   bit(8) value[length];
}
```

8.7.32.2 Semantics

The `SFCommandBuffer syntax element` is coded as an array of bytes whose length is first specified.

`lengthBits` is the number of bits used to encode the buffer length.

`length` is the length of the buffer coded using `lengthBits`.

`value` is an array of bytes of length `length`. It shall contain a `CommandFrame`, padded if necessary to complete the last byte.

8.7.33 QuantizedField

8.7.33.1 Syntax

```
class QuantizedField(FieldData field) {
   switch (field.quantType) {
      case 9:
         int(1) direction
      case 10:
         int(2) orientation
      default:
         break;
    }
    for (i=0;i<getNbComp(field);i++) {
      int(field.nbBits) vq[i];
    }
}
```

8.7.33.2 Semantics

The value is quantized using the quantization process described in subclause 8.3.

For normals, the direction and orientation values specified in the quantization process are first coded. For rotations, only the orientation value is coded.

The compressed components, `vq[i]`, of the field's value are then coded in sequence as unsigned integers using the number of bits specified in the field data structure.

8.7.34 SFScript

8.7.34.1 Syntax

```
class SFScript(NodeData protoNodeData) {
   bit(1) isListDescription;
   if (isListDescription) {
      ScriptFieldsListDescription(protoNodeData);
   } else {
      ScriptFieldsVectorDescripion(protoNodeData);
   }
   const bit(1) reserved=1;
   EncodedScript();
}
```

8.7.34.2 Semantics

The SF`Script` class is used to represent a **Script** node. This can be done as a list description or as a vector description, depending on the value in `isListDescription`. The script is encoded using the bitstream syntax for `EncodedScript`, given below. This bitstream is a tree representation of the BNF grammar for ECMAScript (ISO/IEC 16262). Each node determines the parse decision selected in parsing the script, and thus the resulting bitstream can be used to interpret the script directly.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.7.35 ScriptFieldsListDescription

8.7.35.1 Syntax

```
class ScriptFieldsListDescription(NodeData protoNodeData) {
   bit(1) endFlag; // List description of the fields
   while (!EndFlag) {
      ScriptField(protoNodeData);
      bit(1) endFlag;
   }
}
```

8.7.35.2 Semantics

`ScriptFieldsListDescription` reads a list description of the fields in the **Script** node. When `endFlag` has value 1, the list has ended and no more values are read.

8.7.36 ScriptFieldsVectorDescription

8.7.36.1 Syntax

```
class ScriptFieldsVectorDescription(NodeData protoNodeData) {
   bit(4) fieldBits;  // Number of bits for number of fields
   bit(fieldBits)  numFields; // Number of fields in the script
   for (i=0; i<numFields; ++i) {
      ScriptField(protoNodeData);
   }
}
```

8.7.36.2 Semantics

`ScriptFieldsVectorDescription` reads a value `numFields`, to determine how many fields are in the **Script** node, and these are read sequentially. The number of bits used to give the number of fields is first read as 4 bits in `fieldBits`.

8.7.37 ScriptField

8.7.37.1 Syntax

```
class ScriptField(NodeData protoNodeData) {
   bit(2) eventType;
   bit(6) fieldType;
   String fieldName;
   if (protoNodeData != null ) {
    bit(1) isedField;
    if (isedField){
      bit(protoNodeData.nALLbits) protoField;
    } else {
      if (eventType == FIELD) {
        bit(1) hasInitialValue;
        if (hasInitialValue){
          FieldData fieldData = MakeField(fieldType);
          Field value(fieldData, protoNodeData);
        }
      }
    }
  } else {
    if (eventType == FIELD) {
      bit(1) hasInitialValue;
      if (hasInitialValue){
          FieldData fieldData = MakeField(fieldType);
          Field value(fieldData, protoNodeData);
      }
    }
  }
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
}
```

8.7.37.2 Semantics

The `ScriptField` contains one field for the **Script** node. The `eventType` specifies the type of field, with values 0, 1, and 2 representing fields, eventIns and eventOuts, respectively. The `fieldType` is given in Table 86. This determines the type of the field. The fieldName gives the name of this field; the name is used to refer to this field from within the script.

When the event is a field, it may have a default value. This presence of this value is indicated by `hasInitialValue` being 1. In this case, the field value is read using the `Field` class. In order to be able to use the `Field` class, a node of type `NodeData` is created that then has the appropriate field value for each `fieldType` (the `fieldType` index can be used to reference field structures of the appropriate type).

Table 86 — Field Types for Script fields and PROTO fields.

`fieldType` value	Field type
0bx000000	SFBool
0bx000001	SFFloat
0bx000010	SFTime
0bx000011	SFInt32
0bx000100	SFString
0bx000101	SFVec3f
0bx000110	SFVec2f
0bx000111	SFColor
0bx001000	SFRotation
0bx001001	SFImage
0bx001010	SFNode
0bx100000	MFBool
0bx100001	MFFloat
0bx100010	MFTime
0bx100011	MFInt32
0bx100100	MFString
0bx100101	MFVec3f
0bx100110	MFVec2f
0bx100111	MFColor
0bx101000	MFRotation
0bx101001	MFImage
0bx101010	MFNode
0bx101011	SFVec4f
0bx101100	MFVec4f

8.7.38 EncodedScript

8.7.38.1 Syntax

```
class EncodedScript {
   bit(1) hasFunction
   while (hasFunction) {
      Function function;
      bit(1) hasFunction
   }
}
```

8.7.38.2 Semantics

A script is a collection of functions, listed sequantially while `hasFunction` is TRUE.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.7.39 Function

8.7.39.1 Syntax

```
class Function {
   Identifier identifier;
   Arguments arguments;
   StatementBlock statementBlock;
}
```

8.7.39.2 Semantics

Each function consists of an `identifier`, a list of `arguments`, and a `statementBlock` which contains the script statements executed when the function is called.

8.7.40 Arguments

8.7.40.1 Syntax

```
class Arguments {
   bit(1) hasArgument
   while (hasArgument) {
      Identifier identifier;
      bit(1) hasArgument
   }
}
```

8.7.40.2 Semantics

The argument list is of arbitrary length, and terminates when `hasArgument` is 0. Each argument consists of one `identifier`.

8.7.41 StatementBlock

8.7.41.1 Syntax

```
class StatementBlock {
   bit(1) isCompoundStatement
   if (isCompoundStatement) {
      bit(1) hasStatement
      while (hasStatement) {
         Statement statement;
         bit(1) hasStatement
      }
      else {
         Statement statement;
      }
   }
}
```

8.7.41.2 Semantics

A `statementBlock` consists of either a `compoundStatement`, which holds several script statements, or a single statement, indicated by the value of `isCompoundStatement`. When the `statementBlock` consists of several statements, the `hasStatement` bit is used to signal either the end of the list or the existance of another statement.

8.7.42 Statement

8.7.42.1 Syntax

```
class Statement {
   bit(3) statementType
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   switch statementType {
      case ifStatementType:
         IFStatement ifStatement;
         break;
      case forStatementType:
         FORStatement forStatement;
          break;
      case whileStatementType:
         WHILEStatement whileStatement;
         break;
      case returnStatementType:
         RETURNStatement returnStatement;
         break;
      case compoundExpressionType:
         CompoundExpression compoundExpression;
         break;
      case breakStatementType:
      case continueStatementType:
           break;
      case switchStatementType:
           SWITCHStatement switchStatement;
           break;
   }
}
```

8.7.42.2 Semantics

A `Statement` may consist of one of the following specific statement types:

`ifStatement`

`forStatement`

`whileStatement`

`returnStatement`

`compoundExpression`

`breakStatement`

`continueStatement`.

`switchStatement`.

These statement types are indicated by a value from 0-7, respectively, called `statementType`.

8.7.43 IFStatement

8.7.43.1 Syntax

```
class IFStatement {
   CompoundExpression compoundExpression;
   StatementBlock statementBlock;
   bit(1) hasELSEStatement
   if (hasELSEStatement) {
      StatementBlock statementBlock;
   }
}
```

8.7.43.2 Semantics

An `IFStatement` is used for conditional execution of a `statementBlock`. It consists of a `CompoundExpression` followed by a `statementBlock`. The `statementBlock` is interpreted when the `CompoundExpression` evaluates to a non-zero or non-empty value. The `IFStatement` has an optional additional `statementBlockwhich` is included when

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

hasElseStatement is 1. This second, optional compoundStatement is interpreted when the CompoundExpression evaluates to a zero or empty value.

8.7.44 FORStatement

8.7.44.1 Syntax

```
class FORStatement {
   OptionalExpression optionalExpression;
   OptionalExpression optionalExpression;
   OptionalExpression optionalExpression;
   StatementBlock statementBlock;
}
```

8.7.44.2 Semantics

A FORStatement is used to iterate over values, stopping when a conditional expression fails. The first optionalExpression shall be executed when the statement is interpretted. The second optionalExpression shall then be evaluated, and if it returns a non-zero or non-empty value, the statementBlock shall be executed. The third optionalExpression shall then be executed. After this process shall repeat starting with the execution of the second optionalExpression again, the statementBlock, and the third optionalExpression.

8.7.45 WHILEStatement

8.7.45.1 Syntax

```
class WHILEStatement {
   CompoundExpression compoundExpression;
   StatementBlock statementBlock;
}
```

8.7.45.2 Semantics

The WHILEStatement is used to conditionally execute a statementBlock for so long as the compoundExpression evaluates to a non-zero or non-empty value.

8.7.46 RETURNStatement

8.7.46.1 Syntax

```
class RETURNStatement {
   bit(1) returnValue
   if (returnValue) {
      CompoundExpression compoundExpression;
   }
}
```

8.7.46.2 Semantics

The RETURNStatement is used to return a value from a function. When a function has no return value, returnValue shall be 0. Otherwise, the returned value shall be the last value evaluated for compoundExpression.

8.7.47 CompoundExpression

8.7.47.1 Syntax

```
class CompoundExpression {
   do {
      Expression expression;
      bit(1) hasExpression
   } while (hasExpression);
}
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.7.47.2 Semantics

A `CompoundExpression` is a list of `expressions`, terminated when `hasExpression` has value 0. The value of the compound expression shall be the value of the last evaluated expression.

8.7.48 SWITCHStatement

8.7.48.1 Syntax

```
class SWITCHStatement {

     CompoundExpression compoundExpression; // the switched value
     bit(5) numbits // number of bits for the case value
   do {
     bit(numbits) caseValue; #a case value
     StatementBlock statementBlock; // statements  in case
     bit(1) hasMoreCases
   } while (hasMoreCases);
   bit(1) hasDefault;
   if (hasDefault) {
     StatementBlock statementBlock; // default statements in case
   }
}
```

8.7.48.2 Semantics

A `SWITCHStatement` is an expression that must evaluate to an integer value. It is followed by pairs of integer values in `value` stored with `numbits` bits and `StatementBlock`s. The values represent the value of a case statement, which are encoded repeatedly until `hasMoreCases` is 0. An optinal default `StatementBlock` is then encoded.

8.7.49 optionalExpression

8.7.49.1 Syntax

```
class optionalExpression {
   bit(1) hasCompoundExpression
   if (hasCompoundExpression) {
     CompoundExpression compoundExpression;
   }
}
```

8.7.49.2 Semantics

An `optionalExpression` may be an empty expression, containing no executable statements, or a `compoundExpression`. This is indicated by the value of `hasCompoundExpression`.

8.7.50 Expression

8.7.50.1 Syntax

```
class Expression {
   bit(6) expressionType
   switch expressionType {
     case curvedExpressionType:          // (compoundExpression)
       CompoundExpression compoundExpression;
       break;
     case negativeExpressionType:           // -expression
     case notExpressionType:             // !expression
     case onescompExpressionType:           // ~expression
     case incrementExpressionType:            // ++expression
     case decrementExpressionType:            // --expression
     case postIncrementExpressionType:       // expression++
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      case postDecrementExpressionType:       // expression--
         Expression expression;
         break;
      case conditionExpressionType:   // expression ? expression : expression
         Expression expression;
         Expression expression;
         Expression expression;
         break;
      case stringExpressonType:
         String string;
         break;
      case numberExpressionType:
         Number number;
         break;
      case variableExpressionType:
         Identifier identifier;
         break;
      case functionCallExpressionType:
      case objectConstructExpressionType:
         Identifier identifier;
         Params params;
         break;
      case objectMemberAccessExpressionType:
         Expression expression;
         Identifier identifier;
         break;
      case objectMethodCallExpressionType:
         Expression expression;
         Identifier identifier;
         Params params;
         break;
      case arrayDereferenceExpressionType:
         Expression expression;
         CompoundExpression compoundExpression;
         break;
      case booleanExpressionType:
         Boolean boolean;
         break;
      case varExpressionType:
         Arguments arguments;
         break;
      default: // =, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, >>>=,
         // ==, !=, <, <=, >, >=, +, -, *, /, %, &&, ||, &, |,
         // ^, <<, >>, >>>
         Expression expression;
         Expression expression;
         break;
   }
}
```

8.7.50.2 Semantics

An expression may contain one of a number of possible executed statements, specified by the value in `expressionType`. These are listed below, according the value of `expressionType`.

`curvedExpressionType=0:`

The expression consists of a `compoundExpression`.

`negativeExpressionType=1:`

An `expression` shall be evaluated and the value returned shall be negated.

`notExpressionType=2:`

An `expression` shall be evaluated and its returned value shall be logically negated (empty values return non-empty, zero values return non-zero, and vice-versa).

`onescompExpressionType=3:`

An `expression` shall be evaluated numerically (string values will yield an undefined result) and the value returned shall be bitwise negated.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

incrementExpressionType=4:

An expression shall be evaluated numerically (string values will yield an undefined result) and the value returned shall incremented by 1.

decrementExpressionType=5:

An expression shall be evaluated numerically (string values will yield an undefined result) and the value returned shall be decremented by 1.

postIncrementExpressionType=6:

An expression shall be evaluated numerically (string values will yield an undefined result) and its returned value shall be incremented by 1. The returned value of this expression shall be the value prior to the increment being applied.

postDecrementExpressionType=7:

An expression shall be evaluated numerically (string values will yield an undefined result) and its returned value shall be decremented by 1. The returned value of this expression shall be the value prior to the decrement being applied.

conditionExpressionType=8:

Three expressions shall be evaluated. If the first expression returns a non-zero or non-empty value, then the returned value of this expression shall be the value of the second expression. Otherwise, the returned value of this expression shall be the value of the third expression.

stringExpressonType=9:

The expression contains a string.

numberExpressionType=10:

The expression is a number.

variableExpressionType=11:

The expression is a variable and shall return the value held by the variable determined by identifier.

functionCallExpressionType=12:

An identifier determines which function shall be evaluated. The params shall be passed to the function by value. The returned value of the expression shall be the value returned by the function in its returnStatement.

objectConstructExpressionType=13:

A new object shall be created (using a 'new' statement in the script) and the object shall be held in the variable determined by identifier. A list of params shall be passed to any constructors that exist for the object.

objectMemberAccessExpressionType=14:

A member variable of an object shall be accessed and the returned value of the expression shall be the value in this member variable. Normally, the first expression will evaluate to a node in the scene graph (which is accessed through a script variable). This means that the first expression will normally evaluate to an identifier reference. The following identifier will then refer to a field of the node.

objectMethodCallExpressionType=15:

A method of an object shall be evaluated. The first expression shall evaluate to an object. The following identifier shall specify a method of this object. The following params shall be passed to the method. The value of this expression shall be the value returned by the method.

arrayDereferenceExpressionType=16:

The expression shall be an array element reference. The first expression shall evaluate to an array reference. The following compoundExpression shall evaluate to a number that shall then be used to index the array. The returned value of this expression shall be the value held in the referenced array element.

The following binary operands evaluate two expressions and return a value based on a binary operation of these two expressions. The binary operation and value of expressionType is listed below for each binary operation. Unless explicitely stated, a string value for either of the expressions will yield an undefined result.

BinaryOperand(=) = 17:

The first expression shall evaluate to an identifier which shall be assigned the value of the second expression.

BinaryOperand(+=) = 18:

The first expression shall evaluate to an identifier. If the value held by the variable is numerical, the variable value shall be incremented by the value of the second expression, which shall also evaluate to a numerical value. If the variable is a string, then its new value shall be its original value with the second expression (which shall be a string) appended.

BinaryOperand(-=) = 19:

The first expression shall evaluate to an identifier whose value shall be decremented by the value of the second expression.

BinaryOperand(*=) = 20:

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The first `expression` shall evaluate to an `identifier` whose value shall be set to its current value multiplied by the value of the second `expression`.

`BinaryOperand(/=) = 21:`

The first `expression` shall evaluate to an `identifier` whose value shall be set to its current value divided by the value of the second `expression`.

`BinaryOperand(%=) = 22:`

The first `expression` shall evaluate to an `identifier` whose value shall be set to its current value modulo the value of the second `expression`. The expressions shall both evaluate to integer values.

`BinaryOperand(&=) = 23:`

The first `expression` shall evaluate to an `identifier` whose value shall be set to its current value logically bitwise ANDed with the value of the second `expression`.

`BinaryOperand(|=) = 24:`

The first `expression` shall evaluate to an `identifier` whose value shall be set to its current value logically bitwise ORed with the value of the second `expression`.

`BinaryOperand(^=) = 25:`

The first `expression` shall evaluate to an `identifier` whose value shall be set to its current value logically bitwise EXCLUSIVE-ORed with the value of the second `expression`.

`BinaryOperand(<<=) = 26:`

The first `expression` shall evaluate to an `identifier` whose value shall be set to its current value bitwise shifted to the left a number of bits specified by the second `expression`.

`BinaryOperand(>>=) = 27:`

The first `expression` shall evaluate to an `identifier` whose value shall be set to its current value bitwise shifted to the right a number of bits specified by the second `expression`.

`BinaryOperand(>>>=) = 28:`

The first `expression` shall evaluate to an `identifier` whose value shall be set to its current value bitwise shifted to the right (with the least significant bits looped) a number of bits specified by the second `expression`.

`BinaryOperand(==) = 29:`

This expression shall return a non-zero value when the first and second `expression` are identical. Otherwise, the result of this expression shall be zero.

`BinaryOperand(!=) = 30:`

This expression shall return a non-zero value when the first and second `expression` are not identical. Otherwise, the result of this expression shall be zero.

`BinaryOperand(<) = 31:`

This expression shall return a non-zero value when the first `expression` is numerically or lexicographically less than the second. Otherwise, the result of this expression shall be zero.

`BinaryOperand(<=) = 32:`

This expression shall return a non-zero value when the first `expression` is numerically or lexicographically less than or equal to the second. Otherwise, the result of this expression shall be zero.

`BinaryOperand(>) = 33:`

This expression shall return a non-zero value when the first `expression` is numerically or lexicographically greater than the second. Otherwise, the result of this expression shall be zero.

`BinaryOperan(>=) = 34:`

This expression shall return a non-zero value when the first `expression` is numerically or lexicographically greater than or equal to the second. Otherwise, the result of this expression shall be zero.

`BinaryOperand(+) = 35:`

This expression shall return the sum of the first and second `expressions`. If both `expressions` are strings, then the result shall be the first `string` concatenated with the second.

`BinaryOperand(-) = 36:`

This expression shall return the difference of the first and second `expressions`.

`BinaryOperand(*) = 37:`

This expression shall return the product of the first and second `expressions`.

`BinaryOperand(/) = 38:`

This expression shall returns the quotient of the first and second `expressions`.

`BinaryOperand(%) = 39:`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

This expression shall return the value of the first `expression` modulo the second `expression`.

`BinaryOperand(&&) = 40:`

This expression shall return the logical AND of the first and second `expressions`.

`BinaryOperand(||) = 41:`

This expression shall return the logical OR of the first and second `expressions`.

`BinaryOperand(&) = 42:`

This expression shall return the logical bitwise AND of the first and second `expressions`.

`BinaryOperand(|) = 43:`

This expression shall return the logical bitwise OR of the first and second `expressions`.

`BinaryOperand(^) = 44:`

This expression shall return the logical bitwise XOR of the first and second `expressions`.

`BinaryOperand(<<) = 45:`

This expression shall return the value of the first `expression` shifted to the left by the number of bits specified as the value of the second `expression`.

`BinaryOperand(>>) = 46:`

Returns the value of the first `expression` shifted to the right by the number of bits specified as the value of the second `expression`.

`BinaryOperand(>>>) = 47:`

This expression shall return the value of the first `expression` shifted to the right (with the least significant bit looped to the most significant bit) by the number of bits specified as the value of the second `expression`.

`booleanExpressionType = 48;`

This expression is a boolean value of “TRUE” or “FALSE.”

`varExpressionType = 49;`

This expression hold a ‘var’ expression containing a list of list of variables whose scope is local to the enclosing function. This expression can only appear as part of a top level CompoundExpression.

8.7.51 Params

8.7.51.1 Syntax

```
class Params {
   bit(1) hasParam
   while(hasParam) {
      Expression expression;
      bit(1) hasParam
   }
}
```

8.7.51.2 Semantics

The `Params` class consists of a (possibly empty) list of `expressions`. The `hasParam` bit indicates either the end of the list, or the existance of another `expression`.

8.7.52 Identifier

8.7.52.1 Syntax

```
class Identifier {
   bit(1) received
   if (received) {
      bit(num) identifierCode // num is calculated by counting
         // number of distinguished identifiers
         // received so far
   }
   else {
      String string;
   }
}
```


Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2005 – All rights reserved
Not for Resale

8.7.52.2 Semantics

An `identifier` is used to identify a variable. If the `identifier` has occurred before in the script (or as a field name in the Script node), then an `identifierCode` value is sent using `num` bits. This is indicated by the `received` bit. If the `identifier` has not occured before in the script (or as a field name in the Script node), then an `identifierCode` value is sent using `num` bits. The value of `num`, that is, the number of bits needed to send the index of the identifier in a list of all previousy occuring identifiers, is variable and is determined by the minimum number of bits needed to specify the length of the list of all previously occuring identifiers.

A Script node may identify the field of a node by its name or by its field index coded in ALL mode. When accessing the field by ID, the identifier syntax used is "`_fieldN`", where N is the ALL ID. This allows accessing proto fields in a script without having to encode the scene with the USENAMES flag set.

A Script node may create a new instance of a prototyped node (e.g. instantiate a proto) by using the SFNode constructor with the proto name as an argument, if known, or with the syntax `new_node=new SFNode('_protoZ')` where Z is the protoID as coded in the BIFS stream.

8.7.53 String

8.7.53.1 Syntax

```
class String {
   bit(8) char
   while (char!=0) {
      bit(8) char
   }
}
```

8.7.53.2 Semantics

A `String` type consist of a null-terminated list of 8 bit characters. All characters are coded using the UTF-8 character encoding.

8.7.54 Number

8.7.54.1 Syntax

```
class Number {
   bit(1) isInteger
   if (isInteger) {
      bit(5) numbits // number of bits the integer is represented
      bit(numbits) value // integer value
   }
   else {
      bit(4) floatChar // 0-9, ., E,-, END_SYMBOL
      while (floatChar!=END_SYMBOL) {
         bit(4) floatChar
      }
   }
}
```

8.7.54.2 Semantics

A number shall be represented as an integer, indicated by `isInteger`, or as a list of 4 bit characters, representing (in order) the characters 0 ,1 ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,9, . , E,- , END-SYMBOL. The END-SYMBOL value can be any of 13, 14, or 15, and is used to signal the end of the float value list. The list of characters shall result in a human readable float value in scientific notation.

8.7.55 Boolean

8.7.55.1 Syntax

```
class Boolean {
   bit(1) value
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
}
```

8.7.55.2 Semantics

A Boolean value is represented by a one-bit value.

8.7.56 ROUTEs

8.7.56.1 Syntax

```
class ROUTEs() {
   bit(1) ListDescription;
   if (ListDescription) {
      ListROUTEs lroutes();
   } else {
      VectorROUTEs vroutes();
   }
}
```

8.7.56.2 Semantics

`ROUTEs` may be encoded with a list (`ListROUTEs`) or vector (`VectorROUTEs`) description.

8.7.57 ListROUTEs

8.7.57.1 Syntax

```
class ListROUTEs() {
   do {
      ROUTE route();
      bit(1) moreROUTEs;
   }
   while (moreROUTEs);
}
```

8.7.57.2 Semantics

The ROUTEs are coded as a list, with the `moreROUTEs` flag used to indicate the end of the list (when set to false).

8.7.58 VectorROUTEs

8.7.58.1 Syntax

```
class VectorROUTEs() {
   int(5) nBits;
   int(nBits) length;
   ROUTE route[length]();
}
```

8.7.58.2 Semantics

The ROUTEs are coded as a vector whose dimension, `length`, is first specified.

8.7.59 ROUTE

8.7.59.1 Syntax

```
class ROUTE() {
   bit(1) isUpdateable;
   if (isUpdateable) {
      bit(BIFSConfiguration.routeIDbits) routeID;
      if (USENAMES) {
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        String routeName;
      }
   }

   bit(BIFSConfiguration.nodeIDbits) outNodeID;
   NodeData  nodeOUT = GetNodeFromID(outNodeID);
   int(nodeOUT.nOUTbits) outFieldRef;
   bit(BIFSConfiguration.nodeIDbits) inNodeID;
   NodeData nodeIN = GetNodeFromID(inNodeID);
   int(nodeIN.nINbits) inFieldRef;
}
```

8.7.59.2 Semantics

This is the basic syntax element used to represent a ROUTE. If `isUpdateable` is TRUE ('1') then a `routeID` is sent to enable further reference to this route. Further, if the global value of USENAMES is set, a string name, used by MPEG-J to reference the ROUTE, is also sent.

The ROUTE description is then sent. The `nodeID` of the target node is coded, followed by the target field's `outID`. The `nodeID` of the source node is then coded, followed by the source field's `inID`.

8.8 BIFS-Anim

8.8.1 Overview

The BIFS-Anim session has two parts: the `AnimationMask` and the `AnimationFrames`. The `AnimationMask` specifies the nodes and fields to be animated. It is sent in BIFS configuration, in the object descriptor for the BIFS elementary stream. The animation frames are sent in a separate BIFS stream. When parsing the BIFS-Anim stream, the node structure and related functions as described in node coding tables in electronic attachment are known at the receiving terminal. The decoding data structure `AnimationMask` (see 8.2.5) is constructed when the `AnimationMask` syntax is read, and further used in the decoding process of the BIFS-Anim frames.

`AnimationFrames` contain update information for thevalues of the animated fields described in the `AnimationMask`. They are the access units of the BIFS-Anim stream. An `AnimationFrame` can send information in intra or in predictive mode. In intra mode, the values are quantized and coded directly. In predictive mode, the difference between the quantized value of the current and the last transmitted value of the field are coded. The encoding is performed using an adaptative arithmetic coder described in subclause 8.11.

The use of the adaptive arithmetic coder is as follows:

At the beginning of each predictive frame, the adaptive arithmetic coder is reset. At the end of each frame, it is flushed.

Each animated field has its own set of models. At each intra frame, if the stream has been declared in random access mode (see 8.5.2), the models are reset to the uniform statistics. If the stream is not in random access mode, the models are not reset unless the decoding structures (`AnimQP`) are modified.

8.8.2 AnimationFrame

8.8.2.1 Syntax

```
class AnimationFrame() {
   AnimationFrameHeader header(BIFSConfiguration.animMask);
   AnimationFrameData data(BIFSConfiguration.animMask);
}
```

8.8.2.2 Semantics

The `AnimationFrame` is the access unit of the BIFS-Anim stream. It contains the `AnimationFrameHeader`, which specifies timing, and specifies which nodes are animated in the list of animated nodes, and the `AnimationFrameData`, which contains the data for all nodes being animated.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.8.3 AnimationFrameHeader

8.8.3.1 Syntax

```
class AnimationFrameHeader(AnimationMask mask) {
   bit(23)* next;
   if (next==0) {
      bit(32) AnimationStartCode;
   }

   bit(1) mask.isIntra;
   bit(1) mask.isActive[mask.numNodes];
   if (isIntra) {
      bit(1) isFrameRate;
      if (isFrameRate)
         FrameRate rate;
      bit(1) isTimeCode;
      if (isTimeCode) {
         unsigned int(18) timeCode;
      }
   }
   bit(1) hasSkipFrames;
   if (hasSkipFrames) {
      SkipFrames skip;
   }
}
```

8.8.3.2 Semantics

In the `AnimationFrameHeader`, a start code may be sent at each intra or predictive frame to enable resynchronization. The first 23 bits are read ahead, and stored as the integer `next`.

If `next` is 0 (in other words, the first 23 bits if the `AnimationFrame` are 0), the first 32 bits of the `AnimationFrame` shall be read and interpreted as a start code that precedes the `AnimationFrame`.

The start codes for FBA and Mesh2D are defined in ISO/IEC 14496-2. For generic BIFS-Anim, either no start-code is used or the generic BIFS-Anim start code is used 0xC7 .

If the boolean `isIntra` is TRUE, the current animation frame contains intra-coded values, otherwise it is a predictive frame.

The array of booleans `isActive` specifies which nodes shall be animated for this frame. `isActive` shall contain one boolean for each node in the `AnimationMask`. The boolean is set to TRUE if the node is to be animated; FALSE otherwise.

In intra mode, some additional timing information is also specified. The timing information obeys the syntax of the Facial Animation specification in ISO/IEC 14496-2. Finally, it is possible to skip a number of `AnimationFrames` by using the `FrameSkip` syntax specified in ISO/IEC 14496-2.

8.8.4 FrameRate

8.8.4.1 Syntax

```
class FrameRate {
   unsigned int(8) frameRate;
   unsigned int(4) seconds;
   bit(1) frequencyOffset;
}
```

8.8.4.2 Semantics

`frame_rate` is an 8-bit unsigned integer indicating the reference frame rate of the sequence.

`seconds` is a 4-bit unsigned integer indicating the fractional reference frame rate. The frame rate is computed as follows:

frame rate = (`frame_rate` + `seconds`/16).

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

`frequency_offset` is a 1-bit flag which when set to '1' indicates that the frame rate uses the NTSC frequency offset of 1000/1001. This bit would typically be set when `frame_rate` = 24, 30 or 60, in which case the resulting frame rate would be 23.97, 29.94 or 59.97 respectively. When set to '0' no frequency offset is present, i.e. if (`frequency_offset` ==1), frame rate = (1000/1001) * (`frame_rate` + `seconds`/16).

8.8.5 SkipFrame

8.8.5.1 Syntax

```
class SkipFrame {
   int nFrame = 0;
   do {
      bit(4) number_of_frames_to_skip;
      nFrame = number_of_frames_to_skip + nFrame;
   } while (number_of_frames_to_skip == 0b1111);
}
```

8.8.5.2 Semantics

`number_of_frames_to_skip` is a 4-bit unsigned integer indicating the number of frames skipped. If the `number_of_frames_to_skip` is equal to 15 (pattern "1111") then another 4-bit word follows allowing a skip of up to 29 frames (pattern "11111110") to be specified. If the 8-bits pattern equals "11111111", then another 4-bits word shall follow and so on, and the number of frames skipped is incremented by 30. Each 4-bit pattern of '1111' increments the total number of frames to skip with 15.

8.8.6 AnimationFrameData

8.8.6.1 Syntax

```
class AnimationFrameData (AnimationMask mask) {

   int i;
   for (i=0; i<mask.numNodes; i++) {
      if (mask.isActive[i]) {
            NodeData node = mask.animNode[i]
         switch (node.nodeType) {
         case FaceType:

         case BodyType:

            fba_object_plane_data();
            break;
         case IndexedFaceSet2DType:
            MeshObjectPlaneData();
            break;
         default
            int j;
            for(j=0; j<node.numDYNfields; j++) {
               if (node.isAnimField[j])
                  AnimationField AField(node.field[node.dyn2all[j]],mask.isIntra);
            }
         }
      }
   }
}
```

8.8.6.2 Semantics

The `AnimationFrameData` corresponds to the field data for the nodes being animated. In the case of an **IndexedFaceSet2D**, a **Face**, or a **Body** node pointed to by the AnimationMask, the syntax used is that defined ISO/IEC 14496-2 for animation frames and not the generic BIFS-Anim syntax as defined in 8.8.7.

`fba_object_plane_data()` is defined in 6.2.10.1, ISO/IEC 14496-2 .

`MeshObjectPlaneData()` is defined in 6.2.9.1, ISO/IEC 14496-2 .

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In other cases, for each field declared as an animated field is the AnimationMask, the AnimationField is sent.

BIFS Anim streams shall not contain a combination of generic BIFS-Anim, FBA stream or 2Dmesh.

In predictive mode, at the beginning of the AnimationFrameData, an adaptive arithmetic coder session is initiated by resetting the adaptive arithmetic coder in the way defined by the procedure decoder_reset( ) in 8.11. Then, the animated values are sent using this adaptive arithmetic coder, using and updating their own models.

8.8.7 AnimationField

8.8.7.1 Syntax

```
class AnimationField(FieldData field, boolean isIntra) {
   AnimFieldQP aqp = field.aqp;
   if (isIntra) {
        bit(1) hasQP;
        if(hasQP) {
           AnimQP QP(aqp);
        }
        int i;
        for (i=0; i<aqp.numElements; i++) {
             AnimIValue ivalue(field);
        }
   } else {
        int i;
        for (i=0; i<aqp.numElements; i++) {
             AnimPValue pvalue(field);
        }
   }
}
```

8.8.7.2 Semantics

In an AnimationField, if in intra mode, a new animation quantization parameter value may be sent. The intra frame follows.

In intra mode, if BIFSConfiguration.randomAccess is TRUE , the field's predictive models shall then be reset to be uniform models as defined by the procedure model_reset(PNbBits) in 8.11. If BIFSConfiguration.randomAccess is FALSE, the field's models are reset only if a new AnimQP is received.

- If randomAccess is set to TRUE, then the InitialAnimQP shall be used until the next intra frame.
- If randomAccess is set to FALSE, then the AnimQP that was valid at the previous intra frame shall be used. In this case, no random access is possible ato this particular frame.

The value is then sent: in intra mode, an AnimIValue is expected, in predictive mode an AnimPValue is expected.

8.8.8 AnimQP

8.8.8.1 Syntax

```
class AnimQP(AnimFieldQP aqp) {

   bit (1) IMinMax ;
   if (IMinMax) {

    aqp.useDefault=FALSE;
   switch(aqp.animType) {
        case  4:   // Color
        case  8:   // BoundFloats
          bit(1)  aqp.useDefault
        case  1:   // Position 3D
        case  2:   // Position 2D
        case 15:   // Position 4D
        case 11:   // Size 3D
        case 12:   // Size 2D
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        case  7:   // Floats
           if (!aqp.useDefault) {
              for (i=0;i<getNbBounds(aqp.animType);i++) {
                 bit(1)       useEfficientCoding
                 GenericFloat aqp.Imin[i](useEfficientCoding);
              }
              for (i=0;i<getNbBounds(aqp.animType);i++)
                 bit(1)       useEfficientCoding
                 GenericFloat aqp.Imax[i](useEfficientCoding);
           }
           break;

        case 13:   // Integers
           int(32)    aqp.IminInt[0];
        break;

   }

   bit (1) hasINbBits;
   if (hasINbBits)
      unsigned int(5)         aqp.INbBits;

   bit (1) PMinMax ;
   if (PMinMax) {
      for (i=0;i<getNbBounds(aqp.animType);i++) {
           int(INbBits+1) vq
           aqp.Pmin[i] = vq-2^aqp.INbBits;
      }
    }

   bit (1)hasPNbBits;
   if (hasPNbBits) {
      unsigned int(4) aqp.PNbBits;
   }

}
```

8.8.8.2 Semantics

The AnimQP specifies the quantization parameters that shall be used until the next intra frame is received. AnimQP is identical to InitialAnimQP (subclause 8.5.7) with the exception that each quantization parameter may or may not be sent.

If BIFSConfiguration.randomAccess is TRUE and if the parameter is not coded, then the parameter defined in the InitialAnimQP in the AnimationMask is used by default.

If BIFSConfiguration.randomAccess is FALSE and if the parameter is not coded, then the parameter defined in the latest AnimQP (or InitialAnimQP if this parameter was never modified) is used.

8.8.9 AnimIValue

8.8.9.1 Syntax

```
class AnimIValue(FieldData field) {
   switch (field.animType) {
      case  9:   // Normal
         int(1)       direction
      case 10:   // Rotation
         int(2)       orientation
         break;
      default:
         break;
    }
   for (j=0;j<getNbComp(field);j++) {
        int(field.nbBits)  vq[j];
    }
}
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.8.9.2 Semantics

The `AnimIValue` represents the quantized intra value of a field. The value is coded according to the quantization process described in 8.3.3.

For normals the direction and orientation values specified in the quantization process are first coded. For rotations only the orientation value is coded. If the bit representing the direction is 0, the normal's direction is set to 1, if the bit is 1, the normal's direction is set to –1. The value of the orientation is coded as an unsigned integer using 2 bits.

The compressed components `vq[i]` of the field's value are then coded as a sequence of unsigned integers using the number of bits specified in the field data structure.

The decoding process in intra mode computes the animation values by applying the inverse quantization process.

8.8.10 AnimPValue

8.8.10.1 Syntax

```
class AnimPValue(FieldData field) {
   switch (field.animType) {
      case  9:   // Normal
         int(1)     inverse
         break;
      default:
         break;
    }
   for (j=0;j<getNbComp(field);j++) {
      int(aacNbBits)  vqDelta[j];
   }
}
```

8.8.10.2 Semantics

The `AnimPValue` represents the difference between the previously received quantized value and the current quantized value of a field. The value is coded using the compensation process `AddDelta` described in 8.4.

The values are decoded from the adaptive arithmetic coder bitstream with the procedure v_{aac} =`aa_decode(model)` defined in 8.11. The model is updated with the procedure `model_update(model,` v_{aac} `)`.

For normals the inverse value is decoded through the adaptive arithmetic coder with a uniform, non-updated model. If the bit is 0, then `inverse` is set to 1, the bit it is 1, `inverse` is set to –1.

The compensation values `vqDelta[i]` are then decoded in sequence. Let $v_q(t-1)$ be the quantized value decoded at the previous frame and $v_{aac}(t)$ be the value decoded by the frame's adaptive arithmetic decoder at instant *t* with the field's models. The value a time *t* is obtained from the previous value as follows:

$$v_\delta(t) = v_{aac}(t) + PMin$$
$$v_q(t) = \mathrm{AddDelta}(v_q(t-1), v_\delta(t))$$
$$v(t) = \mathrm{InvQuant}\left(v_q(t)\right)$$

The field's models are updated each time a value is decoded through the adaptive arithmetic coder.

If the `animType` is 1 (`Position3D`) or 2 (`Position2D`), each component of the field's value is using its own model and offset `PMin[i]`. In all other cases the same model and offset `PMin[0]` is used for all the components.

`aacNbBits` is the variable number of bits needed for the adaptive arithemtic coder to decode the symbol (see 8.11).

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.9 Interpolator compression

8.9.1 Overview

The interpolator compression is a tool to efficiently compress the interpolator nodes in BIFS. The decoder structure of the interpolator compression is shown in Figure 46. The interpolator decoder consists of Header Decoder, Key Decoder, KeyValue Decoder and Interpolator Synthesizer. The header information for key and keyValue are decoded in the header decoder and are used in Key Decoder and KeyValue Decoder. The Key Decoder and KeyValue Decoder receive the arithmetic coded bitstream and restore the key and keyValue data of the interpolator. The interpolator synthesizer restores the keys and keyValues that has been excluded in the bitstream.

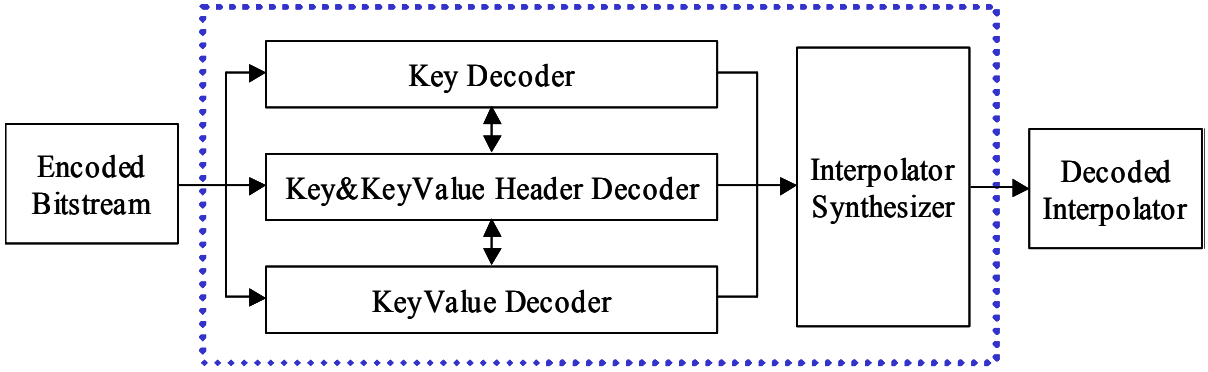


Figure 46 — General Structure of Interpolator Decoder

8.9.2 Key decoding

8.9.2.1 KeyHeader

8.9.2.1.1 Syntax

```
class KeyHeader  {
   int i;
   unsigned int(5) nKeyQBit;
   unsigned int(5) nNumKeyCodingBit;
   unsigned int(nNumKeyCodingBit) nNumberOfKey;
   unsigned int(4) nKeyDigit;
   bit(1) bIsLinearKeySubRegion;
   if(bIsLinearKeySubRegion == 1)
      LinearKey lKey(nKeyDigit);
   bit(1) bRangeFlag;
   if(bRangeFlag == 1)
      KeyMinMax keyMinMax(nKeyDigit);
   unsigned int(5) nBitSize;
   unsigned int(2) nKDPCMOrder;
   for(i = 0; i < nKDPCMOrder + 1; i++) {
      bit(1) nQIntraKeySign[[i]];
      if(i == 0 && nQIntraKeySign[i] == 1)
         continue;
      unsigned int(nBitSize) nQIntraKey[[i]];
   }
   bit(1) bShiftFlag;
   if(bShiftFlag == 1) {
      bit(1) nKeyShiftSign;
      unsigned int(nBitSize) nKeyShift;
   }
   unsigned int(3) nDNDOrder;
   if(nDNDOrder == 7) {
      bit(1) bNoDND;
      if(bNoDND == 1)
         nDNDOrder = -1;
   }
```

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   int nMaxQBit = nBitSize;
   for(i = 0; i < nDNDOrder; i++) {
      bit(1) nKeyMaxSign[[i]];
      unsigned int(nMaxQBit) nKeyMax[[i]];
      nMaxQBit = (int)(log10(abs(nKeyMax[i]))/log10(2))+1;
      if(nMaxQBit+1 < nBitSize)
         nMaxQBit += 1;
      else
         nMaxQBit = nBitSize;
   }
   int bSignedAACFlag;
   int nKeyCodingBitQBit = (int)(log10(nKeyQBit)/log10(2))+1;
   unsigned int(nKeyCodingBitQBit) nKeyCodingBit;
   if(nDNDOrder != -1 && nDNDOrder != 0) {
      bit(1) bKeyInvertDownFlag;
      if(bKeyInvertDownFlag == 1) {
         unsigned int(nKeyCodingBit) nKeyInvertDown;
         bSignedAACFlag = 0;
      } else {
         bSignedAACFlag = 1;
      }
   } else {
      bSignedAACFlag = 0;
   }
}
```

8.9.2.1.2 Semantics

The key header data are the information needed to decode the key data. The main information in the key header are the number of keys, the quantization bit, the intra key data, the DND header and the actual bits used for decoding.

nKeyQBit is the quantization bit that is used in the inverse quantization to restore the float values. nNumKeyCodingBit indicates the bit size of nNumberOfKey, which indicates the number of key data. nKeyDigit indicates the maximum significant digit in the original key data and can be used for rounding off the decoded values.

When the information on linear key sub-regions is included in the header, bIsLinearKeySubRegion flag is set to 1. In this case, the keys included in certain sub-region within the whole range of keys can be calculated using the decoded header information following the bIsLinearKeySubRegion flag.

bRangeFlag indicates whether the range of key data is from 0 to 1 or not. If the range is not 0 to 1, the minimum value and the maximum value are decoded from the KeyMinMax class. KeyMinMax class reconstructs the minimum value and the maximum value for inverse quantization. Each value can be separated into the mantissa and the exponent.

nBitSize is the bit size of nQIntraKey, nKeyShift and is the initial bit size of nKeyMax.

nQIntraKey is the magnitude of the first quantized intra data. It is combined together with the nQIntraKeySign, which indicates the sign of nQIntraKey. It is used as a base for restoring the rest of the quanitzed key data. For all the sign bits in the interpolator compression, the value 0 denotes a positive sign and 1 denotes a negative sign.

nKDPCMOrder is the order of DPCM minus 1. The range of the order may be from 1 to 3. The number of the quantized intra data is the same as the order of DPCM.

nKeyShift, together with the sign bit nKeyShiftSign, is the integer number that indicates the amount of shift in the key data decoder. These two values are decoded if the bShiftFlag is set to true.

nDNDOrder is the order of DND (Divide-and-divide). The DND is described in the key data decoder (see subclause 8.9.2.6.3). If the value of nDNDOrder is 7, then bNoDND is decoded. This boolean value indicates whether inverse DND will be processed or not. nKeyMax is the maximum value or the minimum value used during each successive inverse DND process. nKeyCodingBit is the bits used for coding key data. bSignedAACFlag indicates which decoding method is used for AAC decoding. If the value is 0, the unsigned AAC decoding is performed. Otherwise, the signed AAC decoding is performed.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

bKeyInvertDownFlag is the boolean value indicating whether nKeyInvertDown is used or not. nKeyInvertDown is the integer value that makes all quantized key data above it to be inverted to negative values starting from –1 and below. If nKeyInvertDown is -1, then inverting is not performed.

8.9.2.2 LinearKey

8.9.2.2.1 Syntax

```
class LinearKey (int nKeyDigit) {
   unsigned int(5) nNumLinearKeyCodingBit;
   unsigned int(nNumLinearKeyCodingBit) nNumberOfLinearKey;
   KeyMinMax kMinMax(nKeyDigit);
}
```

8.9.2.2.2 Semantics

nNumLinearKeyCodingBit : This value indicates the number of bits needed to code the number of keys that are linearly predictable.

nNumberOfLinearKey : This value indicates the number of keys that are linearly predictable.

8.9.2.3 KeyMinMax

8.9.2.3.1 Syntax

```
class KeyMinMax (int nKeyDigit) {

   bit(1) bMinKeyDigitSame;
   if(bMinKeyDigitSame == 0)
      unsigned int(4) nMinKeyDigit;
   else
      nMinKeyDigit = nKeyDigit;
   if(nMinKeyDigit != 0) {
      if(nMinKeyDigit < 8) {
         int count = (int)(log10(10^nMinKeyDigit-1)/log10(2)) + 1;
         bit(1) nMinKeyMantissaSign;
         unsigned int(count) nMinKeyMantissa;
         bit(1) nMinKeyExponentSign;
         unsigned int(6) nMinKeyExponent;
      } else
         float(32) fKeyMin;
   }
   bit(1) bMaxKeyDigitSame;
   if(bMaxKeyDigitSame == 0)
      unsigned int(4) nMaxKeyDigit;
   else
      nMaxKeyDigit = nKeyDigit;
   if(nMaxKeyDigit != 0) {
      if(nMaxKeyDigit < 8) {
         int count = (int)(log10(10^nMaxKeyDigit-1)/log10(2)) + 1;
         bit(1) nMaxKeyMantissaSign;
         unsigned int(count) nMaxKeyMantissa;
         bit(1) bSameExponent;
         if(bSameExponent == 0) {
            bit(1) nMaxKeyExponentSign;
            unsigned int(6) nMaxKeyExponent;
         }
         else
            nMaxKeyExponent = nMinKeyExponent;
      } else
         float(32) fKeyMax;
   }
}
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.9.2.3.2 Semantics

bMinKeyDigitSame : This flag indicates if the maximum significant digit(nKeyDigit) of the entire keys and the significant digit of min key are the same

nMinKeyDigit : This value indicates the significant digit of the min key

nMinKeyMantissaSign : This value indicates a sign of nMinKeyMantissa.

nMinKeyMantissa : This value indicates the mantissa of the min key

nMinKeyExponentSign : This value indicates a sign of nMinKeyExponent.

nMinKeyExponent : This value indicates the exponent of the min key

fKeyMin : This value indicates the value of the min key

bMaxKeyDigitSame : This flag indicates if the maximum significant digit(nKeyDigit) of the entire keys and the significant digit of max key are the same

nMaxkeyDigit : This value indicates the significant digit of the max key

nMaxKeyMantissaSign : This value indicates a sign of nMaxKeyMantissa.

nMaxKeyMantissa : This value indicates the mantissa of the max key

bSameExponent : This flag indicates if the exponent of the max key is the same as nMinKeyExponent.

nMaxKeyExponentSign : This value indicates a sign of nMaxKeyExponent.

nMaxKeyExponent: This value indicates the exponent of the max key

fKeyMax: This value indicates the max key

8.9.2.4 Key

8.9.2.4.1 Syntax

```
class Key (KeyHeader kHeader) {
   int nQKey[kHeader.nNumberOfKey];
   int i;
   int nNumberOfRemainingKey;
   if(kHeader.bIsLinearKeySubRegion == 1)
      nNumberOfRemainingKey = kHeader.nNumberOfKey - kHeader.lKey.nNumberOfLinearKey;
   else
      nNumberOfRemainingKey = kHeader.nNumberOfKey;
   for(i = kHeader.nKDPCMOrder+1; i < nNumberOfRemainingKey; i++) {
      if(kHeader.bSignedAACFlag == 0)
         decodeUnsignedAAC(nQKey[i], kHeader.nKeyCodingBit, keyContext);
      else
         decodeSignedAAC(nQKey[i], kHeader.nKeyCodingBit+1, keySignContext, keyContext);
   }
}
```

8.9.2.4.2 Semantics

nQKey: This array stores the quantized key data that are decoded from bitstream. The keyContext is the context for reading the magnitudes of nQKey. The keySignContext is the context for reading the signs of nQKey.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

decodeUnsignedAAC: This function performs the unsigned decoding of adaptive arithmetic coding with a given context.

decodeSignedAAC: This function performs the signed decoding of adaptive arithmetic coding with a given context.

8.9.2.5 KeySelectionFlag

8.9.2.5.1 Syntax

```
class KeySelectionFlag(KeyHeader kHeader, int bPreserveKey) {
   int i;
   int nNumOfKeyValue = 0;
   if(bPreserveKey == 1) {
      for(i=0; i<kHeader.nNumberOfKey; i++) {
         qf_decode(&keyFlag[i], keyFlagContext);
         if(keyFlag[i] == 1)
            nNumOfKeyValue++;
      }
   } else
      nNumOfKeyValue = kHeader.nNumberOfKey;
}
```

8.9.2.5.2 Semantics

keyFlag: This boolean array indicates whether the keyValue of the i-th key is coded or not.

nNumOfKeyValue: This integer value indicates the number of keyValues to be decoded.

8.9.2.6 Decoding Process

8.9.2.6.1 Overview

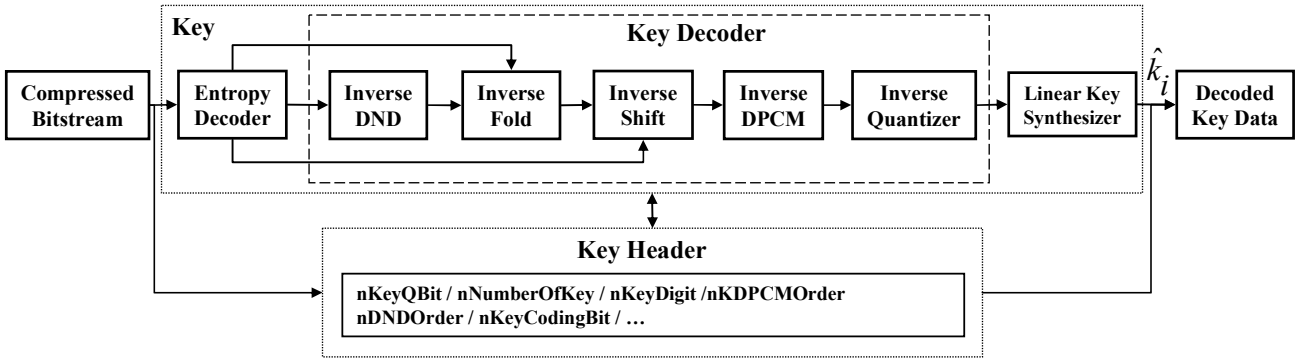


Figure 47 — Block diagram of key decoder for interpolator compression

Figure 47 shows the block diagram of the key field decoder for interpolator compression. Key data are float values between the given range(-∞ ~ +∞). In the key header information, nNumberOfKey indicates the number of key data. The decoder receives the encoded bitstream that was generated by the encoder. There are two parts to decode the key data. Those are key header decoder and key data decoder.

8.9.2.6.2 Key Header Decoding

When the information on linear key sub-regions is included, the equation for decoding keys in the sub-region is as follows.

$$Key_i = fKeyMin + \frac{(fKeyMax - fKeyMin) * i}{(nNumberOfLinearKey - 1)} \qquad (i = 0...nNumberOfLinearKey - 1)$$

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

fKeyMin and fKeyMax are decoded from the encoded bitstream in two different ways. The first one is that these values are directly decoded in the type of 32 bit float if the key digits (nMinKeyDigit, nMaxKeyDigit in KeyMinMax class) are equal to or more than 8. The other one is that these values are decoded into their mantissa and exponent float if the key digits are less than 8. In this case, these values can be calculated using following equation.

$$fKeyMin = \frac{MinKeyMantissaSign * nMinKeyMantissa}{10^{MinKeyExponentSign * nMinKeyExponent}}$$

$$fKeyMax = \frac{MaxKeyMantissaSign * nMaxKeyMantissa}{10^{MaxKeyExponentSign * nMaxKeyExponent}}$$

In this equation, MinKeyMantissaSign has the value of 1 when nMinKeyMantissaSign (in KeyMinMax class) is set to 0 and MinKeyMantissaSign has the value of -1 when nMinKeyMantissaSign is 1. The same rule is also applied to MinKeyExponentSign, MaxKeyMantissaSign, and MaxKeyExponentSign for nMinKeyExponentSign, nMaxKeyMantissaSign and nMaxKeyExponentSign.

8.9.2.6.3 Key Data Decoding

The entropy decoder decodes the quantized key data and passes the data to the key data decoder. It reconstructs the differentiated key data and performs inverse DPCM, inverse Fold and inverse Quantization to generate the final key data.

The decoder for compressed key data uses two types of entropy decoding method, depending on the value of bSignedAACFlag. If the quantized data are signed data (i.e. bSignedAACFlag==1), then decodeSignedAAC() function is used. Otherwise, if the quantized data are unsigned data (i.e. bSignedAACFlag==0), then decodeUnsignedAAC() function is used (see subclause 8.9.6).

The following function describes the decoding process of key data.

```
void decodeKey(int* nQIntraKey, int* nQKey, int nNumberOfRemainingKey, int nKeyQBit)
{
   add(nQKey, nQIntraKey);    // add nQIntraKey array to nQKey array
// Inverse DND
   if(nDNDOrder > 0) {
      if(nKeyInvertDown != -1) {
         for(k = 0; k < nNumberOfRemainingKey; k++)
            invert-down(nQKey[k], nKeyInvertDown);
      }
      for(i = nDNDOrder; i > 0; i--) {
         if(nKeyMax[i-1] >= 0) {
            if(i == 1) {
               for(k = 0; k < nNumberOfRemainingKey; k++)
               inverse-divide(nQKey[k], nKeyMax[i-1]);
            }
            else {
               for(k = 0; k < nNumberOfRemainingKey; k++)
                  inverse-divide-down(nQKey[k], nKeyMax[i-1]);
            }
         } else {
               for(k = 0; k < nNumberOfRemainingKey; k++)
                  inverse-divide-up(nQKey[k], nKeyMax[i-1]);
         }
      }
   }
   //Inverse Fold
   if(nDNDOrder != -1) {
      for(k = 0; k < nNumberOfRemainingKey; k++)
         inverse-fold(nQKey[k])
   }
   for(k = 0; k < nNumberOfRemainingKey; k++)
      inverse-shift(nQKey[k], nKeyShift)//Inverse Shift

   for(i = 0; i < nKDPCMOrder; i++)  //Inverse DPCM
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      inverse-dpcm(nQKey, nNumberOfRemainingKey);

   //Inverse Quantization
   inverse-quantize(nQKey, nNumberOfRemaingKey, nKeyQBit);
}
```

When the key data decoder receives the quantized key data from the entropy decoder, inverse DND is performed, followed by inverse Fold, inverse Shift, inverse DPCM and inverse quantization.

If the order of DND is 0, then inverse DND is bypassed and inverse Fold is performed next.

If the order of DND is -1, then inverse DND and inverse Fold are bypassed and inverse Shift is performed next, using the following equation.

$$\text{inverse-shift}(v) = v + \text{nKeyShift}$$

If the order of DND is greater than 0, then nKeyInvertDown is considered first. If nKeyInvertDown is not -1, then all quantized key data above nKeyInvertDown are inverted to negative values starting from –1 and below, using the following equation.

$$\text{invert-down}(v) = \begin{cases} v & (\text{if } v \le \text{nKeyInvertDown}) \\ \text{nKeyInvertDown} - v & (\text{if } v > \text{nKeyInvertDown}) \end{cases}$$

After considering nKeyInvertDown, the processes of inverse-divide-down or inverse-divide-up are performed, depending on the value of nKeyMax for each DND order. If the value is greater than or equal to 0, then inverse-divide-down is performed using the following equation.

$$\text{inverse-divide-down}(v) = \begin{cases} v & (\text{if } v \ge 0) \\ (\text{nKeyMax}_i+1)+ (v-1) / 2 & (\text{if } v < 0, v \bmod 2 \ne 0) \\ v / 2 & (\text{if } v < 0, v \bmod 2 = 0) \end{cases}$$

However, if nKeyMax is less than 0, then inverse-divide-up is performed using the following equation.

$$\text{inverse-divide-up}(v) = \begin{cases} v & (\text{if } v \ge 0) \\ (\text{nKeyMax}_i -1) - (v-1) / 2 & (\text{if } v < 0, v \bmod 2 \ne 0) \\ v / 2 & (\text{if } v < 0, v \bmod 2 = 0) \end{cases}$$

The above process (inverse-divide-down or inverse-divide-up) is performed as many as the order of DND. At the last stage of inverse DND, if the value of the last nKeyMax is greater than or equal to 0, then inverse-divide is performed using the following equation.

$$\text{inverse-divide}(v) = \begin{cases} v & (\text{if } v \ge 0) \\ v+(\text{nKeyMax}_0 +1) & (\text{if } v < 0) \end{cases}$$

After the inverse DND, if the order of DND is not –1, inverse Fold is performed using the following equation.

$$\text{inverse-fold}(v) = \begin{cases} (v+1) / (-2) & (\text{if } v \bmod 2 \ne 0) \\ v / 2 & (\text{if } v \bmod 2 = 0) \\ 0 & (\text{if } v = 0) \end{cases}$$

After inverse Fold, inverse Shift is performed as described above.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

After the above process, inverse DPCM is performed using the following equation.

v(i+1) = v(i) + delta(i) (i: index of data, v: integer array, delta: difference value)

After inverse DPCM, the inverse quantization is performed. The inverse quantization is performed using the following equation so that the quantized key data (v) are inverse quantized.

$$inverse-quantize(v) = fKeyMin + \frac{v}{(2^{nKeyQBit}-1)} \times (fKeyMax - fKeyMin)$$

fKeyMin and fKeyMax are the minimum and maximum floating point numbers in the original key data. nKeyQBit is the quantization bit size for inverse quantizationand v is the quantized value to be inverse quantized.

When the information on linear key sub-ranges is included in the header, the Linear Key Synthesizer in Figure 47. will collect all the keys from the sub-ranges and the data from the Key Data Decoder to restore the final key data. Finally the keys are sorted in the order of time.

8.9.3 Coordinate Interpolator Decoding

8.9.3.1 CompressedCoordinateInterpolator

8.9.3.1.1 Syntax

```
class CompressedCoordinateInterpolator {
   KeyHeader kHeader;
   CoordIKeyValueHeader coordIKVHeader;
   qf_start();
   aligned(8) Key k(kHeader);
   CoordIKeyValue coordIKeyValue(coordIKVHeader, kHeader.nNumberOfKey);
}
```

8.9.3.1.2 Semantics

This is a top class for reading the compressed bitstream of coordinate interpolator. KeyHeader and Key are the classes for reading key information from the bitstream, which corresponds to key field data in conventional CoordinateInterpolator node. CoordIKeyValueHeader and CoordIKeyValue are the classes for reading keyValue information corresponding to keyValue field data in conventional CoordinateInterpolator node.

The function qf_start() is used for initializing arithmetic decoder before reading AAC encoded part of the bitstream (see subclause 7.13.10.1 of ISO/IEC 14496-2:2004).

8.9.3.2 CoordIKeyValueHeader

8.9.3.2.1 Syntax

```
class CoordIKeyValueHeader {
   bit(1) bTranspose;
   unsigned int(5) nKVQBit;
   unsigned int(5) nCoordQBit;
   unsigned int(nCoordQBit) nNumberOfCoord;
   unsigned int(4) nKVDigit;
   KeyValueMinMax kVMinMax (nKVDigit);
   unsigned int(nKVQBit) nXQMinOfMin;
   unsigned int(nKVQBit) nXQMinOfMax;
   unsigned int(nKVQBit) nYQMinOfMin;
   unsigned int(nKVQBit) nYQMinOfMax;
   unsigned int(nKVQBit) nZQMinOfMin;
   unsigned int(nKVQBit) nZQMinOfMax;
   unsigned int(nKVQBit) nXQMaxOfMin;
   unsigned int(nKVQBit) nXQMaxOfMax;
   unsigned int(nKVQBit) nYQMaxOfMin;
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   unsigned int(nKVQBit) nYQMaxOfMax;
   unsigned int(nKVQBit) nZQMaxOfMin;
   unsigned int(nKVQBit) nZQMaxOfMax;
}
```

8.9.3.2.2 Semantics

The data of keyValue header are decoded after the key header data. The main information in the keyValue header are the number of vertices, the quantization parameter for keyValue and minimum and maximum values for quantization.

bTranspose is the flag for transpose mode or vertex mode. If the value is 1, the transpose mode is selected. Otherwise, the vertex mode is selected. nKVQBit is the quantization bit that is used in the inverse quantization to restore the float values. nCoordQBit indicates the bit size used for nNumberOfCoord, which indicates the number of vertices. nKVDigit is used after the inverse quantization and this indicates the maximum significant digit for the keyValue data. KeyValueMinMax class reconstructs the minimum values and the maximum range for inverse quantization. Each value can be separated into the mantissa and the exponent. KeyValueMinMax class is described in the subclause 8.9.5.3. The remaining header information are the maximum and the minimum values among the maximum and the minimum quantized values of each component of keyValue. For example, nXQMinOfMax indicate the minimum value among the maximum quantized x component values in each vertex. These are necessary for decoding the keyValue data.

8.9.3.3 CoordIKeyValue

8.9.3.3.1 Syntax

```
class CoordIKeyValue (CoordIKeyValueHeader coordIKVHeader, int nNumberOfKey) {
   int j, c;
    if(coordIKVHeader.bTranspose == 1) {
        int temp = nNumberOfKey;
        nNumberOfKey = coordIKVHeader.nNumberOfCoord;
       coordIKVHeader.nNumberOfCoord = temp;
    }

   int nKVACodingBitQBit = (int)(log10(abs(coordIKVHeader.nKVQBit))/log10(2))+1;
   int nDPCMMode[coordIKVHeader.nNumberOfCoord][3];
   unsigned int bSelFlag[coordIKVHeader.nNumberOfCoord][3] = 1;
   CoordIDPCMMode coordIDPCMMode(coordIKVHeader);
   for(j = 0; j < coordIKVHeader.nNumberOfCoord; j++) {
      for(c = 0; c < 3; c++) {
         if(c == 0) {
            if(coordIKVHeader.nXQMaxOfmin <= coordIKVHeader.nXQMinOfmax) {
               qf_decode(&bSelFlag[j][c], selectionFlagContext);
            }
         }
         else if(c == 1) {
            if(coordIKVHeader.nYQMaxOfmin <=  coordIKVHeader.nYQMinOfmax) {
               qf_decode(&bSelFlag[j][c], selectionFlagContext);
            }
         }
         else if(c == 2) {
            if(coordIKVHeader.nZQMaxOfmin <=  coordIKVHeader.nZQMinOfmax) {
               qf_decode(&bSelFlag[j][c], selectionFlagContext);
            }
         }
         if(bSelFlag[j][c] == 1) {
            if(c == 0)
               decodeUnsignedAAC(&nKVACodingBit[j][c], nKVACodingBitQBit, aqpXContext);
            else if(c == 1)
               decodeUnsignedAAC(&nKVACodingBit[j][c], nKVACodingBitQBit, aqpYContext);
            else if(c == 2)
               decodeUnsignedAAC(&nKVACodingBit[j][c], nKVACodingBitQBit, aqpZContext);
            if(j > 0) {
               if(nDPCMMode[j][c] == 2 || nDPCMMode[j][c] == 3) {
                  int nQBitOfRef = (int)(log10(abs(j-1))/log10(2))+1;
                  decodeUnsignedAAC(&nRefVertex[j][c], nQBitOfRef, refContext);
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            }
         }
         if(nKVACodingBit[j][c] != 0) {
            decodeSignedAAC(&nQMin[j][c], coordIKVHeader.nKVQBit+1,
                 qMinSignContext, qMinContext);
            decodeSignedAAC(&nQMax[j][c], coordIKVHeader.nKVQBit+1,
                 qMaxSignContext, qMaxContext);
         }
      } else
         decodeSignedAAC(&nQMin[j][c], coordIKVHeader.nKVQBit+1,
            qMinSignContext, qMinContext);

      CoordIKeyValueDic coordIKeyValueDic(bSelFlag[j][c],
         nKVACodingBit[j][c], nNumberOfKey, c);
   }
  }
}
```

8.9.3.3.2 Semantics

nDPCMMode: This integer array indicates the DPCM mode for each component (x, y, z) of each vertex. The value may be 1 (temporal), 2 (spatial) or 3 (spatiotemporal).

bSelFlag: This boolean array indicates the selection flag for each component of each vertex. Only the component of a vertex with this flag set to true are coded using dictionary coder. The selectionFlagContext is the context for reading the values of bSelFlag.

nKVACodingBit: This integer array indicates the actual number of bits necessary for coding each component of each vertex. The aqpXContext is the context for reading the values of nKVACodingBit.

nRefVertex: This integer array indicates the index of reference vertex for each vertex. The refContext is the context for reading the values of nRefVertex.

nQMin: This integer array indicates the minimum quantized value of each component of each vertex. The qMinContext is the context for reading the values of nQMin. The qMinSignContext is the context for reading the signs of nQMin.

nQMax: This integer array indicates the maximum quantized value of each component of each vertex. The qMaxContext is the context for reading the values of nQMax. The qMaxSignContext is the context for reading the signs of nQMax.

8.9.3.4 CoordIDPCMMode

8.9.3.4.1 Syntax

```
class CoordIDPCMMode (CoordIKeyValueHeader coordIKVHeader) {
   int i, s, k;
   unsigned int bIndexDPCMMode[coordIKVHeader.nNumberOfCoord] = 0;
   int nNumberOfSymbol = 0;
   for(i = 0; i < 27; i++) {
      qf_decode(&bAddressOfDPCMMode[i], dpcmModeDicAddressContext);
      if(bAddressOfDPCMMode[i] == 1)
         nNumberOfSymbol++;
   }
   for(s = 0; s < nNumberOfSymbol; s++) {
      for(k = 1; k < coordIKVHeader.nNumberOfCoord; k++) {
         if(bIndexDPCMMode[k] == 0) {
            qf_decode(&bDPCMIndex, dpcmModeDicIndexContext);
            if(bDPCMIndex == 1)
               bIndexDPCMMode[k] = 1;
         }
      }
   }
}
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.9.3.4.2 Semantics

bAddressOfDPCMMode: This boolean array indicates the usage of each DPCM dictionary symbol, which consists of DPCM mode for each component, in the DPCM dictionary table. There are three types of DPCM modes (T, S, and T+S) and three components in a vertex. Therefore, the combination of them results in 27 dictionary symbols, as shown in Table 87 — The DPCM dictionary table. The dpcmModeDicAddressContext is the context for reading the values of bAddressOfDPCMMode.

Table 87 — The DPCM dictionary table

DPCM Mode	Dictionary symbol	DPCM Mode	Dictionary symbol
0	(T, T, T)	14	(S, S, T+S)
1	(T, T, S)	15	(S, T+S, T)
2	(T, T, T+S)	16	(S, T+S, S)
3	(T, S, T)	17	(S, T+S, T+S)
4	(T, S, S)	18	(T+S, T, T)
5	(T, S, T+S)	19	(T+S, T, S)
6	(T, T+S, T)	20	(T+S, T, T+S)
7	(T, T+S, S)	21	(T+S, S, T)
8	(T, T+S, T+S)	22	(T+S, S, S)
9	(S, T, T)	23	(T+S, S, T+S)
10	(S, T, S)	24	(T+S, T+S, T)
11	(S, T, T+S)	25	(T+S, T+S, S)
12	(S, S, T)	26	(T+S, T+S, T+S)
13	(S, S, S)		

bDPCMIndex: This boolean value indicates which DPCM dictionary symbol is used for each vertex. The dpcmModeDicIndexContext is the context for reading the values of bDPCMIndex.

8.9.3.5 CoordIKeyValueDic

8.9.3.5.1 Syntax

```
class CoordIKeyValueDic (unsigned int bSelFlag, unsigned int nKVCodingBit, int
nNumberOfKey, int c) {
   if(bSelFlag == 1 && nKVCodingBit != 0) {
      qf_decode(&nDicModeSelect, dicModeSelectionContext);
      if(nDicModeSelect == 1)
         CoordIIncrementalMode coordIIncrementalMode(nKVCodingBit, nNumberOfKey);
      else
         CoordIOccurrenceMode coordIOccurrenceMode(nKVCodingBit,   nNumberOfKey, c);
   }
}
```

8.9.3.5.2 Semantics

nDicModeSelect: This boolean value indicates which mode is used for dictionary coding. The value of 1 indicates the incremental mode and 0 indicates the occurrence mode.

8.9.3.6 CoordIIncrementalMode

8.9.3.6.1 Syntax

```
class CoordIIncrementalMode (unsigned int nKVCodingBit, int nNumberOfKey) {
   int i, s, k;
   int nSizeOfAddress = (2^(nKVCodingBit+1))-1;
   unsigned int bAddrIndex[nNumberOfKey] = 0;
   int nNumberOfSymbol = 0;
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   for(i = 0; i < nSizeOfAddress; i++) {
      qf_decode(&bAddress[i], dicAddressContext);
      if(bAddress[i] == 1) {
         nNumberOfSymbol++;
      }
   }
   for(s = 0; s < nNumberOfSymbol; s++) {
      qf_decode(&nTrueOne, dicOneContext);
      for(k = 0; k < nNumberOfKey; k++) {
         if(bIndexOfAddr[k] == 0) {
            qf_decode(&bAddrIndex, dicIndexContext);
            if(bAddrIndex == nTrueOne) {
               bAddrIndex[k] = 1;
            }
         }
      }
   }
}
```

8.9.3.6.2 Semantics

bAddress: This boolean array indicates the usage of each incremental dictionary symbol, which represents the quantized keyValue. The number of symbols in the incremental dictionary table is the 2 ^ (nKVCodingBit+1) - 1. An example of incremental dictionary table is shown in Table 88 — An example of incremental dictionary table (nKVCodingBit = 2). The dicAddressContext is the context for reading the values of bAddress.

Table 88 — An example of incremental dictionary table (nKVCodingBit = 2)

Incremental Mode	Dictionary Symbol
0	0
1	1
2	-1
3	2
4	-2
5	3
6	-3

nTrueOne: This boolean value indicates if 1 is used as the true value in the position index. If nTrueOne is 0, then value 0 in the position index is interpreted as true.

bAddrIndex: This boolean value indicates which incremental dictionary symbol is used for each component of each vertex. The dicIndexContext is the context for reading the values of bAddrIndex.

8.9.3.7 CoordIOccurrenceMode

8.9.3.7.1 Syntax

```
class CoordIOccurrenceMode (unsigned int nKVCodingBit, int nNumberOfKey, int c) {
   int i, k;
   unsigned int bIndexOfDic[nNumberOfKey] = 0;
   for(i = 0; i < nNumberOfKey; i++) {
      if(bIndexOfDic[i] == 0) {
         bIndexOfDic[nNumberOfKey] = 1;
         if(c == 0)
            decodeSignedQuasiAAC(&nQKV[i], nKVCodingBit+1,
               kvSignContext, kvXContext);
         else if(c == 1)
            decodeSignedQuasiAAC(&nQKV[i], nKVCodingBit+1,
               kvSignContext, kvYContext);
         else if(c == 2)
            decodeSignedQuasiAAC(&nQKV[i], nKVCodingBit+1,
               kvSignContext, kvZContext);
         qf_decode(&bSoleKV, dicSoleKVContext);
         if(bSoleKV == 0) {
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            qf_decode(&nTrueOne, dicOneContext);
            for(k = i+1; k < nNumberOfKey; k++) {
               if(bIndexOfDic[k] == 0) {
                  int bDicIndex;
                  qf_decode(&bDicIndex, dicIndexContext);
                  if(bDicIndex == nTrueOne)
                     bIndexOfDic[k] = 1;
               }
            }
         }
      }
   }
}
```

8.9.3.7.2 Semantics

nQKV: This integer array contains all of the occurrence dictionary symbols, which are the quantized keyValues. The kvXContext, kvYContext and kvZContext are the contexts for reading the values of nQKV. The kvSignContext is the context for reading the signs of nQKV.

bSoleKV: This boolean value indicates whether the symbol occurs only one time. If so, bSoleKV is 1. The dicSoleKVContext is the context for reading the value of bSoleKV.

bDicIndex: This boolean value indicates which occurrence dictionary symbol is used for each component of each vertex. The dicIndexContext is the context for reading the values of bDicIndex.

8.9.3.8 Decoding Process

8.9.3.8.1 Overview

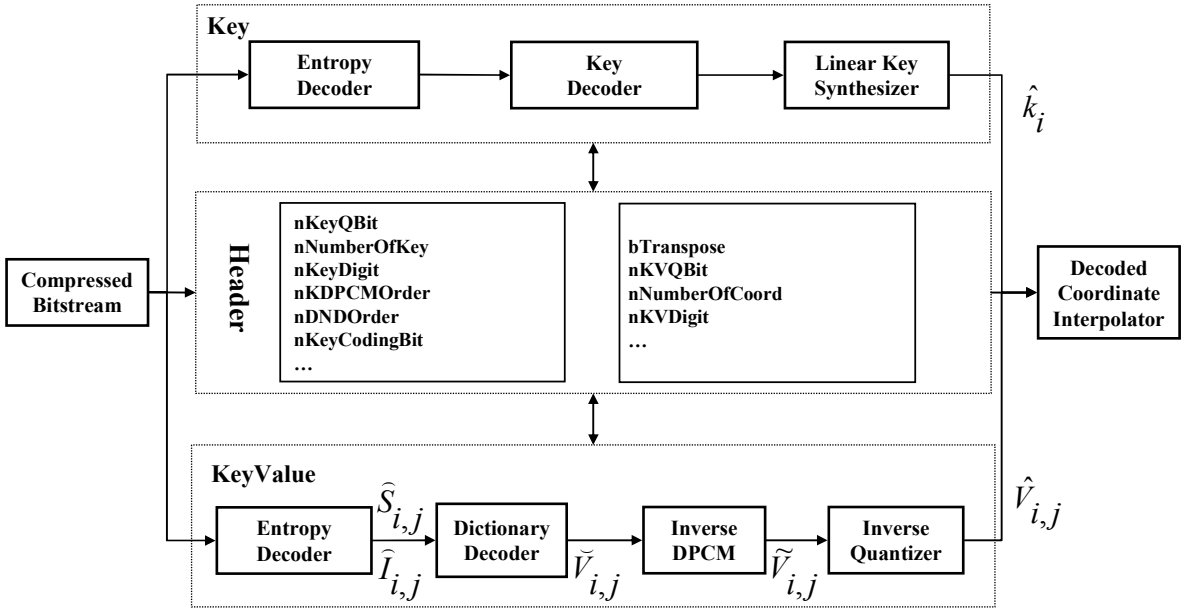


Figure 48 — Decoder for compressed coordinate interpolator

Figure 48 shows the block diagram of the decoder for compressed coordinate interpolator. It consists of the decoders for Key, Key Header, KeyValue Header and KeyValue. Decoding process of Key data is described in subclause **8.9.2.6**. In the following subclause, the decoding process of the KeyValue is described.

The keyValue data are float values in the structure of N×M matrix, where N is the number of keys and M is the number of vertices. In Figure 48, keys are indexed with i and vertices are indexed with j. The matrix structure of keyValue in the coordinate interpolator is shown in Table 89 — Matrix structure of keyValue in coordinate interpolator.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 89 — Matrix structure of keyValue in coordinate interpolator

	1	2	...i	M
1	x(1,1), y(1,1), z(1,1)	x(1,2), y(1,2), z(1,2)	...	x(1,M), y(1,M), z(1,M)
2	x(2,1), y(2,1), z(2,1)	x(2,2), y(2,2), z(2,2)	...	x(2,M), y(2,M), z(2,M)
. . . j	...	...	...x(i,j), y(i,j), z(i,j)	...
N	x(N,1), y(N,1), z(N,1)	x(N,2), y(N,2), z(N,2)	...	x(N,M), y(N,M), z(N,M)

8.9.3.8.2 Key Value Decoder

In this subclause, the decoding process of KeyValue for coordinate interpolator is described. It is comprised of the following steps.

- Entropy Decoding
- Dictionary Decoding
- Inverse DPCM
- Inverse Quantization

8.9.3.8.2.1 Entropy Decoder

The entropy decoder retrieves $\widehat{S}_{i,j}$ and $\widehat{I}_{i,j}$ from the bitstream and passes these values into the dictionary decoder.

$\widehat{S}_{i,j}$ is the dictionary symbol. $\widehat{I}_{i,j}$ is the position index. The flow of the entropy decoder is shown in Figure 49.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

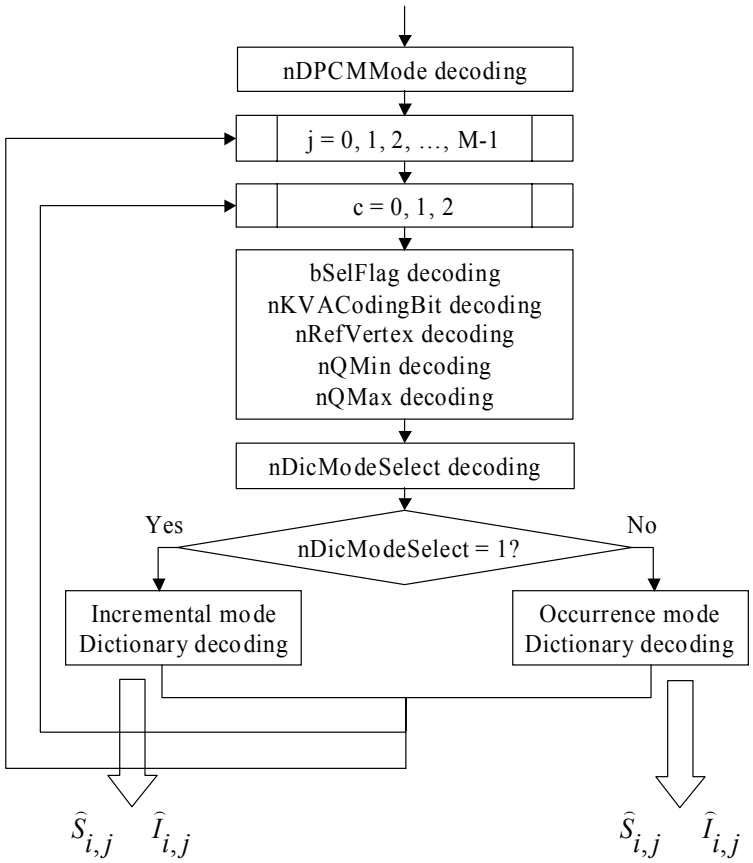


Figure 49 — Flowchart of entropy decoding

In the entropy decoding process, DPCM modes are decoded first.

In the next process, several arrays are decoded from the bitstream, which are bSelFlag, nKVACodingBit, nQMin, nQMax. bSelFlag array has default value of 1 and nKVACodingBit has default value of 0. If bSelFlag value remains 1, then the other arrays (nKVACodingBit, nQMin, nQMax) may be read. However, if the value of bSelFlag is changed to 0, the entropy decoder only decodes nQMin array. After decoding these arrays, the entropy decoder decodes nDicModeSelect which is the flag of dictionary mode. If the flag is 1, Incremental mode is selected. Otherwise, Occurrence mode is selected.

The bitstream structure of each component of each vertex to be decoded is as follows.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

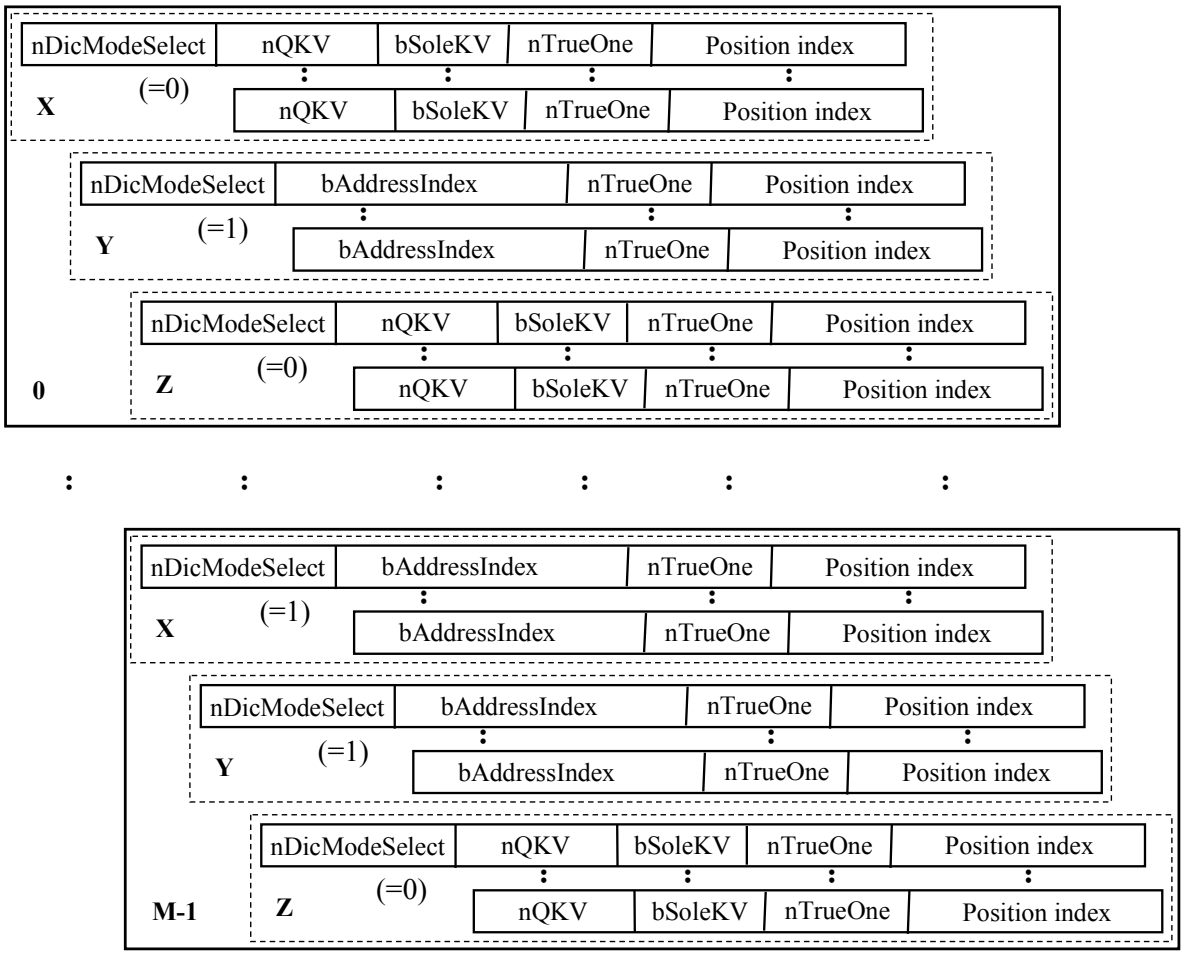


Figure 50 — An example of the bitstream structure of each component of each vertex for compressed coordinate interpolator.

According to nDicModeSelect value, there are two possible types of bitstream. If the nDicModeSelect is 0, the bitstream structure of the occurrence mode is used. Otherwise, the bitstream structure of the incremental mode is used. Figure 50 shows an example of the bitstream structure for compressed coordinate interpolator.

8.9.3.8.2.2 Dictionary Decoder

After the above stage, dictionary decoder is performed. There are two types of dictionary decoder: Incremental dictionary decoder and Occurrence dictionary decoder. For the DPCM mode decoding, only the incremental dictionary decoder is used. And for the keyValue decoding, either of the dictionary decoder can be used.

In case of keyValue decoding, the dictionary decoder is only performed for each component of each vertex with the quantization selection flag value (bSelFlag) of 1.

The dictionary decoder receives the dictionary symbols ($\widehat{S}_{i,j}$) and position indexes ($\widehat{I}_{i,j}$) as the input and generates the differentiated keyValues as the output. In case of occurrence mode, the dictionary is decoded in the order of occurrence of symbols for each component of each vertex. When in incremental mode, the dictionary is decoded in the incremental order of symbols for each component of each vertex. The dictionary decoder also uses the position index for each symbol in the dictionary. The output of the dictionary decoder is the differentiated keyValue.

The following example shows the method for decoding differentiated keyValue in case of occurrence mode.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

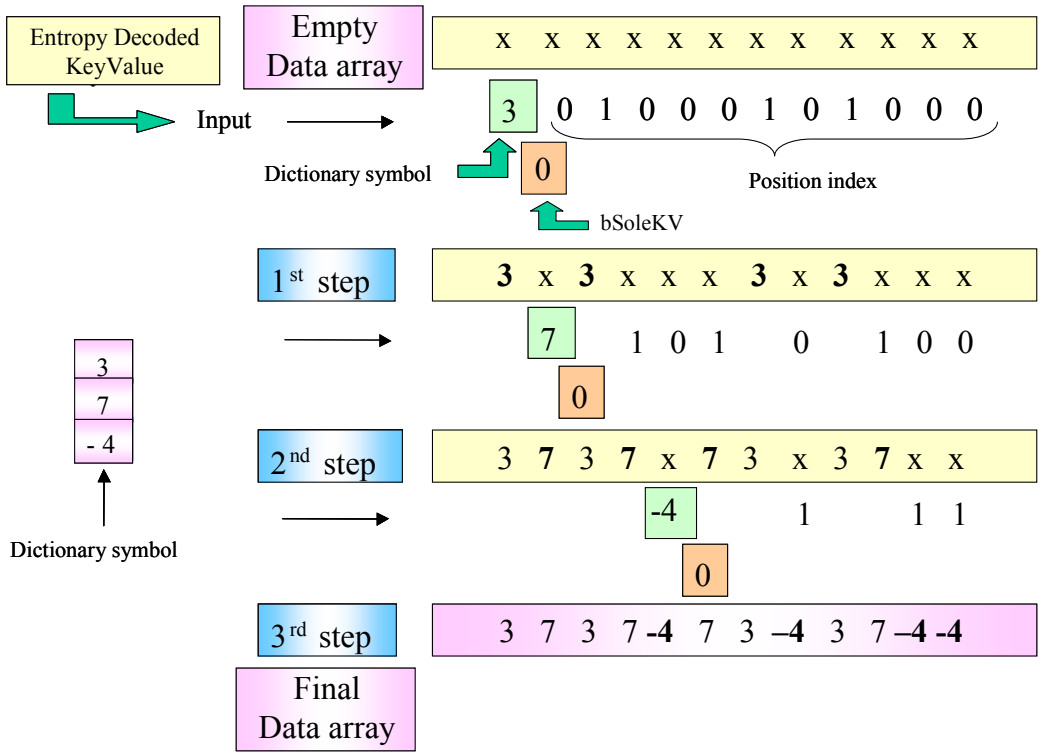


Figure 51 — An example of occurrence mode decoding

In Figure 51, the dictionary table receives the symbols (3, 7, -4) in the order of their occurence. For each symbol in the dictionary table, the position indices are received as (0 1 0 0 0 1 0 1 0 0 0), (1 0 1 0 1 0 0) and (1 1 1). In the occurrence mode, the first step is to retrieve the first symbol [3] and finds out the positions where the symbol occurs. The positions will be found by the value of 1 in the position index . Therefore, the intermediate result after retrieving the position index for symbol [3] is (3 x 3 x x x 3 x 3 x x x). Then, the dictionary decoder proceeds with the next symbol [7]. When retrieving the position index for [7], the positions that are found for the first symbol are not considered. Therefore, the position index for [7] is (1 0 1 0 1 0 0), instead of (0 1 0 1 0 0 0 1 0 0) and the intermediate result after this is (3 7 3 7 x 7 3 x 3 7 x x). Finally, the position index for the final symbol [-4] is (1 1 1) and the result of the dictionary decoder is generated as (3 7 3 7 -4 7 3 -4 3 7 –4 -4).

However, position indices for a symbol does not exist if bSoleKV is true, because this indicates that the dictionary symbol occurs only once.

The following example shows the method for decoding differentiated keyValue in case of incremental mode.

© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

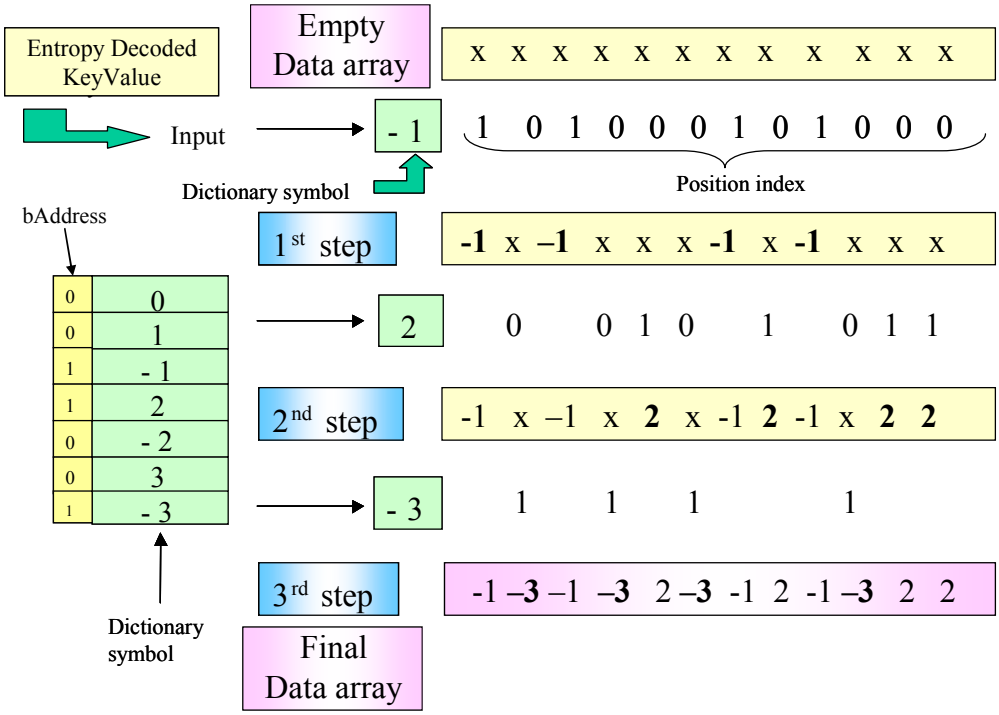


Figure 52 — An example of incremental mode decoding

In case of incremental mode, the flags for each symbol in the dictionary are retrieved. These flags indicates if the corresponding dictionary symbol is used. The order of dictionary symbol is the incremental order of the absolute values and positive value preceeding the negative value (i.e. 0, 1, -1, 2, -2, 3, -3 and so on). The size of incremental dictionary table is $2^{(nKVCodingBit+1)}$-1, where nKVCodingBit is the quantization bits retrieved from nKVACodingBit array. If the symbol flags are (0 0 1 1 0 0 1), then the symbols used in this dictionay are -1, 2 and -3. These symbols will have the corresponding position indices as follows: (1 0 1 0 0 0 1 0 1 0 0 0), (0 0 1 0 1 0 1 1) and (1 1 1 1) correspondingly.

Starting from the first symbol [-1], it's position indices (1 0 1 0 0 0 1 0 1 0 0 0) are used to achieve the intermediate result as (-1 x -1 x x x -1 x -1 x x x). Then the second symbol [2] are filled into the empty slots according to it's position indices (0 0 1 0 1 0 1 1). Thus, the intermediate result after the second symbol is filled becomes (-1 x -1 x 2 x -1 2 -1 x 2 2). Finally, the last symbol [-3] is filled to the empty slots, using (1 1 1 1) and the final result of the dictionary is generated as (-1 -3 -1 -3 2 -3 -1 2 -1 -3 2 2).

The DPCM mode data are decoded by incremental dictionary decoder as described above. However, the dictionary symbols are the combination of DPCM mode for each component in a vertex. Therefore, the size of the dictionary table is fixed to 27.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2005 – All rights reserved
Not for Resale

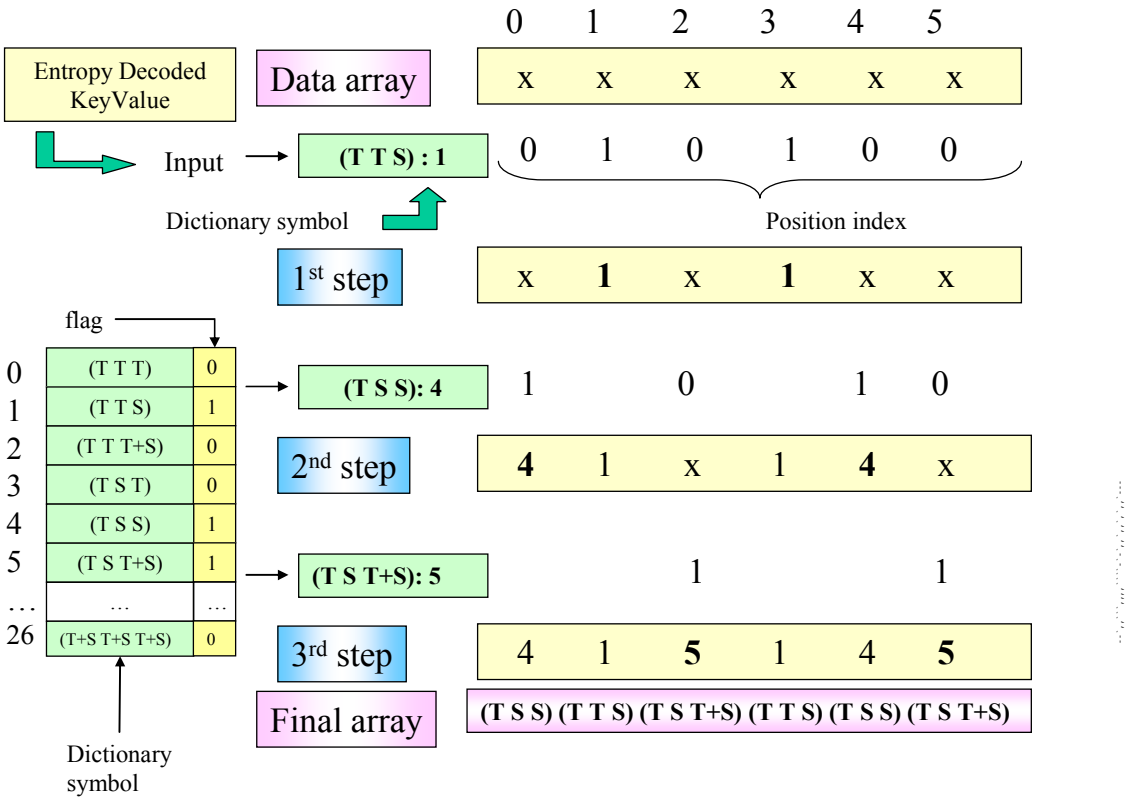


Figure 53 — An example of DPCM mode decoding

8.9.3.8.2.3 Inverse DPCM

After the above stage, the next decoding process is inverse DPCM depending on the DPCM modes and the reference vertices. There are three ways to perform inverse DPCM. Those are Inverse Temporal(T) DPCM, Inverse Spatial(S) DPCM and Inverse Spatiotemporal(T+S) DPCM. Let $\tilde{V}_{i,j}$ be the quantized keyValue data of i-th key and j-th vertex. Let $\breve{V}_{i,j}$ be the difference value of i-th key and j-th vertex in the keyValue. The followings are the equations for each type of inverse DPCM.

Inverse T. DPCM: $\tilde{V}_{i,j} = \breve{V}_{i,j} + \tilde{V}_{i-1,j}$

Inverse S. DPCM: $\tilde{V}_{i,j} = \breve{V}_{i,j} + \tilde{V}_{i,\mathrm{Ref}}$

Inverse T+S. DPCM: $\tilde{V}_{i,j} = \breve{V}_{i,j} + \tilde{V}_{i-1,\mathrm{j}} + (\tilde{V}_{i,\mathrm{Ref}} - \tilde{V}_{i-1,\mathrm{Ref}})$

Ref is the index of a reference vertex. During inverse DPCM process, inverse circular quantization is performed for the current reconstructed keyValue. The method of inverse circular quantization can be found in subclause 8.9.5.6.3.

8.9.3.8.2.4 Inverse Quantization

After inverse DPCM, inverse quantization is performed. If the quantization selection flag is 0, then the corresponding component of the vertex has only one quantized value (QMin). Also, if the quantization selection flag value is not 0 and the nKVCodingBit value is 0, then the decoded keyValue is set to 0. In this case, 0 is assigned to the corresponding component of the vertex.

The inverse quantization is performed using the following equation so that the quantized keyValue (v) are inverse quantized.

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$\hat{V}_{i,j,x} = fMin_X + \frac{\tilde{V}_{i,j,x}}{(2^{nKVQBits} - 1)} * fMax$$

$$\hat{V}_{i,j,y} = fMin_Y + \frac{\tilde{V}_{i,j,y}}{(2^{nKVQBits} - 1)} * fMax$$

$$\hat{V}_{i,j,z} = fMin_Z + \frac{\tilde{V}_{i,j,z}}{(2^{nKVQBits} - 1)} * fMax$$

fMin is the minimum value and fMax is the maximum range of the original keyValue and nKVQBits is the quantization bit size for inverse quantization.

After inverse quantization, if the current mode is transpose mode, the matrix generated after inverse quantization is the diagonally transposed matrix of the original keyValue matrix. Therefore, if transpose mode is true, then the matrix must be diagonally transposed again in order to obtain the final matrix of keyValue. For example, given the original keyValue matrix as shown in Table 89 — Matrix structure of keyValue in coordinate interpolator, if the transpose mode is true, then the result matrix after the inverse quantization is shown in Table 90 — Diagonally transposed keyValue matrix.

Table 90 — Diagonally transposed keyValue matrix

	1	2	...j	N
1	x(1,1), y(1,1), z(1,1)	x(2,1), y(2,1), z(2,1)	...	x(N,1), y(N,1), z(N,1)
2	x(1,2), y(1,2), z(1,2)	x(2,2), y(2,2), z(2,2)	...	x(N,2), y(N,2), z(N,2)
. . . i	...	...	...x(j,i), y(j,i), z(j,i)	...
M	x(1,M), y(1,M), z(1,M)	x(2,M), y(2,M), z(2,M)	...	x(N,M), y(N,M), z(N,M)

8.9.4 Orientation Interpolator Decoding

8.9.4.1 CompressedOrientationInterpolator

8.9.4.1.1 Syntax

```
class CompressedOrientationInterpolator {
   KeyHeader  kHeader;
   OriIKeyValueHeader oriIKVHeader;
   qf_start();
   aligned(8) KeySelectionFlag ksFlag(kHeader, oriIKVHeader.bPreserveKey);
   Key k(kHeader);
   if(oriIKVHeader.nKVDPCMOrder == 0)//1st order DPCM
      OriIDPCMKeyValue oriIDPCMKeyValue(oriIKVHeader.oriIDPCMKVHeader,
ksFlag.nNumberOfKeyValue-1);
   else //2nd order DPCM
      OriIDPCMKeyValue oriIDPCMKeyValue(oriIKVHeader.oriIDPCMKVHeader,
ksFlag.nNumberOfKeyValue-2);
}
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.9.4.1.2 Semantics

This is a top class for reading the compressed bitstream of orientation interpolator. KeyHeader, KeySelectionFlag, and Key are the classes for reading key information from the bitstream, which corresponds to key field data in conventional orientation interpolator. OriIKeyValueHeader and OriIDPCMKeyValue are the classes for reading keyValue information corresponding to keyValue field data in conventional orientation interpolator.

The function qf_start() is used for initializing arithmetic decoder before reading encoded part of the bitstream with AAC(Adaptive Arithmetic Coding) (see subclause 7.13.10.1 of ISO/IEC 14496-2:2004)

8.9.4.2 OriIKeyValueHeader

8.9.4.2.1 Syntax

```
class OriIKeyValueHeader () {
   bit(1) bPreserveKey;
   unsigned int(5) nKVQBit;
   bit(1) nKVDPCMOrder;
   OriIDPCMKeyValueHeader oriIDPCMKVHeader(nKVQBit, nKVDPCMOrder);
}
```

8.9.4.2.2 Semantics

bPreserveKey: This flag indicates if the current decoding mode is key preserving orpath preserving mode. If the flag is true, then the current decoding mode is key preserving mode.

nKVQBit: This value indicates the quantization bit size of the keyValue data.

nKVDPCMOrder: This value indicates the order of inverse DPCM that is used in keyValue decoding. nKVDPCMOrder=0 indicates the 1st order inverse DPCM and nKVDPCMOrder=1 indicates the 2nd order inverse DPCM.

8.9.4.3 OriDPCMKeyValueHeader

8.9.4.3.1 Syntax

```
class OriIDPCMKeyValueHeader (int nKVQBit, unsigned int nKVDPCMOrder) {
   unsigned int(nKVQBit-1) firstQKV_S;
   bit(1) nFirstXSign;
   unsigned int(nKVQBit-1) firstQKV_X;
   bit(1) nFirstYSign;
   unsigned int(nKVQBit-1) firstQKV_Y;
   bit(1) nFirstZSign;
   unsigned int(nKVQBit-1) firstQKV_Z;
   if (nKVDPCMOrder==1) {    //2nd order DPCM
      bit(1) nSecondXSign;
      unsigned int(nKVQBit-1) secondQKV_X;
      bit(1) nSecondYSign;
      unsigned int(nKVQBit-1) secondQKV_Y;
      bit(1) nSecondZSign;
      unsigned int(nKVQBit-1) secondQKV_Z;
      bit(1) bIsMoreTwoKVs;
   }
   if (nKVDPCMOrder==0 || bIsMoreTwoKVs==1) {
      bit(1) x_keyvalue_flag;
      OriIKeyValueCodingBit oriIKVCodingBit_X(x_keyvalue_flag, nKVQBit);
      bit(1) y_keyvalue_flag;
      OriIKeyValueCodingBit oriIKVCodingBit_Y(y_keyvalue_flag, nKVQBit);
      bit(1) z_keyvalue_flag;
      OriIKeyValueCodingBit oriIKVCodingBit_Z(z_keyvalue_flag, nKVQBit);
   }
}
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.9.4.3.2 Semantics

firstQKV_S, firstQKV_X, firstQKV_Y, firstQKV_Z: These values indicate the first value of each component *(s, x, y, z)* of quantized keyValue in quaternion form.

nFirstXSign, nFirstYSign, nFirstZSign: These values indicate the sign of the first value of three components *(x, y, z)* of quantized keyValue.

secondQKV_X, secondQKV_Y, secondQKV_Z: These values indicate the second value of three component *(x, y, z)* of quantized keyValue in quaternion form.

nSecondXSign, nSecondYSign, nSecondZSign: These values indicate the sign of the second value of three components *(x, y, z)* of quantized keyValue.

bIsMoreTwoKVs: This flag indicates if there are more than two keyValues to be decoded, in case of 2nd order inverse DPCM.

x_keyvalue_flag, y_keyvalue_flag, z_keyvalue_flag: These flags indicate, for three components *(x, y, z)*, if all of the quantized values are the same.

8.9.4.4 OriIKeyValueCodingBit

8.9.4.4.1 Syntax

```
class OriIKeyValueCodingBit (unsigned int flag_bit, int nKVQBit) {
   int count = (int)(log10(nKVQBit)/log10(2)) + 1;
   if(flag_bit == 0) {
      unsigned int(count) nKVCodingBit;
      if(nKVCodingBit == 1)
         unsigned int(nKVCodingBit) nAllKeyValue;
      else {
         bit(1) nSign;
         unsigned int(nKVCodingBit-1) nAllKeyValue;
      }
   } else {
      bit(1) bIsUnaryAAC;
      if(bIsUnaryAAC != 1)
         unsigned int(count) nKVCodingBit;
   }
}
```

8.9.4.4.2 Semantics

nKVCodingBit: This value indicates the actual number of bits used for coding components *x, y* and *z*, of all quaternion except intra keyValue (firstQKV_S, firstQKV_X, firstQKV_Y, firstQKV_Z and secondQKV_X, secondQKV_Y, secondQKV_Z in OriIDPCMKeyValueHeader class).

nAllKeyValue: This value indicates a quantized value of each component of keyValue for all the keys when its flag_bit is set to 0.

nSign: This value indicates the sign of nAllKeyValue.

bIsUnaryAAC: This flag indicates which adaptive arithmetic decoding method is used for the quantized values of each of the three components (*x, y* and *z*). If true, then unary AAC is used. Otherwise, binary AAC is used.

8.9.4.5 OriIDPCMKeyValue

8.9.4.5.1 Syntax

```
class OriIDPCMKeyValue(OriIDPCMKeyValueHeader kvHeader, int nNumKV) {
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   int i;
   if(kvHeader.x_keyvalue_flag != 0) {
      if(kvHeader.oriIKVCodingBit_X.bIsUnaryAAC == 1)
         for(i = 0;i < nNumKV;i++)
            decodeUnaryAAC(&DeltaKeyValue[i].x, kVXSignContext, kVXUnaryContext);
      else
         for(i = 0;i < nNumKV;i++)
            decodeSignedAAC(&DeltaKeyValue[i].x, kvHeader.oriIKVCodingBit_X.nKVCodingBit,
kVXSignContext, kVXContext);
   }
   if(kvHeader.y_keyvalue_flag != 0) {
      if(kvHeader.oriIKVCodingBit_Y.bIsUnaryAAC == 1)
         for(i = 0;i < nNumKV;i++)
            decodeUnaryAAC(&DeltaKeyValue[i].y, kVYSignContext, kVYUnaryContext);
      else
         for(i = 0;i < nNumKV;i++)
            decodeSignedAAC(&DeltaKeyValue[i].y, kvHeader.oriIKVCodingBit_Y.nKVCodingBit,
kVYSignContext, kVYContext);
   }
   if(kvHeader.z_keyvalue_flag != 0) {
      if(kvHeader.oriIKVCodingBit_Z.bIsUnaryAAC == 1)
         for(i = 0;i < nNumKV;i++)
            decodeUnaryAAC(&DeltaKeyValue[i].z, kVZSignContext, kVZUnaryContext);
      else
         for(i = 0;i < nNumKV;i++)
            decodeSignedAAC(&DeltaKeyValue[i].z, kvHeader.oriIKVCodingBit_Z.nKVCodingBit,
kVZSignContext, kVZContext);
   }
}
```

8.9.4.5.2 Semantics

DeltaKeyValue: This array stores the quantized values related to three components *(x, y, z)* of key Value in quaternion form. It is arithmetic decoded from the bitstream by the function decodeUnaryAAC or decodeSignedAAC. The decoding process of these functions is explained in detail in subclause 8.9.6.

kVXSignContext, kVYSignContext, kVZSignContext: These are the contexts that are used for decoding the signs of three components of DeltaKeyValue in the function decodeUnaryAAC or decodeSignedAAC.

kVXUnaryContext, kVYUnaryContext, kVZUnaryContext: These are the contexts that are used for decoding the value of three components of DeltaKeyValue in the function decodeUnaryAAC.

kVXContext, kVYContext, kVZContext: These are the contexts that are used for decoding the value of three components of DeltaKeyValue in the function decodeSignedAAC.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.9.4.6 Decoding Process

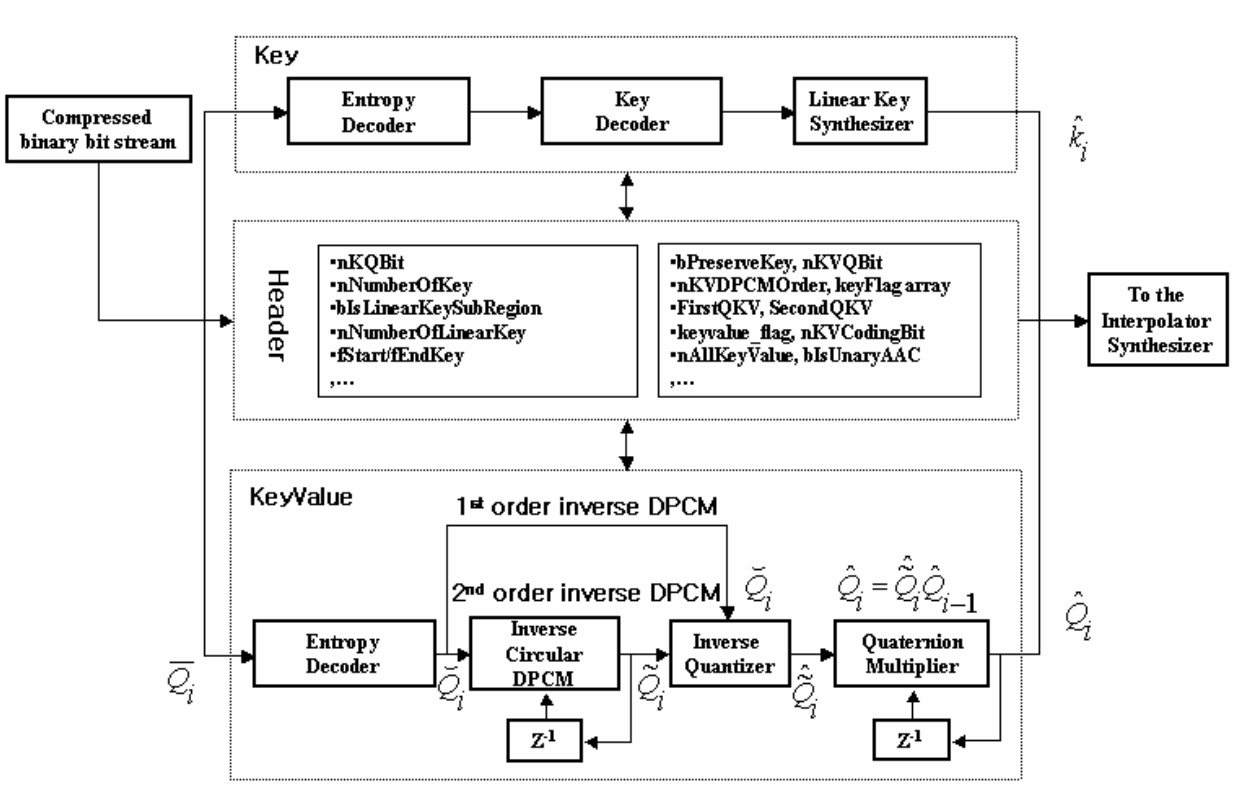


Figure 54 — Decoder structure for compressed orientation interpolator

Figure 54 shows the block diagram of the decoder structure of the compressed orientation interpolator. It is comprised of 3 parts.

— Header (for Keys and keyValues) decoder

— Key decoder

— keyValue decoder.

Decoding process of key header and key data is described in subclause 8.9.2.6. In the following subclauses, the decoding process of keyValue header and keyValue for orientation interpolator is described.

8.9.4.6.1 KeyValue Header decoder

8.9.4.6.1.1 Key preserving and path preserving mode

In the keyValue header information, there is a bPreserveKey flag(in OrilKeyValueHeader class), which indicates if the current decoding is key preserving mode or path preserving mode. The key preserving mode is for the case where a random access may be required for the data in interpolator node of the BIFS Scene using BIFS commands (replace, delete, insert, etc.). In this case, the keys in the interpolator node are preserved. However, the number of keyValues may be reduced and the remaining points are indicated by keyFlag array (in KeySelectionFlag class). For example, given an interpolator curve as shown in Figure 55, and when 4 points have been selected, the key selection flag array (keyFlag array in KeySelectionFlag class) will have the values as shown in Table 91 — Values of key selection flag array after Key Selection process. Therefore, in case of key preserving mode, decoded key selection flag array is used to find out the existence of keyValue for each key. If the keyValue of a key is missing, then it can be restored through spherical linear interpolation of the existing neighbor keyValues.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

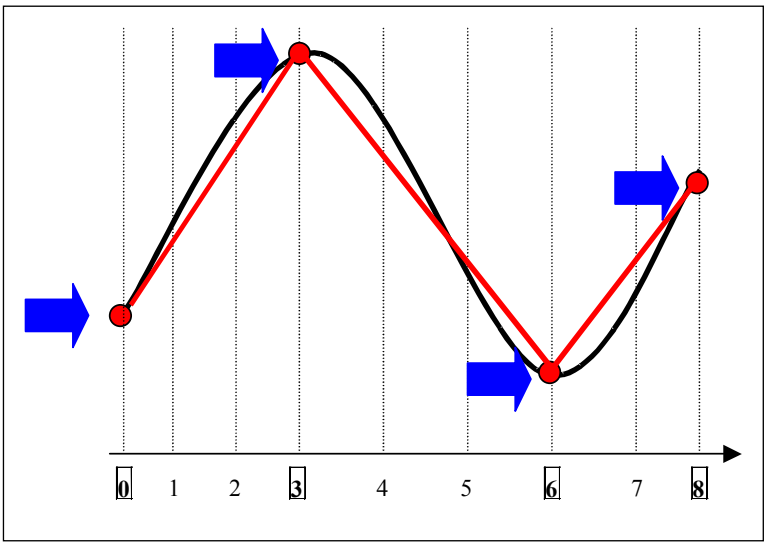


Figure 55 — Key Selection process in Interpolator

Table 91 — Values of key selection flag array after Key Selection process

Original Points	P0	P1	P2	P3	P4	P5	P6	P7	P8
Key selection Flag	1	0	0	1	0	0	1	0	1

The animation path preserving mode is for the case where interpolator node is only used for describing the interpolation of the path of an animation and random access is not necessary. In this case, it is allowed to remove some keys and corresponding keyValues in the interpolator node within a given error range for more efficient compression and the keyFlag array will not exist in the encoded bitstream.

8.9.4.6.1.2 Decoding of Intra keyValues

In the keyValue header information, there are first and second quantized keyValues in quaternion form according to the order of inverse DPCM and several flags that are needed to decode the remaining keyValues.

The first and second quantized keyValues are reconstructed using inverse quantization and quaternion multiplication. Then, they are converted into angular displacement from quaternion.

If the order of inverse DPCM is 1(e.g. nKVDPCMOrder is set to 0), only the first quantized keyValue is contained in the keyValue header information. If the quaternion of first inverse quantized keyValue is $\hat{Q}_0 = (\hat{q}_{0,0}, \hat{q}_{0,1}, \hat{q}_{0,2}, \hat{q}_{0,3})^T$, the first inverse quantized keyValue will be

$$\hat{q}_{0,0} = \tan\left(\frac{\pi}{4} * \left(\frac{|firstQKV_S|}{2^{nKVQBit-1}-1}\right)\right)$$

$$\hat{q}_{0,1} = \tan\left(\frac{\pi}{4} * \left(xSign * \frac{|firstQKV_X|}{2^{nKVQBit-1}-1}\right)\right)$$

$$\hat{q}_{0,2} = \tan\left(\frac{\pi}{4} * \left(ySign * \frac{|firstQKV_Y|}{2^{nKVQBit-1}-1}\right)\right)$$

$$\hat{q}_{0,3} = \tan\left(\frac{\pi}{4} * \left(zSign * \frac{|firstQKV_Z|}{2^{nKVQBit-1}-1}\right)\right)$$

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In these equations, xSign has the value 1 when nFirstXSign (in OriIDPCMKeyValueHeader class) is 1. Otherwise it will have the value -1. The same applies to ySign with nFirstYSign and zSign with nFirstZSign.

Finally, this reconstructed quaternion is converted into angular displacement to be used in orientation interpolator. The reconstructed angular displacement from the keyValue can be represented with a 4-dimensional vector as $(\hat{x}_i, \hat{y}_i, \hat{z}_i, \hat{\theta}_i)^T$, where *i* represents the current key, $(\hat{x}_i, \hat{y}_i, \hat{z}_i)$ represents the vector of the rotation axis, and $\hat{\theta}_i$ represents the rotation angle to the counter-clockwise direction. Therefore, this reconstructed quaternion is converted into angular displacement in the Interpolator Synthesizer, using following equations.

$$\hat{x}_0 = \hat{q}_{0,1} * \frac{1}{\sin(\hat{\theta}_0/2)}$$
$$\hat{y}_0 = \hat{q}_{0,2} * \frac{1}{\sin(\hat{\theta}_0/2)} \quad \text{..... Eq. 1}$$
$$\hat{z}_0 = \hat{q}_{0,3} * \frac{1}{\sin(\hat{\theta}_0/2)}$$
$$\hat{\theta}_0 = \arccos(\hat{q}_{0,0}) * 2$$

If the order of inverse DPCM is 2 (e.g. nKVDPCMOrder is 1), the first and second quantized keyValues are contained in the keyValue header information. In this case, the first keyValue is reconstructed with the same rule described above. For reconstructing the second keyValue, a slightly different rule is applied, because only the three components (secondQKV_X, secondQKV_Y, secondQKV_Z) of second quantized keyValue are transmitted through the encoded bitstream and also these components are not intra keyValue but differentiated keyValue($\tilde{Q}_1 = (\tilde{q}_{1,1}, \tilde{q}_{1,2}, \tilde{q}_{1,3})$) from first keyValue. If the quaternion of second differentiated and inverse quantized keyValue is $\hat{\tilde{Q}}_1 = (\hat{\tilde{q}}_{1,0}, \hat{\tilde{q}}_{1,1}, \hat{\tilde{q}}_{1,2}, \hat{\tilde{q}}_{1,3})^T$, it will have

$$\hat{\tilde{q}}_{1,0} = \sqrt{1 - (\hat{\tilde{q}}_{1,1}^2 + \hat{\tilde{q}}_{1,2}^2 + \hat{\tilde{q}}_{1,3}^2)}$$
$$\hat{\tilde{q}}_{1,1} = \tan\left(\frac{\pi}{4} * \left(secondXSign * \frac{|secondQKV_X|}{2^{nKVQBit-1} - 1} \right)\right)$$
$$\hat{\tilde{q}}_{1,2} = \tan\left(\frac{\pi}{4} * \left(secondYSign * \frac{|secondQKV_Y|}{2^{nKVQBit-1} - 1} \right)\right)$$
$$\hat{\tilde{q}}_{1,3} = \tan\left(\frac{\pi}{4} * \left(secondZSign * \frac{|secondQKV_Z|}{2^{nKVQBit-1} - 1} \right)\right)$$

In these equations, secondXSign has the value 1 when nSecondXSign (in OriIDPCMKeyValueHeader class) is 1. Otherwise it will have the value -1. The same applies to secondYSign with nSecondYSign and secondZSign with nSecondZSign.

And if the reconstructed quaternion of second keyValue is $\hat{Q}_1 = (\hat{q}_{1,0}, \hat{q}_{1,1}, \hat{q}_{1,2}, \hat{q}_{1,3})^T$, $\hat{Q}_1$ will be calculated by the quaternion multiplication between $\hat{\tilde{Q}}_1$ and $\hat{Q}_0$. Therefore, the equation is

$$\hat{Q}_1 = \hat{\tilde{Q}}_1 \hat{Q}_0$$

This reconstructed quaternion is also converted into angular displacement in the Interpolator Synthesizer, using Eq. 1.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.9.4.6.2 keyValue decoder

8.9.4.6.2.1 Entropy Decoder

After reading the header information and key data, the keyValue data ($\overline{Q}_i = (\overline{q}_{i,1}, \overline{q}_{i,2}, \overline{q}_{i,3})$) in the encoded bitstream are passed to the entropy decoder (adaptive arithmetic decoder), if the keyvalue_flag (x_keyvalue_flag, y_keyvalue_flag and z_keyvalue_flag in OriIDPCMKeyValueHeader class) is set to 1. Otherwise, if the keyvalue_flag is set to 0, it means that all the remaining quantized and differentiated values of each component of every keyValues (except first and second keyValue according to the order of inverse DPCM) have the same values respectively that are represented by nAllKeyValues (in OriIKeyValueCodingBit class). Therefore, there is no keyValue data in the encoded bitstream that are passed to the entropy decoder.

If $\breve{Q}_i = (\breve{q}_{i,1}, \breve{q}_{i,2}, \breve{q}_{i,3}) = (DeltaKeyValue[i].x, DeltaKeyValue[i].y, DeltaKeyValue[i].z)$, is defined as the output of the entropy decoder, then it is decoded by the following function.

$$\begin{cases} \breve{q}_{i,1} = \text{oriIKVCodingBit_X.nAllKeyValue}, & (if\ x_keyvalue_flag == 0) \\ Entropy_Decoder(\bullet): \breve{q}_{i,1} = f(\overline{q}_{i,1}), & (if\ x_keyvalue_flag == 1) \end{cases}$$

$$\begin{cases} \breve{q}_{i,2} = \text{oriIKVCodingBit_Y.nAllKeyValue}, & (if\ y_keyvalue_flag == 0) \\ Entropy_Decoder(\bullet): \breve{q}_{i,2} = f(\overline{q}_{i,2}), & (if\ y_keyvalue_flag == 1) \end{cases}$$

$$\begin{cases} \breve{q}_{i,3} = \text{oriIKVCodingBit_Z.nAllKeyValue}, & (if\ z_keyvalue_flag == 0) \\ Entropy_Decoder(\bullet): \breve{q}_{i,3} = f(\overline{q}_{i,3}), & (if\ z_keyvalue_flag == 1) \end{cases}$$

$$(i = 0,..., nNumberOfKeyValue - 2 \quad or \quad i = 0,..., nNumberOfKeyValue - 3)$$

The function of entropy decoder to be used here is described in subclause 8.9.6. The parameter i goes up to nNumberOfKeyValue-2 in the case of 1st order inverse DPCM (nKVDPCMOrder=0) and goes up to nNumberOfKeyValue-3 in the case of 2nd order inverse DPCM (nKVDPCMOrder=1). The order of each component of quaternion within the bitstream is shown in Figure 56

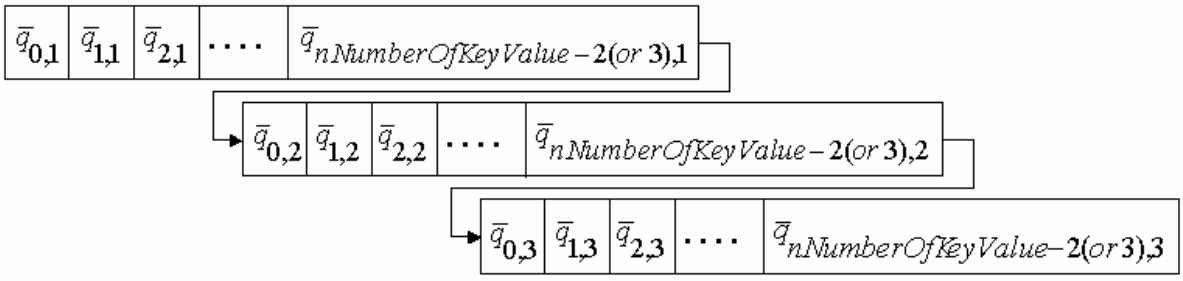


Figure 56 — The order of keyValue data in the bitstream

If bIsUnaryAAC (in OriIKeyValueCodingBit class) is set to 1, the keyValue data in the encoded bitstream passes the UnaryAAC decoder. Otherwise SignedAAC decoder. (see subclause 8.9.6)

After arithmetic decoding of keyValue, the output (DeltaKeyValue array in OriIDPCMKeyValue class) of arithmetic decoder is sent to Inverse circular DPCM or inverse Quantizer as shown in Figure 54. depending on the order of inverse DPCM.

8.9.4.6.2.2 1st Order Inverse DPCM

If the order of inverse DPCM is 1 (nKVDPCMOrder=0), the output of the arithmetic decoder is sent to Inverse Quantizer bypassing Inverse circular DPCM. If the output of arithmetic decoder is

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$\breve{Q}_i = (\breve{q}_{i,1}, \breve{q}_{i,2}, \breve{q}_{i,3}) = (DeltaKeyValue[i].x, DeltaKeyValue[i].y, DeltaKeyValue[i].z)$ and the quaternions of differentiated and inverse quantized keyValues are $\hat{\breve{Q}}_i = (\hat{\breve{q}}_{i,0}, \hat{\breve{q}}_{i,1}, \hat{\breve{q}}_{i,2}, \hat{\breve{q}}_{i,3})^T$, then $\hat{\breve{Q}}_i$ will have

$$\hat{\breve{q}}_{i,0} = \sqrt{1-(\hat{\breve{q}}_{i,1}^2 + \hat{\breve{q}}_{i,2}^2 + \hat{\breve{q}}_{i,3}^2)}$$

$$\hat{\breve{q}}_{i,j} = \tan(\frac{\pi}{4} * \left(\mathrm{sgn}(\breve{q}_{i\text{-}1,j}) * \frac{\left|\breve{q}_{i\text{-}1,j}\right|}{2^{nKVQBit-1}-1} \right))$$

$$(i = 1,...,nNumberOfKeyValue - 1,\ j = 1,2,3)$$

And if the reconstructed quaternions of these keyValues are $\hat{Q}_i = (\hat{q}_{i,0}, \hat{q}_{i,1}, \hat{q}_{i,2}, \hat{q}_{i,3})^T$, $\hat{Q}_i$ will be calculated by the quaternion multiplication between $\hat{\breve{Q}}_i$ and previously reconstructed quaternion in the Quaternion Multiplier in Figure 54. Therefore, the equation is

$$\hat{Q}_i = \hat{\breve{Q}}_i \hat{Q}_{i-1}$$

$$(i = 1,...,nNumberOfKeyValue - 1)$$

These reconstructed quaternions are also converted into angular displacements in the Interpolator Synthesizer, using Eq. 1

8.9.4.6.2.3 2nd Order Inverse DPCM

If the order of inverse DPCM is 2 (nKVDPCMOrder=1), the output of the arithmetic decoder is sent to Inverse circular DPCM in Figure 54. If the output of arithmetic decoder is $\breve{Q}_i = (\breve{q}_{i,1}, \breve{q}_{i,2}, \breve{q}_{i,3}) = (DeltaKeyValue[i].x, DeltaKeyValue[i].y, DeltaKeyValue[i].z)$ and the output of this inverse circular DPCM is $\widetilde{Q}_i = (\widetilde{q}_{i,1}, \widetilde{q}_{i,2}, \widetilde{q}_{i,3})$, then $\widetilde{Q}_i$ will have

$$ICDPCM(\bullet): \widetilde{q}_{i,j} = f(\breve{q}_{i-2,j}, \widetilde{q}_{i-1,j})$$

$$(i = 2,...,nNumberOfKeyValue - 1,\ j = 1,2,3)$$

The following is the C++ style syntactic description of function ICDPCM(inverse circular DPCM)

```
ICDPCM(int* curIDPCMKeyValue, int deltaKeyValue, int prevICDPCMKeyValue)
{
   int circularDelta;
   int tempIDPCMKeyValue;
   prevICDPCMKeyValue += ((1 << (nKVQBit-1))-1);

   if(deltaKeyValue >= 0.0)
      circularDelta = deltaKeyValue - ((1 << nKVQBit)-1);
   else
      circularDelta = deltaKeyValue + ((1 << nKVQBit)-1);

   tempIDPCMKeyValue = circularDelta + prevICDPCMKeyValue;

   if((tempIDPCMKeyValue >= 0.0) && (tempIDPCMKeyValue < ((1 << nKVQBit)-1)))
      *curIDPCMKeyValue = tempIDPCMKeyValue;
   else
      *curIDPCMKeyValue = deltaKeyValue + prevICDPCMKeyValue;

   *curIDPCMKeyValue -= ((1 << (nKVQBit-1))-1);
}
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The inverse circular DPCM chooses the value within quantization range between the value of deltaKeyValue and complement of deltaKeyValue (*circularDelta*).

The quaternions of differentiated and inverse quantized keyValues $\hat{\tilde{Q}}_i = (\hat{\tilde{q}}_{i,0}, \hat{\tilde{q}}_{i,1}, \hat{\tilde{q}}_{i,2}, \hat{\tilde{q}}_{i,3})^T$ can be calculated from the output of this inverse circular DPCM $\tilde{Q}_i = (\tilde{q}_{i,1}, \tilde{q}_{i,2}, \tilde{q}_{i,3})$. $\hat{\tilde{Q}}_i$ will have

$$\hat{\tilde{q}}_{i,0} = \sqrt{1 - (\hat{\tilde{q}}_{i,1}^2 + \hat{\tilde{q}}_{i,2}^2 + \hat{\tilde{q}}_{i,3}^2)}$$

$$\hat{\tilde{q}}_{i,j} = \tan\left(\frac{\pi}{4} * \left(\mathrm{sgn}(\tilde{q}_{i,j}) * \frac{|\tilde{q}_{i,j}|}{2^{nKVQBit-1} - 1} \right)\right)$$

$$(i = 2,..., nNumberOfKeyValue - 1,\ j = 1,2,3)$$

If the reconstructed quaternions of these keyValues are $\hat{Q}_i = (\hat{q}_{i,0}, \hat{q}_{i,1}, \hat{q}_{i,2}, \hat{q}_{i,3})^T$, then $\hat{Q}_i$ will be calculated by the quaternion multiplication between $\hat{\tilde{Q}}_i$ and previously reconstructed quaternion in the Quaternion Multiplier in Figure 54. Therefore, the equation is

$$\hat{Q}_i = \hat{\tilde{Q}}_i \hat{Q}_{i-1}$$

$$(i = 2,..., nNumberOfKeyValue - 1)$$

These reconstructed quaternions are also converted into angular displacements in the Interpolator Synthesizer, using Eq. 1

8.9.5 Position Interpolator Decoding

8.9.5.1 CompressedPositionInterpolator

8.9.5.1.1 Syntax

```
class CompressedPositionInterpolator {
   KeyHeader kHeader;
   PosIKeyValueHeader posIKVHeader;
   qf_start();
   aligned(8) KeySelectionFlag ksFlag(kHeader, posIKVHeader.bPreserveKey);
   Key k(kHeader);
   PosIKeyValue posIKV(posIKVHeader, ksFlags.nNumberOfKeyValue);
}
```

8.9.5.1.2 Semantics

This is a top class for reading the compressed bitstream of position interpolator. It consists of KeyHeader, PosIKeyValueHeader, KeySelectionFlag, Key and PosIKeyValue. KeyHeader and PosIKeyValueHeader contain the header information to decode Key and PosIKeyValue. KeySelectionFlag has the flags, which indicate, for each key, if the corresponding keyValue is included in the PosIKeyValue. Finally, Key and PosIKeyValue class read the key and keyValue data respectively for position interpolator from the bitstream.

The function qf_start() is used for initializing arithmetic decoder before reading AAC encoded part of the bitstream (see subclause 7.13.10.1 of ISO/IEC 14496-2:2004).

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.9.5.2 PosIKeyValueHeader

8.9.5.2.1 Syntax

```
class PosIKeyValueHeader() {
   bit(1) bPreserveKey;
   unsigned int(5) nKVQBit;
   bit(1) x_keyvalue_flag;
   bit(1) y_keyvalue_flag;
   bit(1) z_keyvalue_flag;
   unsigned int(4) nKVDigit;
   int nBits = (int)(log10(nKVQBit)/log10(2)) + 1;
   if(x_keyvalue_flag == 1) {
      bit(1) nKVDPCMOrder_X;
      bit(1) bIsUnaryAAC_X;
      if(bIsUnaryAAC_X != 1) {
         unsigned int(nBits) nKVCodingBit_X;
         unsigned int(1) nStartIndex_X;
      }
   }
   if(y_keyvalue_flag == 1) {
      bit(1) nKVDPCMOrder_Y;
      bit(1) bIsUnaryAAC_Y;
      if(bIsUnaryAAC_Y != 1) {
         unsigned int(nBits) nKVCodingBit_Y;
         unsigned int(1) nStartIndex_Y;
      }
   }
   if(z_keyvalue_flag == 1) {
      bit(1) nKVDPCMOrder_Z;
      bit(1) bIsUnaryAAC_Z;
      if(bIsUnaryAAC_Z != 1) {
         unsigned int(nBits) nKVCodingBit_Z;
         unsigned int(1) nStartIndex_Z;
      }
   }
   if (bIsUnaryAAC_X != 1)
      if((nStartIndex_X == 1) && (x_keyvalue_flag == 1))
         unsigned int(nKVQBit) firstKV_X;
   if (bIsUnaryAAC_Y != 1)
      if((nStartIndex_Y == 1) && (y_keyvalue_flag == 1))
         unsigned int(nKVQBit) firstKV_Y;
   if (bIsUnaryAAC_Z != 1)
      if((nStartIndex_Z == 1) && (z_keyvalue_flag == 1))
         unsigned int(nKVQBit) firstKV_Z;
   KeyValueMinMax kvMinMax(nKVDigit);
}
```

8.9.5.2.2 Semantics

bPreserveKey indicates if the current mode is key preserving mode or not.

nKVQBit indicates the quantization bit size of the keyValue data.

x_keyvalue_flag, y_keyvalue_flag and z_keyvalue_flag indicate, for three components (x, y, z), if all of the quantized values are the same.

nKVDigit indicates the maximum significant digit of keyValue.

nKVDPCMOrder_X, nKVDPCMOrder_Y and nKVDPCMOrder_Z indicate the order of DPCM used for each keyValue component. The flags are set to 0 if 1st order DPCM is used, and 1 if 2nd order DPCM is used.

bIsUnaryAAC_X, bIsUnaryAAC_Y, bIsUnaryAAC_Z indicate if unary AAC is used for decoding the key value data components, x, y and z, respectively.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

nKVCodingBit_X, nKVCodingBit_Y and nKVCodingBit_Z indicate the actual number of bits used for coding each keyValue component.

nStartIndex_X, nStartIndex_Y and nStartIndex_Z indicate the start index of each key value component to be decoded using entropy decoder.

firstKV_X, firstKV_Y and firstKV_Z indicate the x, y and z of first quantized keyValues.

8.9.5.3 KeyValueMinMax

8.9.5.3.1 Syntax

```
class KeyValueMinMax(int nKeyValueDigit) {
   bit(1) bUse32Float;
   if(bUse32Float == 0) {
      bit(2) nWhichAxis;
      bit(1) bAllSameMantissaDigitFlag;
      if(bAllSameMantissaDigitFlag == 0) {
         unsigned int(4) nMantissaDigit_X;
         unsigned int(4) nMantissaDigit_Y;
         unsigned int(4) nMantissaDigit_Z;
      } else {
         bit(1) bSameKVDigitFlag;
         if(bSameKVDigitFlag == 0)
            unsigned int(4) nMantissaDigit_X;
         else
            nMantissaDigit_X = nKeyValueDigit;
         nMantissaDigit_Y = nMantissaDigit_X;
         nMantissaDigit_Z = nMantissaDigit_X;
      }
      bit(1) bMaxDigitFlag;
      if(bMaxDigitFlag == 1)
         unsigned int(4) nMantissaDigit_M;
      else {
         switch(nWhichAxis){
         case 0:
            nMantissaDigit_M = nMantissaDigit_X;
            break;
         case 1:
            nMantissaDigit_M = nMantissaDigit_Y;
            break;
         case 2:
            nMantissaDigit_M = nMantissaDigit_Z;
            break;
         }
      }
      unsigned int(6) nExponentBits;
      nExponentBits = (int)(log10(nExponentBits)/log10(2)) + 1;
      bit(1) bAllSameExponentSign;
      if(bAllSameExponentSign == 1)
         bit(1) nExponentSign;
      FloatingPointNumber fpnMin_X(nMantissaDigit_X, nExponentBits, bAllSameExponentSign,
nExponentSign);
      FloatingPointNumber fpnMin_Y(nMantissaDigit_Y, nExponentBits, bAllSameExponentSign,
nExponentSign);
      FloatingPointNumber fpnMin_Z(nMantissaDigit_Z, nExponentBits, bAllSameExponentSign,
nExponentSign);
      FloatingPointNumber fpnMax(nMantissaDigit_M, nExponentBits, bAllSameExponentSign,
nExponentSign);
   } else {
      float(32) fMin_X;
      float(32) fMin_Y;
      float(32) fMin_Z;
      float(32) fMax;
   }
```

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
}
```

8.9.5.3.2 Semantics

KeyValueMinMax classs retrieves the minimum and the maximum value, which are used for normalization of keyValue.

bUse32Float indicates if 32-bit float is used or not for storing the minimum and the maximum value. If bUse32Float is 0, then FloatingPointNumber, the floating-point number represented in a decimal system, is used.

nWhichAxis indicates the component that has the maximum range.

nWhichAxis	component
0	x
1	y
2	z

bAllSameMantissaDigitFlag indicates if the mantissa digit of the minimum value of x, y and z are the same.

bSameKVDigitFlag indicates if all of the mantissa digit are the same as nKVDigit.

nMantissaDigit_X, nMantissaDigit_Y and nMantissaDigit_Z indicate the mantissa digit of the minimum value of x, y and z, respectively.

bMaxDigitFlag indicates if the digit of the mantissa of maximum value is different from the digit of the mantissa of minimum values. If so, the digit of the mantissa of maximum range is read from the bitstream.

nMantissaDigit_M indicates the mantissa digit of the maximum value.

nExponentBits indicates the number of bits needed to code the exponent value.

bAllSameExponentSign indicates if the sign of exponents of x, y and z are the same.

nExponentSign indicates the sign of exponents when bAllSameExponentSign is true.

fpnMin_X, fpnMin_Y, fpnMin_Z and fpnMax indicate the floating-point number in decimal system. These values are decoded as described in FloatingPointNumber. Then each decoded value is assigned to fMin_X, fMin_Y, fMin_Z and fMax respectively.

fMin_X, fMin_Y and fMin_Z indicate the minimum values of each axis.

If bUse32Float is set to ‘0’, then fMax indicates the maximum value of the component, which has the maximum range. Otherwise, fMax indicates the maximum range.

fMin_X, fMin_Y, fMin_Z and fMax are used for inverse normalization.

8.9.5.4 FloatingPointNumber

8.9.5.4.1 Syntax

```
class FloatingPointNumber(unsigned int nDigit, unsigned int nExponentBits, unsigned int
bAllSameExponentSign, int nSameExponentSign) {
   if(nDigit != 0) {
      bit(1) nSign;
      int nBits = (int)(log10(10^nDigit - 1)/log10(2)) + 1;
      unsigned int(nBits) nMantissa;
      unsigned int(nExpoentBits) nExponent;
      if(bAllSameExponentSign == 0)
         bit(1) nExponentSign;
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      else
         nExponentSign = nSameExponentSign;
   }
}
```

8.9.5.4.2 Semantics

FloatingpointNumber is used to represent a floating-point number in decimal system. The floating-point number is decoded as follows.

If nDigit is zero, the floating-point number is 0.0.

If nDigit is not zero, the floating-point number is

$$(nSign * nMantissa)/10^{nExponentSign*nExponent}$$

nSign indicates the sign of the following value.

nMantissa indicates the mantissa value of FloatingPointNumber in a decimal system.

nExponent indicates the exponent value of FloatingPointNumber in a decimal system.

nExponentSign indicates the sign of the exponent of FloatingPointNumber in a decimal system.

8.9.5.5 PosIKeyValue

8.9.5.5.1 Syntax

```
class PosIKeyValue (PosIKeyValueHeader kVHeader, int nNumberOfKeyValue) {
   if(kVHeader.x_keyvalue_flag == 1) {
      if(kVHeader.bIsUnaryAAC_X == 1)
         for(i=0; i<nNumberOfKeyValue; i++)
            decodeUnaryAAC(&keyValue_X[i], kVXSignContext, kVXUContext);
      else
         decodeSQAAC(keyValue_X, kVHeader.nKVCodingBit_X, kVXSignContext,
kVXMaxValueContext, kVXFoundContext, kVXNotFoundContext, nNumberOfKeyValue,
kVHeader.nStartIndex_X);
   }
   if(kVHeader.y_keyvalue_flag == 1) {
      if(kVHeader.bIsUnaryAAC_Y == 1) {
         for(i= 0; i<nNumberOfKeyValue; i++)
            decodeUnaryAAC(&keyValue_Y[i], kVYSignContext, kVYUContext);
      else
         decodeSQAAC(keyValue_Y, kVHeader.nKVCodingBit_Y, kVYSignContext,
kVYMaxValueContext, kVYFoundContext, kVYNotFoundContext, nNumberOfKeyValue,
kVHeader.nStartIndex_Y);
   }
   if(kVHeader.z_keyvalue_flag == 1) {
      if(kVHeader.bIsUnaryAAC_Z == 1)
         for(i= 0; i<nNumberOfKeyValue; i++)
            decodeUnaryAAC(&keyValue_Z[i], kVZSignContext, kVZUContext);
      else
         decodeSQAAC(keyValue_Z, kVHeader.nKVCodingBit_Z, kVZSignContext,
kVZMaxValueContext, kVZFoundContext, kVZNotFoundContext, nNumberOfKeyValue,
kVHeader.nStartIndex_Z);
   }
}
```

8.9.5.5.2 Semantics

keyValue_X, keyValue_Y and keyValue_Z indicate the array of each component of keyValue in position interpolator. If bIsUnaryAAC_X is set to 0 and nStartIndex_X is set to 1, then keyValue_X[0] is filled with the value firstKV_X in PosIKeyValueHeader class. Otherwise, keyValue_X[0] is decoded from bitstream using arithmetic decoder. In the same

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

way, keyValue_Y[0] and keyValue_Z[0] are determined. It is arithmetic decoded from the bitstream by the function decodeUnaryAAC or decodeSQAAC (see subclause 8.9.6).

The context model kVXSignContext, kVYSignContext and kVZSignContext are used for entropy decoding the sign of keyValue_X, keyValue_Y and keyValue_Z and these contexts are passed to the function decodeUnaryAAC or decodeSQAAC. MaxValueContext, FoundContext and NotFoundContext are for entropy decoding the absolute value of keyValue (e.g., kVXMaxValueContext, kVXFoundContext and kVXNotFoundContext are used for decoding keyValue_X) and are passed to the function decodeSQAAC. The context model kVXUContext, kVYUContext and kVZUContext are used for decoding keyValue_X, keyValue_Y and keyValue_Z and passed to the function decodeUnaryAAC.

8.9.5.6 Decoding Process

8.9.5.6.1 Overview

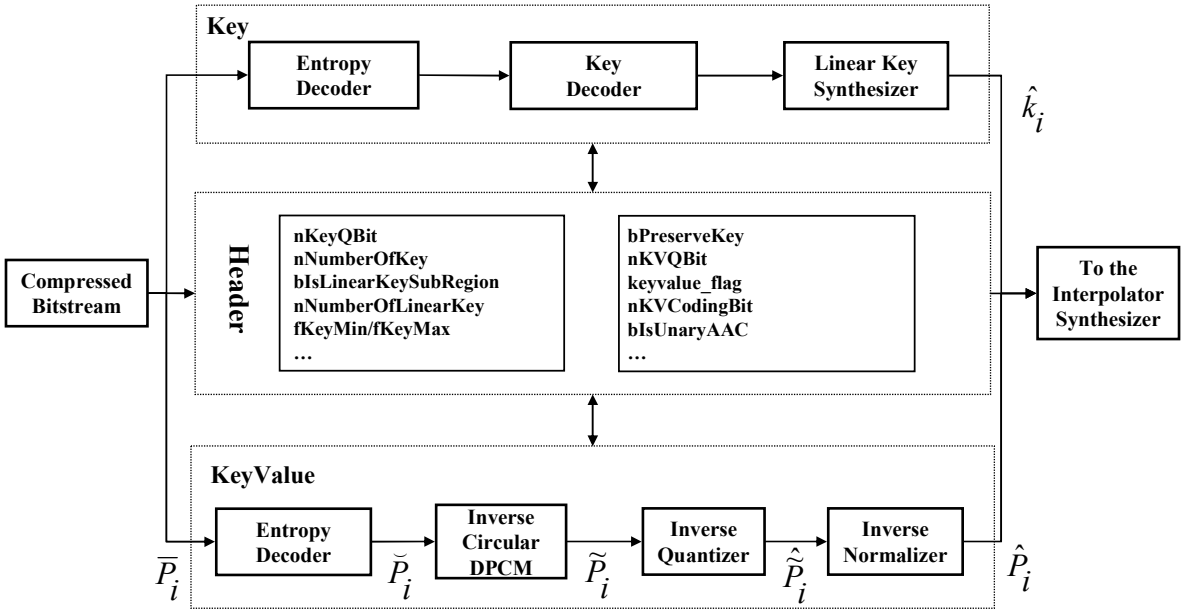


Figure 57 — Decoder structure for compressed position interpolator

The decoder structure of compressed position interpolator is shown in Figure 57. It consists of the decoders for Key, Header and KeyValue. Decoding process of Key data is described in subclause 8.9.2.6. In the following subclauses, the decoding process of KeyValue for position interpolator is described. It is comprised of the following steps.

- Entropy Decoding
- Inverse Circular DPCM
- Inverse Quantization
- Inverse Normalization
- Interpolator synthesizing

8.9.5.6.2 Entropy Decoder

After reading the header information and key data, the keyValue data($\overline{P}_i = (\overline{p}_{i,x}, \overline{p}_{i,y}, \overline{p}_{i,z})$) in the encoded bitstream are passed to the entropy decoder (adaptive arithmetic decoder), if the keyvalue_flag (x_keyvalue_flag, y_keyvalue_flag and z_keyvalue_flag in PosIKeyValueHeader class) is set to 1. Otherwise, if the keyvalue_flag is set to 0, it means that all quantized values of each component of every keyValues have the same values respectively that are represented by

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

fMin_X, fMin_Y, fMin_Z (in KeyValueMinMax class). Therefore, there is no keyValue data in the encoded bitstream that are passed to the entropy decoder.

If $\breve{P}_i = (\breve{p}_{i,x}, \breve{p}_{i,y}, \breve{p}_{i,z}) = (keyValue_X[i], keyValue_Y[i], keyValue_Z[i])$ is defined as the output of the entropy decoder, then it is decoded by the following function.

$$\begin{cases} \hat{P}_{i,x} = fMin_X \\ \quad (i = 0,...,nNumberOfKeyValue - 1, if\ \ x_keyvalue_flag == 0) \\ Entropy_Decoder(\bullet) : \breve{P}_{i,x} = f(\overline{P}_{i,x}) \\ \quad (i = nStartIndex_X,...,nNumberOfKeyValue - 1, if\ \ x_keyvalue_flag == 1) \end{cases}$$

$$\begin{cases} \hat{P}_{i,y} = fMin_Y \\ \quad (i = 0,...,nNumberOfKeyValue - 1, if\ \ y_keyvalue_flag == 0) \\ Entropy_Decoder(\bullet) : \breve{P}_{i,y} = f(\overline{P}_{i,y}) \\ \quad (i = nStartIndex_Y,...,nNumberOfKeyValue - 1, if\ \ y_keyvalue_flag == 1) \end{cases}$$

$$\begin{cases} \hat{P}_{i,z} = fMin_Z \\ \quad (i = 0,...,nNumberOfKeyValue - 1, if\ \ z_keyvalue_flag == 0) \\ Entropy_Decoder(\bullet) : \breve{P}_{i,z} = f(\overline{P}_{i,z}) \\ \quad (i = nStartIndex_Z,...,nNumberOfKeyValue - 1, if\ \ z_keyvalue_flag == 1) \end{cases}$$

The order of bitstream read by entropy decoder is shown in Figure 58.

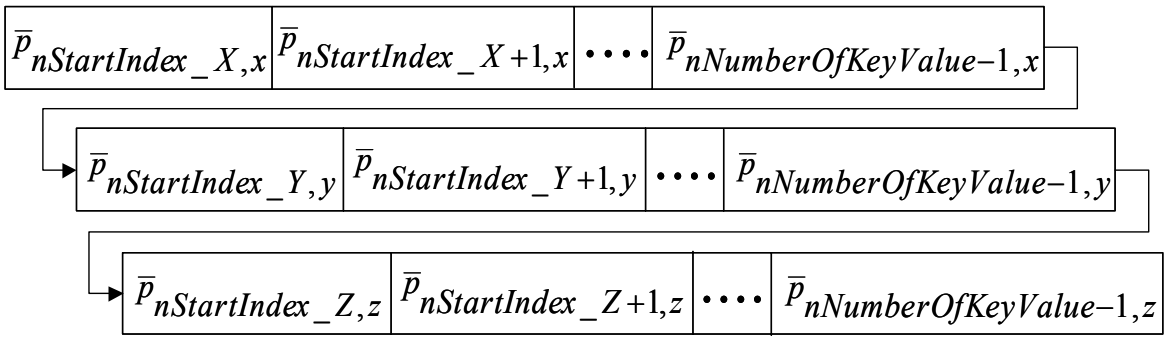


Figure 58 — The decoding order of keyValue data in entropy decoder

The decoder for compressed position interpolator uses two types of entropy decoding method. One is unary AAC and the other is SQ (Successive Quantization) AAC. If the AAC flag for a component (i.e., bIsUnaryAAC_X, bIsUnaryAAC_Y and bIsUnaryAAC_Z) is true, then unary AAC is used. Otherwise, SQ AAC is used. (see subclause 8.9.6)

8.9.5.6.3 Inverse Circular DPCM

The entropy decoding is followed by inverse circular DPCM. Depending on a DPCM flag (nKVDPCMOrder_X, nKVDPCMOrder_Y and nKVDPCMOrder_Z in PosIHeader class), 1st DPCM or 2nd DPCM is performed. Each formula is as follows.

- **Inverse 1st-order circular DPCM function**

© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$ICDPCM_1stOrder(\bullet): \widetilde{P}_{i,x} = f(\breve{P}_{i,x}, \breve{P}_{i-1,x}) \quad (i = nStartIndex_X, ..., nNumberOfKeyValue)$$
$$ICDPCM_1stOrder(\bullet): \widetilde{P}_{i,y} = f(\breve{P}_{i,y}, \breve{P}_{i-1,y}) \quad (i = nStartIndex_Y, ..., nNumberOfKeyValue)$$
$$ICDPCM_1stOrder(\bullet): \widetilde{P}_{i,z} = f(\breve{P}_{i,z}, \breve{P}_{i-1,z}) \quad (i = nStartIndex_Z, ..., nNumberOfKeyValue)$$

The following is the C++ style syntactic description of the function ICDPCM_1stOrder.

```
ICDPCM_1stOrder(int* curIDPCMKeyValue, int deltaKeyValue, int prevKeyValue) {

   int nCircularValue;
   if(deltaKeyValue >= 0)
      nCircularValue = deltaKeyValue – (2^nKeyValueQBits – 1);
   else
      nCircularValue = deltaKeyValue + (2^nKeyValueQBits – 1);
   *curIDPCMKeyValue = deltaKeyValue + prevKeyValue;
   if(*curIDPCMKeyValue < 0)
      * curIDPCMKeyValue = prevKeyValue + nCircualrValue;
   else if(*curIDPCMKeyValue > (2^nKeyValueQBits – 1))
      * curIDPCMKeyValue = prevKeyValue + nCircualrValue;
}
```

- **Inverse 2nd-order circular DPCM function**

$$ICDPCM_2ndOrder(\bullet): \widetilde{P}_{i,x} = f(\breve{P}_{i,x}, \breve{P}_{i-1,x}, \breve{P}_{i-2,x}) \quad (i = nStartIndex_X, ..., nNumberOfKeyValue)$$
$$ICDPCM_2ndOrder(\bullet): \widetilde{P}_{i,y} = f(\breve{P}_{i,y}, \breve{P}_{i-1,y}, \breve{P}_{i-2,y}) \quad (i = nStartIndex_Y, ..., nNumberOfKeyValue)$$
$$ICDPCM_2ndOrder(\bullet): \widetilde{P}_{i,z} = f(\breve{P}_{i,z}, \breve{P}_{i-1,z}, \breve{P}_{i-2,z}) \quad (i = nStartIndex_Z, ..., nNumberOfKeyValue)$$

The 2nd-order circular DPCM does not allow the decoded values to exceed the quantization range

The following is the C++ style syntactic description of the function ICDPCM_2ndOrder

```
ICDPCM_2ndOrder(int* curIDPCMKeyValue, int deltaKeyValue, int prevKeyValue, int
prevPrevKeyValue) {
   int nPredictiveValue, nCircularValue;
   nPredictiveValue = prevKeyValue – prevPrevKeyValue + prevKeyValue;
   if(nPredictiveValue > (2^nKeyValueQBits – 1))
      *curIDPCMKeyValue = (2^nKeyValueQBits – 1) - deltaKeyValue;
   else if(nPredictiveValue < 0)
      *curIDPCMKeyValue = deltaKeyValue;
   else
      *curIDPCMKeyValue = deltaKeyValue + nPredictiveValue;
   if(*curIDPCMKeyValue >= 0)
      nCircularValue = *curIDPCMKeyValue – (nMax + 1);
   else
      nCircularValue = *curIDPCMKeyValue + (nMax + 1);
   if((*curIDPCMKeyValue < 0) || (*curIDPCMKeyValue > (2^nKeyValueQBits – 1)))
      * curIDPCMKeyValue = nCircularValue;
}
```

Both of ICDPCM_1stOrder and ICDPCM_2ndOrder includes inverse circular quantization routine. Inverse circular quantization chooses the value within quantization range.

8.9.5.6.4 Inverse Quantization

The ouput from the inverse circular DPCM will go through inverse quantization and inverse normalization.

Inverse quantizaion is performed as follows.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$\hat{\tilde{P}}_{i,j} = \frac{\tilde{P}_{i,j}}{(2^{nKeyValueQBits} - 1)}$$
$$(i = nStartIndex..nNumberOfKeyValue, j = x, y, z)$$

8.9.5.6.5 Inverse Normalization

The next step is the inverse normalization routine as follows.

If bUse32Float is set to '0', then fMax indicates the maximum value of the component, which has the maximum range. Otherwise, fMax indicates the maximum range.

$$if\ (bUse32Float == 0)$$
$$\begin{cases} Range_{max} = fMax - fMin_X, & if\ nwhichAxis = 0 \\ Range_{max} = fMax - fMin_Y, & if\ nwhichAxis = 1 \\ Range_{max} = fMax - fMin_Z, & if\ nwhichAxis = 2 \end{cases}$$
$$else$$
$$Range_{max} = fMax$$

The process of inverse normalization with $Range_{max}$ is as follows.

$$\hat{P}_{i,x} = \hat{\tilde{P}}_{i,x} * Range_{max} + fMin_X$$
$$\hat{P}_{i,y} = \hat{\tilde{P}}_{i,y} * Range_{max} + fMin_Y$$
$$\hat{P}_{i,z} = \hat{\tilde{P}}_{i,z} * Range_{max} + fMin_Z$$
$$(i = nStartIndex..nNumberOfKeyValue)$$

8.9.5.6.6 Interpolator Synthesizer

In case of key preserving mode, keyFlag described in KeySelectionFlag class is used for determining the existence of each keyValue. The absent keyValues are restored by interpolating the existing neighbor keyValues. Key preserving mode and path preserving mode are explained in detail in subclause 8.9.4.6.1.1.

8.9.6 Adaptive Arithmetic Coding

In this subclause, the adaptive arithmetic coders used in interpolator compression are described, using the C++ style syntactic description. qf_decode() is the function which reads one bit from bitstream and is described in the subclause 7.13.10.2 of ISO/IEC 14496-2: 2004.

8.9.6.1 decodeSignedAAC

This function decodes a signed value from adaptive arithmetic coded bitstream using the contexts for signs and values.

```
void decodeSignedAAC(int *nDecodedValue, int qstep, QState *signContext, QState
*valueContext) {
   int b;
   b = qstep - 2;
   int msb = 0;
   do {
      qf_decode(&msb, &valueContext[b]);
      msb = msb << b;
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      b--;
   } while (msb == 0 && b >=0);
   int sgn = 0;
   int rest = 0;
   if(msb != 0) {
      qf_decode(&sgn, signContext);
      while (b >= 0) {
         int temp = 0;
         qf_decode(&temp, &valueContext[b]);
         rest |= (temp << b);
         b--;
      }
   }
   if(sgn)
      *nDecodedValue = -(msb+rest);
   else
      *nDecodedValue = (msb+rest);
}
```

8.9.6.2 decodeUnsignedAAC

This function decodes a unsigned value from adaptive arithmetic coded bitstream using the context for values.

```
void decodeUnsignedAAC(int *nDecodedValue, int qstep, QState *valueContext) {
   int b;
   b = qstep - 1;
   int msb = 0;
   do {
      qf_decode(&msb, &valueContext[b]);
      msb = msb << b;
      b--;
   } while (msb == 0 && b >= 0);
   int rest = 0;
   if(msb != 0) {
      while (b >= 0) {
         int temp = 0;
         qf_decode(&temp, &valueContext[b]);
         rest |= (temp << b);
         b--;
      }
   }
   *nDecodedValue = (msb + rest);
}
```

8.9.6.3 decodeUnaryAAC

Unary AAC reads 0's from the bitstream until it encounters 1. Then the number of 0's determines the magnitude of the decoded value. After reading 1, the decoder reads one more bit which determines the sign of the decoded value.

```
void decodeUnaryAAC(int* nDecodedValue, QState* signContext, QState* valueContext)
{
   int nBits = -1;
   bit bBit;
   do {
      qf_decode(&bBit, valueContext);
      nBits++;
   } while(bBit == 0);
   if(nBits != 0) {
      qf_decode(&bBit, signContext);
      if(bBit == 0)
         * nDecodedValue = nBits;
      else
         * nDecodedValue = -nBits;
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   }
   else
      * nDecodedValue = 0;
}
```

8.9.6.4 decodeSQAAC

SQ (Successive Quantization) AAC determines the value of each symbol by successively refining the range of quantization. The steps for decoding the values are as follows.

0) Decode the maximum value

1) Decode the sign value of all the symbols

2) The boundary of each symbol is initialized to [0, max].

3) Repeat the following until the range of every symbol is 0 (i.e. the upper bound == the lower bound).

3.1) For each symbol with positive range

3.1.1) Read one bit.

3.1.2) If 0 is read, it means that the symbol is in the lower half of its range. In this case, the value of the upper bound is changed to middle value – 1

3.1.3) If 1 is read, it means that the symbol is in the upper half of its range. In this case, the value of the lower bound is changed to the middle value.

3.2) If there is only one symbol with maximum value as the upper bound, it means that the corresponding symbol is the maximum value. In this case, the value of lower bound is changed to the maximum value. This is performed only once during the whole SQ process.

There are 4 contexts used in SQ AAC. maxValueContext is used for the maximum value and the signContext is used for the sign of the symbols. foundContext and notFoundContext is used for symbols. For each symbol, notFoundContext is used until the first '1' is read. And after that, foundContext is used.

```
void decodeSQAAC(int* anDecodedValues, int qstep, QState* signContext, QState*
maxValueContext, QState* foundContext, QState* notFoundContext, int keynum, int start) {
   int range_mid[keynum], range_min[keynum], range_max[keynum], sign[keynum];
   bool found[keynum];
   int max_val = 0;
   int max_count = 1;
   bool all_level_done = false;
   int max_index = -1;
   bit bBit;
   for (int j=qstep; j>=0; j--) {
      qf_decode(&bBit, &maxValueContext);
      max_val = (max_val << 1) + bBit;
   }
   for(int i=start; i<keynum; i++) {
      range_min[i] = 0;
      range_max[i] = max_val;
      found[i] = false;
      qf_decode(&bBit, &signContext);
      if (bBit)
         sign[i] = -1;
      else
         sign[i] = 1;
   }
   while (!all_level_done) {
      all_level_done = true;
      max_index = -1;
      for(i=start; i<keynum; i++) {
         if (range_max[i] != range_min[i]) {
            all_level_done = false;
            range_mid[i] = (int)((range_max[i] - range_min[i])/2) + 1 + range_min[i];
            if (found[i])
               qf_decode(&bBit, &foundContext);
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            else
               qf_decode(&bBit, &notFoundContext);
            if (bit) {
               if (max_count == 1 && range_max[i] == max_val) {
                  if (max_index == -1)
                     max_index = i;
                  else
                     max_index = keynum;
               }
               range_min[i] = range_mid[i];
               found[i] = true;
            }
            else
               range_max[i] = range_mid[i]-1;
         }
      }
      if (max_index >= 0 && max_index < keynum) {
         range_min[max_index] = max_val;
         max_count = 0;
      }
   }
   for(i=start; i<keynum; i++)
      anDecodedValues[i] = range_max[i] * sign[i];
}
```

8.9.6.5 decodeSignedQuasiAAC

This function decodes a signed value from adaptive arithmetic coded bitstream using the contexts for signs and values. The difference from decodeSignedAAC is using zero context for data following the sign bit.

```
void decodeSignedQuasiAAC(int *nDecodedValue, int qstep, QState *signContext, QState
*valueContext)
{
   int b = qstep - 2;
   int msb = 0;
   do {
      qf_decode(&msb, &valueContext[b]);
      msb = msb << b;
      b--;
   } while (msb == 0 && b >= 0);
   int sgn = 0;
   int rest = 0;
   if(msb != 0) {
      qf_decode(&sgn, signContext);
      while (b >= 0) {
         int temp = 0;
         qf_decode(&temp, zeroContext);
         rest |= (temp << b);
         b--;
      }
   }
   if(sgn)
      *nDecodedValue = -(msb+rest);
   else
      *nDecodedValue = (msb+rest);
}
```

8.10 Definition of bodySceneGraph nodes

8.10.1 Introduction

This Annex includes the normative definitions of the nodes used in the **bodySceneGraph** field of the **BDP** node (see 7.2.2.20).

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.10.2 Detailed Semantics

The VRML working group on Humanoid Animation (H-Anim) is working on a standard specification of bodies. The bodySceneGraph syntax is strongly based on ISO/IEC 14772-1:1997/Amd.1.

The H-Anim specification contains 3 types of Nodes, among other nodes: `Joint` node describes the skeleton hierarchy of the body, `Segment` node describes the surface information of the body, `HumanoidInfo` node includes information about the model.

8.10.3 Overview

The human body consists of a number of segments (such as the forearm, hand and foot) which are connected to each other by joints (such as the elbow, wrist and ankle). In order for a decoder to animate a humanoid, it needs to obtain access to the joints and alter the joint angles.

Each segment of the body will typically be defined by children nodes of type **IndexedFaceSet**, and an application may need to alter the locations of the vertices in that mesh. The application may also need to obtain information about which vertices should be treated as a group for the purpose of deformation.

The **bodySceneGraph** field of a **BDP** node contains a set of `Joint` nodes that are arranged to form a hierarchy. Each `Joint` node can contain other Joint nodes, and may also contain a `Segment` node which describes the body part associated with that joint. Each Segment can also have a number of `Site` nodes, which define locations relative to the segment. `Sites nodes` can be used for attaching clothing and jewelry, and can be used as end-effectors for inverse kinematics applications. They can also be used to define eyepoints and viewpoint locations.

Each `Segment` node can have a number of `Displacer` nodes, which specify which vertices within the segment correspond to particular feature or configuration of vertices. The **bodySceneGraph** node also contains a single Humanoid node which stores human-readable data about the humanoid such as author and copyright information. That node also stores references to all the `Joint, Segment` and `Site` nodes, and serves as a "wrapper" for the humanoid. In addition, it provides a top-level Transform for positioning the humanoid in its environment.

Keyframe animation sequences can be stored in the same file, with the outputs of various interpolator nodes being ROUTEd to the joints of the body. Alternatively, the file may include **Script** nodes which access the joints directly. In addition, applications can obtain references to the individual joints and segments from the `Humanoid` node. Such applications will typically animate the humanoid by setting the joint rotations through BAPs.

8.10.4 The Nodes

In order to simplify the creation of humanoids, several new node types are introduced. Each node is defined by a PROTO. The basic implementation of all the nodes is very straightforward, yet each provides enough flexibility to allow more advanced techniques to be used.

8.10.4.1 The Joint Node

Each joint in the body is represented by a Joint node. The implementation for a `Joint` is a **Transform** node, which is used to define the relationship of each body segment to its immediate parent.

The `Joint` node is also used to store other joint-specific information. In particular, a joint name is provided so that applications can identify each `Joint` node at runtime.

In addition, the `Joint` node may contain hints for inverse-kinematics systems that wish to control the H-Anim figure. These hints include the upper and lower joint limits, the orientation of the joint limits, and a stiffness/resistance value. Note that these limits are not enforced by any mechanism within the scene graph of the humanoid, and are provided for information purposes only. Use of this information and enforcement of the joint limits is up to the application.

The `Joint` PROTO looks like follows:

```
PROTO Joint [
    exposedField     SFString     name               ""
    exposedField     SFVec3f      center             0 0 0
    exposedField     SFRotation   rotation           0 0 1 0
    exposedField     SFVec3f      scale              1 1 1
    exposedField     SFRotation   scaleOrientation   0 0 1 0
    exposedField     SFVec3f      translation        0 0 0
    exposedField     MFFloat      ulimit             []
    exposedField     MFFloat      llimit             []
    exposedField     SFRotation   limitOrientation   0 0 1 0
    exposedField     MFFloat      stiffness          [ 1 1 1 ]
    exposedField     MFNode       children           []
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
]
```

NOTE - Most of the fields correspond to those of the Transform node. This is because the typical implementation of the `Joint` PROTO will be:

```
{
    Transform {
        translation IS translation
        rotation IS rotation
        scale IS scale
        scaleOrientation IS scaleOrientation
        center IS center
        children IS children
    }
}
```

The `center` exposedField gives the position of the Joint's center of rotation, relative to the root of the overall humanoid body description. Note that the `center` field is not intended to receive events. The locations of the joint centers are available by reading the center fields of the `Joint nodes`.

Since the locations of the joint centers are all in the same coordinate frame, the length of any segment can be determined by simply subtracting the locations of the joint centers. The exception will be segments at the ends of the fingers and toes, for which the `Site` locations within the `Segment` must be used (see the description of `Sites` below for details).

The `ulimit` and `llimit` fields of the `Joint PROTO` specify the upper and lower joint rotation limits. Both fields are three-element MFFloats containing separate values for the X, Y and Z rotation limits. The ulimit field stores the upper (i.e. maximum) values for rotation around the X, Y and Z axes. The llimit field stores the lower (i.e. minimum) values for rotation around those axes. Note that the default values for each of these fields is [], which means that the joint is assumed to be unconstrained.

The `limitOrientation` exposedField gives the orientation of the coordinate frame in which the ulimit and llimit values are to be interpreted. The limitOrientation describes the orientation of a local coordinate frame, relative to the `Joint` center position described by the `center` exposedField.

The `stiffness` exposedField, if present, contains values ranging between 0.0 and 1.0 which give the inverse kinematics system hints about the "willingness" of a joint to move a particular degree of freedom. For example, a `Joint` node's stiffness can be used in an arm joint chain to give preference to moving the left wrist and left elbow over moving the left shoulder, or it can be used within a single `Joint` node with multiple degrees of freedom to give preference to individual degrees of freedom. The larger the stiffness value, the more the joint will resist movement.

Each `Joint` should have a *DEF* name that matches the name field for that `Joint`, but with a distinguishing prefix in front of it. Only a single humanoid is contained within a **Body** node, the prefix should be "`hanim_...`" (for Humanoid Animation). For example, the left shoulder would have a DEF name of "`hanim_l_shoulder`".

The DEF name is used for static routing, which would typically connect the BAPs to segments, and define segment names for **bodyDefTables**. In addition, optionally, it may be used for connect **OrientationInterpolators** in the humanoid file to the `joints`.

It will occasionally be useful for the person creating a humanoid to be able to add additional joints to the body. The body remains humanoid in form, and is still generally expected to have the basic joints described later in this document. However, they may be thought of as a minimal set to which extensions may be added (such as a prehensile tail). See the section on Non-standard `Joints` and `Segments`. If necessary, some of the joints (such as the many vertebrae) may be omitted.

Each body segment is stored in a `Segment` node. The `Segment` node will typically be implemented as a **Group** node containing one or more **Shape**s or perhaps **Transform** nodes that position the body part within its coordinate system (see Annex, for details). The use of LOD nodes is recommended if the geometry of the `Segment` is complex.

```
PROTO Segment [
    exposedField    SFString    name               ""
    exposedField    SFVec3f     centerOfMass       0 0 0
    exposedField    SFVec3f     momentsOfInertia   1 1 1
    exposedField    SFFloat     mass               0
    exposedField    MFNode      children           [ ]
    exposedField    SFNode      coord              NULL
    exposedField    MFNode      displacers         [ ]
    eventIn         MFNode      addChildren
    eventIn         MFNode      removeChildren
]
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

This will typically be implemented as follows:

```
{
    Group {
        children IS children
        addChildren IS addChildren
        removeChildren IS removeChildren
    }
}
```

The fields except `name` are optional.

The `mass` is the total mass of the segment, and the `centerOfMass` is the location within the segment of its center of mass.

If **bodyDefTable**s need to be used, the `Segment` node contains one **IndexedFaceSet** child that shall be used for these tables. The indices of vertices in the **IndexedFaceSet** node should correspond to the indices in **bodyDefTable** node.

8.10.4.2 The Humanoid Node

The `Humanoid` node is used to store human-readable data such as author and copyright information, as well as to store references to the `joint`s, `segment`s and `view`s and to serve as a container for the entire humanoid. It also provides a convenient way of moving the humanoid through its environment.

```
PROTO Humanoid [
    exposedField    SFString   name                ""
    exposedField    MFString   info                [ ]
    exposedField    SFString   version             "1.1"
    exposedField    MFNode     joints              [ ]
    exposedField    MFNode     segments            [ ]
    exposedField    MFNode     sites               [ ]
    exposedField    MFNode     viewpoints          [ ]
    exposedField    MFNode     humanoidBody        [ ]
    exposedField    SFVec3f    center              0 0 0
    exposedField    SFRotation rotation            0 0 1 0
    exposedField    SFVec3f    scale               1 1 1
    exposedField    SFRotation scaleOrientation    0 0 1 0
    exposedField    SFVec3f    translation         0 0 0
]
```

The `Humanoid` node is typically implemented as follows:

```
{
  Transform {
    center           IS center
    rotation         IS rotation
    scale            IS scale
    scaleOrientation IS scaleOrientation
    translation      IS translation
    children [
      Group {
        children IS viewpoints
      }
      Group {
        children IS humanoidBody
      }
    ]
  }
}
```

The `Humanoid` node can be used to position the humanoid in space. Note that the `HumanoidRoot Joint` is typically used to handle animations within the local coordinate system of the humanoid, such as jumping or walking. For example, while walking, the overall movement of the body (such as a swagger) would affect the `HumanoidRoot Joint`, while the average linear velocity through the scene would affect the `Humanoid` node.

The `humanoidBody` field contains the `HumanoidRoot` node. The version field stores the version of this specification that the `Humanoid` file conforms to. Value of '1.1' is excepted.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The info field consists of an array of strings, each of which is of the form "tag=value". The following tags are defined:

```
authorName
authorEmail
copyright
creationDate
usageRestrictions
humanoidVersion
age
gender (typically "male" or "female")
height
weight
```

Additional tag=value pairs can be included as needed.

The `HumanoidVersion` tag refers to the version of the humanoid being used, in order to track revisions to the data. It is not the same as the version field of the `Humanoid` node, which refers to the version of the H-Anim specification which was used when building the humanoid.

The joints field contains references (i.e. USEs) of each of the `Joint` nodes in the body. Each of the referenced joints should be a `Joint` node. The order in which they are listed is irrelevant, since the names of the joints are stored in the joints themselves. Similarly, the segments field contains references to each of the `Segment` nodes of the body, the viewpoints field contains references to the **Viewpoint** nodes in the file, and the sites field contains references to the `Site` nodes in the file.

8.10.4.3 Modeling the Humanoid

Humanoid should be modeled in a standing position, facing in the +Z direction with +Y up and +X to the humanoid's left. The origin (0, 0, 0) should be located at ground level, between the humanoid's feet.

The feet should be flat on the ground, spaced apart about the same distance as the width of the hips. The bottom of the feet should be at Y=0. The arms should be straight and parallel to the sides of the body with the palms of the hands facing inwards towards the thighs. The hands should be flat, with the axes of joints "1" through "3" of the fingers being parallel to the Y axis and the axis of the thumb being angled up at 45 degrees towards the +Z direction. Note that the coordinate system for each joint in the thumb is still oriented to align with that of the overall humanoid.

Movement of the "0" joints of the fingers is typically quite limited, and the rigidity of those articulations varies from finger to finger. Further details about the placement, orientation and movement of the "0" joints can be obtained from any anatomy reference text.

The humanoid should be built with actual human size ranges in mind. All dimensions are in meters. A typical human is roughly 1.75 meters tall.

The default position of the humanoid is defined in ISO/IEC 14496-2:2004.

In this position, all the joint angles should be zero. In other words, all the rotation fields in all the `Joint` nodes should be left at their default value of (0 0 1 0). In addition, the translation fields should be left at their default value of (0 0 0) and the scale factors should be left at their default value of (1 1 1). The only field which should have a non-default value is center, which is used to specify the point around which the joint (and its attached children and body segment if any) will rotate. Sending the default values for translation, rotation and scaling to all the `Joints` in the body must return the body to the neutral position described above.The center field of each joint should be placed so that the joints rotate in the same way that they would on a real human body.

It is suggested, but not required, that all of the body segments should be built in place. That is, they should require no translation, rotation, or scaling to be connected with their neighbors. For example, the hand should be built so that it's in the correct position in relation to the forearm. The forearm should be built so that it's in the correct position in relation to the upper arm, and so on. All the body's coordinates will share a common origin, which is that of the humanoid itself. If this proves difficult for an authoring tool to implement, it is acceptable to use a Transform node inside each `Segment`, or even several **`Transforms`**, in order to position the geometry for that segment correctly.

Note that the coordinate system for each `Joint` is oriented to align with that of the overall humanoid.

8.10.4.3.1 The Joint Hierarchy

The body is typically built as a series of nested `Joints`, each of which may have a `Segment` associated with it. For example:

```
...
DEF hanim_l_shoulder Joint {  name "l_shoulder"
  center 0.167 1.36 -0.0518
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  children    [
    DEF hanim_l_elbow Joint {  name "l_elbow"
      center 0.196 1.07 -0.0518
      children        [
        DEF hanim_l_wrist Joint {  name "l_wrist"
          center 0.213 0.811 -0.0338
          children    [
            DEF hanim_l_hand Segment {  name "l_hand"
              ...
            }
          ]
        }
        DEF hanim_l_forearm Segment {  name "l_forearm"
          ...
        }
      ]
    }
    DEF hanim_l_upperArm Segment {  name "l_upperArm"
      ...
    }
  ]
}
```

8.10.4.3.2 The Body

The names of the `Joint` nodes for the body are listed in the following list:

l_hip, l_knee, l_ankle, l_subtalar, l_midtarsal, l_metatarsal
r_hip, r_knee, r_ankle, r_subtalar, r_midtarsal, r_metatarsal
vl5, vl4, vl3, vl2, vl1,
vt12, vt11, vt10, vt9, vt8, vt7, vt6, vt5, vt4, vt3, vt2, vt1
vc7, vc6, vc5, vc4, vc3, vc2, vc1
l_sternoclavicular, l_acromioclavicular, l_shoulder, l_elbow, l_wrist
r_sternoclavicular, r_acromioclavicular, r_shoulder, r_elbow, r_wrist
HumanoidRoot, sacroiliac (pelvis), skullbase

The vl5 and sacroiliac `Joints` are children of the `HumanoidRoot`. The `HumanoidRoot` is stored in the `humanoidBody` field of the `Humanoid` node, but all other `Joint`s are descended from either vl5 or sacroiliac. If those `Joint`s are missing, lower-level `Joint`s can be children of the `HumanoidRoot`.

8.10.4.3.3 The Hands

The hands `Joint` nodes, if present, should use the following naming convention:

l_pinky0, l_pinky1, l_pinky2, l_pinky3,
l_ring0, l_ring1, l_ring2, l_ring3
l_middle0, l_middle1, l_middle2, l_middle3
l_index0, l_index1, l_index2, l_index3
l_thumb1, l_thumb2, l_thumb3
r_pinky0, r_pinky1, r_pinky2, r_pinky3
r_ring0, r_ring1, r_ring2, r_ring3
r_middle0, r_middle1, r_middle2, r_middle3
r_index0, r_index1, r_index2, r_index3
r_thumb1, r_thumb2, r_thumb3

8.10.4.3.4 Hierarchy

The complete hierarchy is as follows, with the segment names listed beside the `Joint`s to which they're attached:

```
HumanoidRoot : sacrum
  sacroiliac : pelvis
  |   l_hip : l_thigh
  |     l_knee : l_calf
  |       l_ankle : l_hindfoot
  |         l_subtalar : l_midproximal
  |           l_midtarsal : l_middistal
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
|             l_metatarsal : l_forefoot
|   r_hip : r_thigh
|     r_knee : r_calf
|       r_ankle : r_hindfoot
|         r_subtalar :  r_midproximal
|           r_midtarsal : r_middistal
|             r_metatarsal : r_forefoot
vl5  : l5
  vl4 : l4
    vl3  : l3
      vl2 : l2
        vl1  : l1
          vt12 : t12
            vt11 : t11
              vt10  : t10
                vt9 : t9
                  vt8 : t8
                    vt7 : t7
                      vt6  : t6
                        vt5 : t5
                          vt4 : t4
                            vt3 : t3
                              vt2 : t2
                                vt1  : t1
                                  vc7 : c7
                                  | vc6 : c6
                                  |   vc5 : c5
                                  |     vc4  : c4
                                  |       vc3 : c3
                                  |         vc2  : c2
                                  |           vc1 : c1
                                  |             skullbase : skull
                                  |               l_eyelid_joint : l_eyelid
                                  |               r_eyelid_joint : r_eyelid
                                  |               l_eyeball_joint : l_eyeball
                                  |               r_eyeball_joint : r_eyeball
                                  |               l_eyebrow_joint : l_eyebrow
                                  |               r_eyebrow_joint : r_eyebrow
                                  |               temporomandibular : jaw
                                  l_sternoclavicular : l_clavicle
                                  | l_acromioclavicular : l_scapula
                                  |   l_shoulder : l_upperarm
                                  |     l_elbow : l_forearm
                                  |       l_wrist : l_hand
                                  |         l_thumb1 : l_thumb_metacarpal
                                  |           l_thumb2 : l_thumb_proximal
                                  |               l_thumb3 : l_thumb_distal
                                  |         l_index0 : l_index_metacarpal
                                  |           l_index1 : l_index_proximal
                                  |             l_index2 : l_index_middle
                                  |                 l_index3 : l_index_distal
                                  |         l_middle0 : l_middle_metacarpal
                                  |           l_middle1 : l_middle_proximal
                                  |             l_middle2 : l_middle_middle
                                  |               l_middle3 : l_middle_distal
                                  |         l_ring0 : l_ring_metacarpal
                                  |           l_ring1 : l_ring_proximal
                                  |             l_ring2 l_ring_middle
                                  |               l_ring3 : l_ring_distal
                                  |         l_pinky0 : l_pinky_metacarpal
                                  |           l_pinky1 : l_pinky_proximal
                                  |             l_pinky2 : l_pinky_middle
                                  |               l_pinky3 : l_pinky_distal
                                  r_sternoclavicular : r_clavicle
                                    r_acromioclavicular : r_scapula
                                      r_shoulder : r_upperarm
                                        r_elbow : r_forearm
                                          r_wrist : r_hand
                                            r_thumb1 : r_thumb_metacarpal
                                              r_thumb2 : r_thumb_proximal
                                                r_thumb3 : r_thumb_distal
                                            r_index0 : r_index_metacarpal
                                              r_index1 : r_index_proximal
                                                r_index2 : r_index_middle
                                                  r_index3 : r_index_distal
                                            r_middle0 : r_middle_metacarpal
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  r_middle1 : r_middle_proximal
    r_middle2 : r_middle_middle
      r_middle3 : r_middle_distal
r_ring0 : r_ring_metacarpal
  r_ring1 : r_ring_proximal
    r_ring2 : r_ring_middle
      r_ring3 : r_ring_distal
r_pinky0 : r_pinky_metacarpal
  r_pinky1 : r_pinky_proximal
    r_pinky2 : r_pinky_middle
      r_pinky3 : r_pinky_distal
```

Depending on your fonts, the number '1' and the letter 'l' may look similar. This is particularly true for the lumbar vertebrae and their corresponding joints (e.g. vl5 and l5). The letter 'l' is for Lumbar, the letter 't' is for Thorasic, and the letter 'c' is for Cervical.

The term "proximal" means "the nearer" segment, and "distal" means "the farther" segment.

Both the sacroiliac and the `vl5` vertebrae are top-level `Joint`s, and are stored in the `bodyDefinition` field of the `Humanoid` node.

The l_`sternoclavicular` and r_`sternoclavicular` `Joint`s are children of `vt1`, and siblings of `vc7`.

The skullbase `Joint` is technically the "atlanto-occipital" `Joint`.

The left and right metatarsals are technically the left and right "tarsometatarsal" joints.

A `Joint` node may contain 1-3 BAPs. The following table presents the joints, and associated BAPs and segment names.

Table 92 — BAPs in the Joint node

JOINT NODE	ASSOCIATED BAPs	ATTACHED SEGMENT
sacroiliac	sacroiliac_tilt, sacroiliac_torsion, sacroiliac_roll	Pelvis
l_hip	l_hip_flexion, l_hip_abduct, l_hip_twisting	l_thigh
r_hip	r_hip_flexion, r_hip_abduct, r_hip_twisting	r_thigh
l_knee	l_knee_flexion, l_knee_twisting	l_calf
r_knee	r_knee_flexion, r_knee_twisting	r_calf
l_ankle	l_ankle_flexion, l_ankle_twisting	l_hindfoot
r_ankle	r_ankle_flexion, r_ankle_twisting	r_hindfoot
l_subtalar	l_subtalar_flexion	l_midproximal
r_subtalar	r_subtalar_flexion	r_midproximal
l_midtarsal	l_midtarsal_flexion	l_middistal
r_midtarsal	r_midtarsal_flexion	r_middistal
l_metatarsal	l_metatarsal_flexion	l_forefoot
r_metatarsal	r_metatarsal_flexion	r_forefoot
vl5	vl5roll, vl5torsion, vl5tilt	l5
vl4	vl4roll, vl4torsion, vl4tilt	l4
vl3	vl3roll, vl3torsion, vl3tilt	l3
vl2	vl2roll, vl2torsion, vl2tilt	l2
vl1	vl1roll, vl1torsion, vl1tilt	l1
vt12	vt12roll, vt12torsion, vt12tilt	t12
vt11	vt11roll, vt11torsion, vt11tilt	t11
vt10	vt10roll, vt10torsion, vt10tilt	t10
vt9	vt9roll, vt9torsion, vt9tilt	t9
vt8	vt8roll, vt8torsion, vt8tilt	t8
vt7	vt7roll, vt7torsion, vt7tilt	t7
vt6	vt6roll, vt6torsion, vt6tilt	t6
vt5	vt5roll, vt5torsion, vt5tilt	t5
vt4	vt4roll, vt4torsion, vt4tilt	t4
vt3	vt3roll, vt3torsion, vt3tilt	t3
vt2	vt2roll, vt2torsion, vt2tilt	t2
vt1	vt1roll, vt1torsion, vt1tilt	t1
vc7	vc7roll, vc7torsion, vc7tilt	c7
vc6	vc6roll, vc6torsion, vc6tilt	c6
vc5	vc5roll, vc5torsion, vc5tilt	c5
vc4	vc4roll, vc4torsion, vc4tilt	c4
vc3	vc3roll, vc3torsion, vc3tilt	c3
vc2	vc2roll, vc2torsion, vc2tilt	c2
vc1	vc1roll, vc1torsion, vc1tilt	c1
Skullbase	skullbase_roll, skullbase_torsion, skullbase_tilt	skull
l_sternoclavicular	l_sternoclavicular_abduct, l_sternoclavicular_rotate	l_clavicle
l_acromioclavicular	l_acromioclavicular_abduct, l_acromioclavicular_rotate	l_scapula

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

l_shoulder	l_shoulder_flexion, l_shoulder_abduct,l_shoulder_twisting	l_upperarm
l_elbow	l_elbow_flexion, l_elbow_twisting	l_forearm
r_sternoclavicular	r_sternoclavicular_abduct, r_sternoclavicular_rotate	r_clavicle
r_acromioclavicular	r_acromioclavicular_abduct, r_acromioclavicular_rotate	r_scapula
r_shoulder	r_shoulder_flexion, r_shoulder_abduct, r_shoulder_twisting	r_upperarm
r_elbow	r_elbow_flexion, r_elbow_twisting	r_forearm
r_wrist	r_wrist_flexion, r_wrist_pivot, r_wrist_twisting	r_wrist
r_thumb1	r_thumb1_flexion, r_thumb1_pivot, r_thumb1_twisting	r_thumb_metacarpal
r_thumb2	r_thumb2_flexion	r_thumb_proximal
r_thumb3	r_thumb3_flexion	r_thumb_distal
r_index0	r_index0_flexion	r_index_metacarpal
r_index1	r_index1_flexion, r_index1_pivot, r_index1_twisting	r_index_proximal
r_index2	r_index2_flexion	r_index_middle
r_index3	r_index3_flexion	r_index_distal
r_middle0	r_middle0_flexion	r_middle_metacarpal
r_middle1	r_middle1_flexion, r_middle1_pivot, r_middle1_twisting	r_middle_proximal
r_middle2	r_middle2_flexion	r_middle_middle
r_middle3	r_middle3_flexion	r_middle_distal
r_ring0	r_ring0_flexion	r_ring_metacarpal
r_ring1	r_ring1_flexion, r_ring1_pivot, r_ring1_twisting	r_ring_proximal
r_ring2	r_ring2_flexion	r_ring_middle
r_ring3	r_ring3_flexion	r_ring_distal
r_pinky0	r_pinky0_flexion	r_pinky_metacarpal
r_pinky1	r_pinky1_flexion, r_pinky1_pivot, r_pinky1_twisting	r_pinky_proximal
r_pinky2	r_pinky2_flexion	r_pinky_middle
r_pinky3	r_pinky3_flexion	r_pinky_distal
l_wrist	l_wrist_flexion, l_wrist_pivot, l_wrist_twisting	l_wrist
l_thumb1	l_thumb1_flexion, l_thumb1_pivot, l_thumb1_twisting	l_thumb_metacarpal
l_thumb2	l_thumb2_flexion	l_thumb_proximal
l_thumb3	l_thumb3_flexion	l_thumb_distal
l_index0	l_index0_flexion	l_index_metacarpal
l_index1	l_index1_flexion, l_index1_pivot, l_index1_twisting	l_index_proximal
l_index2	l_index2_flexion	l_index_middle
l_index3	l_index3_flexion	l_index_distal
l_middle0	l_middle0_flexion	l_middle_metacarpal
l_middle1	l_middle1_flexion, l_middle1_pivot, l_middle1_twisting	l_middle_proximal
l_middle2	l_middle2_flexion	l_middle_middle
l_middle3	l_middle3_flexion	l_middle_distal
l_ring0	l_ring0_flexion	l_ring_metacarpal
l_ring1	l_ring1_flexion, l_ring1_pivot, l_ring1_twisting	l_ring_proximal
l_ring2	l_ring2_flexion	l_ring_middle
l_ring3	l_ring3_flexion	l_ring_distal
l_pinky0	l_pinky0_flexion	l_pinky_metacarpal
l_pinky1	l_pinky1_abduct, l_pinky1_flexion	l_pinky_proximal
l_pinky2	l_pinky2_flexion	l_pinky_middle
l_pinky3	l_pinky3_flexion	l_pinky_distal

Note that the body can be defined by a subset of `Joint` nodes.

Many `Joint`s may be omitted, such as most of the vertebrae, the midtarsal, and the acromioclavicular. The spinal `Joint`s that belong to first spine groups are the ones that should be given priority if a full spine is not implemented.

Note that VRML H-Anim syntax permits having multiple humanoids in the same file. However, for the files used for BDPs, it is required that the **BodySceneGraph** node contains only one humanoid.

8.10.4.3.5 Other Nodes

Other nodes, such as non-standard joints, viewpoint nodes, displacement nodes, can be defined. These nodes are ignored for the purposes of body animation from FBA elementary stream, but could be updated using BIFS stream.

8.11 Adaptive Arithmetic Decoder for BIFS-Anim

The following procedures, in C code, describe the adaptive arithmetic decoder used in a BIFS-Anim session. The model is specified through the array `int* cumul_freq[ ]`. The decoded symbol is returned through its index in the model.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

First, the following integers are defined :

```
static long bottom=0, q1=2^14, q2=2^15, q3=3*2^14, top=2^16-1;
```

The decoder is initialized to start decoding an arithmetic coded bitstream by calling the following procedure.

```
static long low, high, code_value, bit, length, sacindex, cum, zerorun=0;

void decoder_reset( )
{
   int i;
   zerorun = 0;        /* clear consecutive zero's counter */
   code_value = 0;
   low = 0;
   high = top;
   for (i = 1;   i <= 16;   i++) { //16 bits are read ahead
      bit_out_psc_layer();
      code_value = 2 * code_value + bit;
   }
   used_bits = 0;
}
```

In the BIFS-Anim decoding process, a symbol is decoded using a model specified through the array `cumul_freq[]` and by calling the following procedure.

```
static long low, high, code_value, bit, length, sacindex, cum, zerorun=0;

int aa_decode(int cumul_freq[ ])
{
   length = high - low + 1;
   cum = (-1 + (code_value - low + 1) * cumul_freq[0]) / length;
   for (sacindex = 1; cumul_freq[sacindex] > cum; sacindex++);
   high = low - 1 + (length * cumul_freq[sacindex-1]) / cumul_freq[0];
   low += (length * cumul_freq[sacindex]) / cumul_freq[0];

   for ( ; ; ) {
      if (high < q2) ;
      else if (low >= q2) {
         code_value -= q2;
         low -= q2;
         high -= q2;
      }
      else if (low >= q1 && high < q3) {
         code_value -= q1;
         low -= q1;
         high -= q1;
      }
      else {
         break;
      }
      low *= 2;
      high = 2*high + 1;
      bit_out_psc_layer();
      code_value = 2*code_value + bit;
      used_bits++;
   }
   return (sacindex-1);
}

void bit_out_psc_layer()
{
   bit = getbits(1);
   if (zerorun > 22) {
      if (!bit) {
         // Error condition… long zero runs shouldn't occur
      } else {
        bit = getbits(1); // removed startCode prevsition bit
        used_bits++;
        zerorun = !bit;   // if 0, start counting again at zerorun = 1
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      }
   else { // not close to hitting a fake startCode
      if (!bit) {
        ++zerorun;
      } else {
        zerorun = 0;
      }
   }
}
```

The model is specified in the array `cumul_freq[ ]`. It is reset with the following procedure. The value of nbBits shall always be 14 or less.

```
void model_reset(int nbBits)
{
   int nbValues = (1<<nbBits)+1;
   int* cumul_freq = (int*) malloc(sizeof(int)*nbValues);
   int i;
   for (i=1;i<=nbValues;i++) {
      cumul_freq[i] = nbValues-i;
}
```

The model is updated when the value `symbol` is read with the following procedure.

```
void update_model(int cumul_freq[ ], int symbol) {
   if (cumul_freq[0] == q1) { //The model is rescaled to avoid overflow
      int cum = 0;
      for(int i=nb_of_symbols-1; i>=0; i--) {
         cum += (cumul_freq[i]-cumul_freq[i+1]+1)/2;
         cumul_freq[i] = cum;
      }
      cumul_freq[nb_of_symbols] = 0;
   }

   while(symbol>0)
      cumul_freq[symbol--] ++;

}
```

Note: The `getbits()` routine may hit the end of the input buffer when `bit_out_psc_layer()` is called, because this routine reads several bits ahead. This condition should be ignored during the decoding of input streams.

Because the Arithmetic encoder reads a few bits ahead, it may read bits that are not part of the BIFS-Anim stream. The following procedure is used to rewind the bitstream the appropriate amount.

```
void adjustBits() {
    int bitin     = used_bits%8;
    if (bitin <= 5) {
        rewind(3); /* rewind the input 3 bytes and read a few bits forward */
        getbits(bitin + 2);
    } else {
        rewind(2); /* rewind the input 2 bytes and read a few bits forward */
        getbits(bitin - 6);
    }
}
```

8.12 Informative : Adaptive Arithmetic Encoder for BIFS-Anim

The following procedures, in C code, describe the adaptative arithmetic encoder used in a BIFS-Anim session. The model is specified through the array `int* cumul_freq[ ]`. A symbol to be encoded is passed to the `encode()` routine, which modifies the model appropriately using the `update()` function.

First, the following integers are defined :

```
static long bottom=0, q1=2^14, q2=2^15, q3=3*2^14, top=2^16-1;
```

The encoder is initialized to start encoding an arithmetic coded bitstream by calling the following procedure:

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
static long low, high, usedbit, length, zerorun=0;

void encoder_reset( ) {
    low           = 0;
    high          = top;
    length        = 0;
    zerorun       = 0;
    usedbits      = 0;
}
```

In the BIFS-Anim encoding process, a symbol is encoded using a model specified through the array `cumul_freq[]` and by calling the following procedure.

```
Void aa_encode(int index, int cumul_freq[ ]) {
    length = high - low + 1;
    high   = low - 1 + (length * cumul_freq[index]) / cumul_freq[0];
    low   += (length * cumul_freq[index+1]) / cumul_freq[0];
    bitcount = 0;


    for ( ; ; ) {
        if (high < q2) {
            bit_opp_bits(false);
        }
        else if (low >= q2) {
            bit_opp_bits(true);
            low  -= q2;
            high -= q2;
        }
        else if (low >= q1 && high < q3) {
            opposite_bits++;
            low  -= q1;
            high -= q1;
        }
        else break;

        low *= 2;
        high = 2*high+1;
    }
}




void bit_opp_bits(int bit) {
    bit_in_psc_layer(bit);

    while(opposite_bits > 0){
        bit_in_psc_layer(1-bit);
        opposite_bits--;
    }
}

void bit_in_psc_layer(int  bit) {
    if(zerorun == 22) {
        write_bit(1);   /* marker to eliminate 23 bit 0 runs */
        bitcount++;
        zerorun=0;
    }

    write_bit(bit); /* write a bit into the bit stream */
    bitcount++;

    if(!bit) {
        zerorun++;
    } else {
        zerorun = 0;
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    }
}
```

The model is reset with the procedure `model_reset()` given in Appendix G. The model is updated when the value `symbol` is written with the `update_model()` procedure in Appendix G.

When the stream is to be flushed, the following routine is used.

```
void flush() {
    bitcount=0;
    opposite_bits++;

    if (low < q1) {
        bit_opp_bits(false);
    } else {
        bit_opp_bits(true);
    }
    reset();
}
```

8.13 View Dependent Object Scalability

8.13.1 Introduction

Coding of View-Dependent Scalability (VDS) parameters for texture can provide for efficient incremental decoding of 3D images (e.g. 2D texture mapped onto a 3D mesh such as terrain). Corresponding tools from the Visual and Systems parts of this specification (ISO/IEC 14496-2 and ISO/IEC 14496-1, respectively) are used in conjunction with downstream and upstream channels of a receiving terminal. The combined capabilities provide the means for a sending terminal to react to a stream of viewpoint information received from a receiving terminal. The sending terminal transmits a series of coded textures optimized for the viewing conditions, which can be applied to the rendering of, textured 3D meshes by the receiving terminal. Each encoded view-dependent texture (initial texture and incremental updates) typically corresponds to a specific 3D view in the user's viewpoint that is first transmitted from the receiving terminal.

A Systems tool transmits 3D viewpoint parameters in the upstream channel back to the sending terminal. The encoder's response is a frequency-selective, view-dependent update of DCT coefficients for the 2D texture (based upon view-dependent projection of the 2D texture in 3D) back to the receiving terminal, via the downstream channel, for decoding by a Visual DCT tool at the receiving terminal. This bilateral communication supports interactive server-based refinement of texture for low-bandwidth transmissions to a receiving terminal that renders the texture in 3D for a user controlling the viewpoint movement. A gain in texture transmission efficiency is traded for longer closed-loop latency in the rendering of the textures in 3D. The receiving terminal coordinates inbound texture updates with local 3D renderings, accounting for network delays so that texture cached in the terminal matches each rendered 3D view.

A method to obtain an optimal coding of 3D data is to take into account the viewing position in order to transmit only the most visible information. This approach reduces greatly the transmission delay, in comparison to transmitting all scene texture that might be viewable in 3D from the sending terminal's database server to the receiving terminal. At a given time, only the most important information is sent, depending on object geometry and viewpoint displacement. This technique allows the data to be streamed across a network, given that a upstream channel is available for sending the new viewing conditions to the remote database. This principle is applied to the texture data to be mapped on a 3D grid mesh. The mesh is first downloaded into the memory of the receiving terminal using the appropriate BIFS node, and then the DCT coefficients of the texture image are updated by taking into account the viewing parameters, i.e. the field of view, the distance and the direction to the viewpoint.

8.13.2 Bitstream Syntax

This subclause details the bitstream syntax for the upstream data and details the rules that govern the way in which higher level syntactic elements may be combined together to generate a compliant bitstream that can be decoded correctly by the receiving terminal.

8.13.3.1 specifies the bitstream syntax for a View Dependent Object which initializes the session at the upstream data decoder. 8.13.3.2 specifies the View Dependent Object Layer and contains the viewpoint information that is to be communicated back to the texture data encoder in the sending terminal.

8.13.2.1 View Dependent Object

```
class ViewDependentObject {
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   unsigned int (32) View_dep_object_start_code;
   unsigned int (16) Field_of_View;
   bit (1) Marker_bit;
   unsigned int (16) Xsize_of_rendering_window;
   bit (1) Marker_bit;
   unsigned int (16) Ysize_of_rendering_window
   bit (1) Marker_bit;
   unsigned int (32)* NextStartCode;
   while (NextStartCode == view_dep_object_layer_start_code){
      ViewDependentObjectLayer vdol;
      unsigned int (32)* NextStartCode;
   }
}

class ViewDependentObjectLayer() {
   unsigned int (32) View_dep_object_layer_start_code;
   unsigned int (16) Xpos1 ;
   bit (1) Marker_bit;
   unsigned int (16) Xpos2;
   bit (1) Marker_bit;
   unsigned int (16) Ypos1;
   bit (1)Marker_bit;
   unsigned int (16) Ypos2;
   bit (1) Marker_bit;
   unsigned int (16) Zpos1;
   bit (1) Marker_bit;
   unsigned int (16) Zpos2;
   bit (1) Marker_bit;
   unsigned int (16) Xaim1;
   bit (1) Marker_bit;
   unsigned int (16) Xaim2;
   bit (1) Marker_bit;
   unsigned int (16) Yaim1;
   bit (1) Marker_bit;
   unsigned int (16) Yaim2;
   bit (1) Marker_bit;
   unsigned int (16) Zaim1;
   bit (1) Marker_bit;
   unsigned int (16) Zaim2;
}
```

8.13.3 Bitstream Semantics

8.13.3.1 View Dependent Object

view_dep_object_start_code: The view_dep_object_start_code is the string '000001BF' in hexadecimal. It initiates a view dependent object session.

field_of_view: This is a 16-bit unsigned integer that specifies the field of view.

marker bit: This is a one bit field, set to '1', to prevent start code emulation within the bitstream.

xsize_of_rendering_window: This is a 16-bit unsigned integer that specifies the horizontal size of the rendering window.

ysize_of_rendering_window: This is a 16-bit unsigned integer that specifies the vertical size of the rendering window.

8.13.3.2 View Dependent Object Layer

view_dep_object_layer_start_code: The view_dep_object_layer_start_code is the bit string '000001BE' in hexadecimal. It initiates a view dependent object layer.

xpos1: This is a 16-bit codeword which forms the lower 16 bits of the 32-bit integer xpos. The integer xpos is to be computed as follows: xpos = xpos1 + (xpos2 << 16). The quantities xpos, ypos, zpos describe the 3D coordinates of the viewer's position.

xpos2: This is a 16-bit codeword which forms the upper 16-bit word of the 32-bit integer xpos.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ypos1: This is a 16-bit codeword which forms the lower 16-bit word of the 32-bit integer ypos. The integer ypos can be computed as follows: ypos = ypos1 + (ypos2 << 16).

ypos2: This is a 16-bit codeword which forms the upper 16bit word of the 32-bit integer xpos.

zpos1: This is a 16-bit codeword which forms the lower 16 bits of the 32-bit integer xpos. The integer zpos can be computed as follows: zpos = zpos1 + (zpos2 << 16).

zpos2: This is a 16-bit codeword which forms the upper 16 bits of the 32-bit integer xpos.

xaim1 – This is a 16-bit codeword which forms the lower 16 bits of the 32-bit integer xaim. The integer xaim can be computed as follows: xaim = xaim1 + (xaim2 << 16). The quantities xaim, yaim, zaim describe the 3D position of the aim point.

xaim2: This is a 16-bit codeword which forms the upper 16 bits of the 32-bit integer xaim.

yaim1: This is a 16-bit codeword which forms the lower 16 bits of the 32-bit integer yaim. The integer yaim can be computed as follows: yaim = yaim1 + (yaim2 << 16).

yaim2: This is a 16-bit codeword which forms the upper 16 bits of the 32-bit integer yaim.

zaim1: This is a 16-bit codeword which forms the lower 16 bits of the 32-bit integer zaim. The integer zaim can be computed as follows: zaim = zaim1 + (zaim2 << 16).

zaim2: This is a 16-bit codeword which forms the upper 16 bits of the 32-bit integer zaim.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9 The Extensible MPEG-4 Textual Format

9.1 Introduction

The XMT framework consists of two levels of textual syntax and semantics: the XMT-A format and the XMT-Ω format, which we will abbreviate by A and Ω, respectively, and use them interchangeably where there is no confusion.

The XMT-A is an XML-based version of MPEG-4 content, which contains a subset of the X3D. Also contained in XMT-A is an MPEG-4 extension to the X3D to represent MPEG-4 specific features. The XMT-A provides a straightforward, one-to-one mapping between the textual and binary formats.

The XMT-Ω is a high-level abstraction of MPEG-4 features designed based on the W3C SMIL. The XMT provides a default mapping from Ω to A, for there is no deterministic mapping between the two, and it also provides content authors with an escape mechanism from Ω to A.

In addition an XMT-C (Common) section contains the definition of elements and attributes that may be used within either XMT-A or XMT-Ω.

9.2 XMT-A Format

9.2.1 Introduction

This section contains the XMT-A format definition that has the goals of representing binary constructs in a textual format, providing an informational one-to-one deterministic mapping to binary coding.

9.2.2 XMT-A Document structure

An XMT-A document has a single optional <Header> element followed by a single <Body> element. The <Header> element contains zero or more <meta> elements and also contains the MPEG-4 specific element for the <InitialObjectDescriptor>.

9.2.2.1 Identifiers and forward references within a document instance

An XMT-A document instance is comprised of a set of elements to represent MPEG-4 systems streams as a textual format. In the textual format the order of the elements in the document is not necessarily the same order as the corresponding binary constructs in the streams. There is however an understood mapping see subclause 9.2.13 for more details.

So elements in the document are timed using a <par> element with a begin attribute to specify the time. These <par> elements may also be nested, and the timing of the nested <par> elements is relative to its parent (as in fact are the top level <par>s because they can be considered to be nested in the topmost implicit par that comprises the body of the document that begins at 0s). To create the binary streams the elements are sorted in temporal order maintaining the document order of any elements that are for the same time. Since the elements may be out of order the question is are forward references allowed of identifiers that are mapped to binary streams.

The answer is that forward references within the document are permitted. However within a single stream, if as the elements are sorted in time then any forward references that remain must not be in violation of the MPEG-4 Systems specification if forward references are not permitted for that stream. I.e. if the temporal sorting does not eliminate forward references and this causes an illegal stream due to unknown Ids, because of the forward references, then some alternate representation should be sought. And across streams forward references in the document are also permitted if the coding leads to valid MPEG-4 system streams.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.3 XMT-A Representation of Nodes

9.2.3.1 Overview

This section provides a description of the XMT-A textual representation of MPEG-4 nodes.

9.2.3.2 XMT-A node elements

9.2.3.2.1 MPEG-4 node/field to XMT-A element/attribute mapping algorithm

The following algorithm is used to convert MPEG-4 nodes and fields to XMT-A elements and attributes.

Each node is converted to an XMT-A element, with its name preserved.

For each field of a node

If the field type is a node, i.e., the field can contain one or more children nodes, then the field is converted to an XMT-A element, with the element name identical to the field name. This element will appear as the child element.

If the field type is non-node and is a plain Field or an exposedField, then the field is made into an attribute of the element, preserving its name. (Fields with eventIn and eventOut types are omitted as they are not encoded and these cannot usefully be attributes of the element.)

An exception to the above rule for node/non-node field conversion is for the <Conditional> node, where the buffer field, although a non-node field, is converted to an element so that it can contain one or more BIFS command elements in this XML representation .

A field without a default value is optional. Fields with MPEG-4 default binary values are given default XML attributes with the same values.

When numerical multiple value fields are to be encoded using predictiveMFField coding, the node shall have a sequence of <PredictiveField> elements as children, one for each encoded field.

9.2.3.2.2 Common attributes and elements

Optional DEF and USE attributes are present on all XMT-A node elements. XMT-A adds the following optional common attributes:

binaryID for deterministic binary encoding,

DEF to code id as name,

9.2.3.2.3 Element and attribute type classifications

Node elements and field attribute types will be classified according to MPEG-4 system node types.

9.2.3.3 Schema and XMT-A examples

Given the algorithm described above, MPEG-4 nodes can easily be converted into XMT-A. This section provides some examples to illustrate the representation. The full set of nodes from clause 7 can be converted this way. The Schema for XMT-A, containing the full set of nodes, can be found in an external annex to this document in a file named **xmt-a.xsd**.

The following example shows the MPEG-4 node Material converted to the XMT-A element <Material>. The Material node has no fields that are nodes and so all its fields have become attributes and DEF/USE is included as a predefined attribute group. The IS subelement is used when the node is part of a PROTO declaration.

```
<element name="Material">
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  <complexType>
    <sequence>
      <element ref="xmta:IS" minOccurs="0" />
    </sequence>
    <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>
```

The DefUseGroup defines the attributes DEF, USE and binaryID. DEF is a string that contains a symbolic ID. USE is a string that refers to a node that has the matching symbolic ID. The type of the element with the USE attribute shall match the type of the element having the matching DEF symbolic ID. The attribute binaryID is an integer defining the binary nodeID to be used when encoding the scene in BIFS.

Some examples of use of the above definition are:

```
<Material ambientIntensity="0.6" emissiveColor=”1.0 0.1 0.78”/>
<Material DEF=”ABlue” emissiveColor=”0.0 0.1 0.88”/>
<Material USE=”ABlue”/>
```

The following example shows the MPEG-4 node **OrderedGroup** converted to the XMT-A element <OrderedGroup>. The **OrderedGroup** node has the children fields that is of type multiple nodes and so that field is converted to an element whilst its other field (order) has become an attribute.

```
<element name="OrderedGroup">
  <complexType>
    <all>
      <element ref="xmta:IS" minOccurs="0" />
      <element name="children" minOccurs="0" form="qualified">
        <complexType>
          <group ref="xmta:SF3DNodesType" minOccurs="0" maxOccurs="unbounded" />
        </complexType>
      </element>
    </all>
    <attribute name="order" type="xmta:MFFloat" use="optional" />
    <attributeGroup ref="xmta:DefUseGroup" />
  </complexType>
</element>
```

An example of its use (the Shapes are incomplete for simplicity) is:

```
<OrderedGroup order=”1.2 6.5”>
  <children>
    <Shape>…<Shape/>
    <Shape>…<Shape/>
  </children>
</OrderedGroup>
```

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.4 XMT-A Routing

9.2.4.1 <ROUTE>

9.2.4.1.1 Description

The <ROUTE> element is the XMT-A representation of the ROUTE. The optional id attribute names the ROUTE and allows the ROUTE to be deleted or replaced at a later time by referring to it via the atRoute attribute of BIFS Commands.

```
<element name="ROUTE">
  <complexType>
    <attribute name="DEF"       type="ID"      use="optional"/>
    <attribute name="binaryID"  type="int"     use="optional"/>
    <attribute name="fromNode"  type="IDREF"   use="required"/>
    <attribute name="fromField" type="xmta:fieldName" use="required"/>
    <attribute name="toNode"    type="IDREF"   use="required"/>
    <attribute name="toField"   type="xmta:fieldName" use="required"/>
  </complexType>
</element>
```

<ROUTE>s can be placed inside the <Scene> element before the closing </Scene>. <ROUTE>s cannot be included inside other elements of the scene.

9.2.5 PROTOs in XMT-A

9.2.5.1 PROTO Declaration

The <ProtoDeclare> element is the equivalent of the PROTO Declaration defined in 8.7.2.2

```
<ProtoDeclare name="" protoID="">
  <field …/> // as many as needed
  …// then nested ProtoDeclare(s)
  …// then nodes
  …// then ROUTEs
</ProtoDeclare>
```

9.2.5.2 PROTO Instance

```
<ProtoInstance name="">
  <fieldValue …/> // as many as needed
</ProtoInstance>
```

9.2.5.3 field

The element <field>, subelement of <ProtoDeclare>, allows the definition of the PROTO declaration interface, as defined in subclause 8.7.2.3.

```
<field name="" type="" vrml97hint="" [typedvalue]="">
  <InterfaceCodingParameters quantCategory="" animCategory="" nbBits=""
                             position3Dmin="" position3Dmax=""
                             position2Dmin="" position2Dmax=""
                             drawOrderMin="" drawOrderMax=""
                             colorMin="" colorMax=""
                             textureCoordinateMin="" textureCoordinateMax=""
                             angleMin="" angleMax=""
                             scaleMin="" scaleMax=""
                             keyMin="" keyMax=""
                             sizeMin="" sizeMax=""
   />
  …// SFNode or MFNode default value here
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
</field>
```

Name is the name of the field, type is an enumerated value from Table 94, vrml97hint is an enumerated value from Table 93. If the field has a default value which is neither a node or a set of nodes, the value is specified as an attribute. The name of the attribute is specific to the type of the field, as shown in Table 95. This type-specific attribute name allows efficient type-checking of the default value with XML Schema.

The InterfaceCodingParameters element allows the specification of the BIFS element of the same name described in subclause 8.7.2.7.

Table 93 — vrml97hint values

scope of the field	*vrml97hint value*
eventIn	**eventIn**
eventOut	**eventOut**
field	**field**
exposedField	**exposedField**

Table 94 — type values

field type	*type value*
SFBool	**Boolean**
MFBool	**Booleans**
SFColor	**Color**
MFColor	**Colors**
SFFloat	**Float**
MFFloat	**Floats**
SFImage	**Image**
MFImage	**Images**
SFInt32	**Integer**
MFInt32	**Integers**
SFNode	**Node**
MFNode	**Nodes**
SFRotation	**Rotation**
MFRotation	**Rotations**
SFString	**String**
MFString	**Strings**
SFTime	**Time**
MFTime	**Time**
SFVec2f	**Vector2**
MFVec2f	**Vector2Array**
SFVec3f	**Vector3**
MFVec3f	**Vector3Array**

Table 95 — typeToAttributeName correspondance

type of field	*name of value attribute*
SFBool	**booleanValue**
MFBool	**booleanArrayValue**
SFColor	**colorValue**
MFColor	**colorArrayValue**
SFFloat	**floatValue**
MFFloat	**floatArrayValue**
SFImage	**imageValue**
MFImage	**imageArrayValue**
SFInt32	**integerValue**
MFInt32	**integerArrayValue**
SFNode	**nodeValue**

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

MFNode	**nodeArrayValue**
SFRotation	**rotationValue**
MFRotation	**rotationArrayValue**
SFString	**stringValue**
MFString	**stringArrayValue**
SFTime	**timeValue**
MFTime	**timeArrayValue**
SFVec2f	**vector2Value**
MFVec2f	**vector2ArrayValue**
SFVec3f	**vector3Value**
MFVec3f	**vector3ArrayValue**

9.2.5.4 fieldValue

The element <fieldValue>, subelement of <ProtoInstance> allows the definition of the PROTO instance interface.

```
<fieldValue name="" [typedvalue]="">
  …// SFNode or MFNode value here
</fieldValue>
```

Name is the name of the field, type is an enumerated value from Table 94, vrml97hint is an enumerated value from Table 93. If the field has a default value which is neither a node or a set of nodes, the value is specified as an attribute. The name of the attribute is specific to the type of the field, as shown in Table 95. This type-specific attribute name allows efficient type-checking of the value with XML Schema.

9.2.5.5 IS

Inside a PROTO declaration, any value can be ISed to one of the fields in the PROTO interface. This IS link is specified with an <IS> element, which can only appear inside <ProtoDeclare>, but may appear in any node, <field> and <fieldValue> element. The <IS> element is a wrapper for one or more IS links. Each IS link is specified through a <connect> element:

```
<IS>
  <connect nodeField="" protoField=""/> // as many as needed
</IS>
```

nodeField refers to the name of the field of the parent node element. protoField refers to the name of the field of one of the parent proto declaration interface which should be connected to nodeField. A node field can be connected to many proto fields. A proto field can be connected to many node fields.

9.2.5.6 Example of PROTO in XMT-A

```
<ProtoDeclare name="P0" protoID="0">
  <field name="field0" type="Vector2" vrml97Hint="field" vector2Value="0.0 0.0"/>
  <Transform2D>
    <IS>
      <connect nodeField="translation" protoField="field0"/>
    </IS>
    <children>
      <Shape>
        <geometry>
          <Rectangle size="0.2 0.2"/>
        </geometry>
      </Shape>
    </children>
  </Transform2D>
</ProtoDeclare>

<ProtoInstance name="P0">
  <fieldValue name="field0" vector2Value="0.0 -0.6"/>
```


-`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
</ProtoInstance>
```

9.2.6 XMT-A Timing

The XMT-A uses one of the SMIL time containers, the <par> element, to group multiple commands.

9.2.6.1 <par>

The XMT-A allows only the "begin" attribute on the <par> element to specify the execution (begin) time of commands. Moreover <par> elements can also contain other <par> elements and for the nested <par> elements their begin time is relative to the parent time container. There is an implied top level <par begin="0.0">. The <par> elements need not appear ordered in time, indeed nesting of <par> elements will often preclude this. Begin times shall be >= 0.0 seconds. The attribute begin has an SFTime type to maintain uniformity with other time fields, in MPEG-4 node elements such as <TimeSensor> and <MovieTexture>, within the scene.

The <par> element may contain
<par>
BIFS Commands
BIFS Anim
Object Descriptor Commands
IPMP Messages
OCI Events
MPEG-J Stream Headers

For a given BIFS stream, all BIFS commands, to be executed at a given time will be coded into a single CommandFrame (and hence a single AU) in the order the commands appear in the document.

All OD commands, for a given OD stream, to be executed at a given time will be coded into a single AU in the order the commands appear in the document.

All IPMP messages, for a given IPMP elementary stream, to be executed at a given time will be coded into a single AU in the order the messages appear in the document.

All OCI Events messages, for a given OCI elementary stream, to be executed at a given time will be coded into a single AU in the order the events appear in the document.

```
<par begin="" atES_ID="">
  <!-- Any number of commands/messages/events and/or <par> elements -->
</par>
```

The atES_ID attribute indicates the destination of the commands and descriptors which are children of the par element. It specifies a symbol which is the ID of the elementary stream into which the commands and descriptors should be encoded. In the case of conflicting atES_ID values, the attribute closest to the command or descriptor takes precedence.

9.2.7 XMT-A Representation of BIFS Commands

9.2.7.1 Overview

This section provides a detailed description of the XMT-A encoding of the MPEG-4 BIFS commands. Commands in BIFS are timed using <par> element construct.

There are three basic BIFS commands in XMT-A: <Insert>, <Delete>, <Replace>. The MPEG-4 ReplaceScene binary command is captured in XMT-A with <Replace> <Scene>… </Scene></Replace>. <Insert>, <Delete> and <Replace> commands can be used on nodes, values in multiple value fields, or routes. In addition <Replace> can act on a whole multiple value field. <Replace> <Scene> replaces the entire scene – both nodes and routes.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.7.2 <Insert>

Insert command provides for node, Indexed value and Route insertion. For Insert atField defaults to value ‘children’ and position defaults to value ‘END’ making it easy to add a Node to a group.

```
<Insert atES_ID=”” atNode="" atRoute=””
        atField="children" position="BEGIN | END | n" value=””>
  <!-- Nodes (including sub-trees) may go here and/or Routes-->
</Insert>
```

The atES_ID attribute indicates the destination of the command. It specifies a symbol which is the ID of the elementary stream into which the command should be encoded. In the case of conflicting atES_ID values, the attribute closest to the command or descriptor takes precedence.

Whenever multiple nodes are inserted, the default for each position is END. If the position attribute contains more values than there are nodes or values to be inserted, then the extra position values are ignored. If the position attribute is not present or contains not enough values, the remaining nodes are inserted at END by default.

9.2.7.2.1 Insert Node

This is the Insert Node version of the Insert command. When atField=”children” (defaulted if attribute not present) the command will be encoded as BIFS Update for Insert Node by nodeID. When atField is any other MFNode field then it will be encoded as BIFS Update for Insert by IndexedValue.

```
<Insert atNode="" atField="children" position="BEGIN | END | n">
  <!-- Node (tree) goes here -->
</Insert>
```

The following are examples of Insert used to insert a node

Inserts a new sub-tree at the END of a group

```
<Insert atNode="MyGroup">
  <Group>
    <children>...</children>
  </Group>
</Insert>
```

the following is equivalent to above example (atField="children" is default)

```
<Insert atNode="MyGroup" atField="children">
  <Group>
    <children>...</children>
  </Group>
</Insert>
```

Inserts a new shape at the END of a group (geometry and appearance content omitted for clarity)

```
<Insert atNode="MyGroup">
  <Shape>
    <geometry>...</geometry>
    <appearance><Appearance DEF="MyShapeStyle">...</appearance>
  </Shape>
</Insert>
```

Inserts 2 new shapes, one at BEGIN and other at the END of a group

```
<Insert atNode="MyGroup" position="BEGIN, END">
  <Shape DEF="MyRect">
    <geometry>...</geometry>
    <appearance>...</appearance>
  </Shape>
  <Shape USE="MyRect"/>
</Insert>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.7.2.2 Insert Indexed Value

This is the InsertIndexedValue version of Insert for simple non-Node fields

```
<Insert atNode="" atField="" position="BEGIN | END | n" value=””/>
```

The following are examples of Insert used to insert simple field values (non-Node values)

Inserts a new colorIndex value 6 at the END of the colorIndex field

```
<Insert atNode="MyIndexedLineSet2D" atField="colorIndex" value="6" />
```

Inserts new colorIndex values 3,3,5,4,2 at positions 2,7,BEGIN, END and 4. Note that position 7 is the new position after the first value has been inserted at position 2 etc. Inserts are done in the order listed The following command will be encoded as 5 BIFS Update commands for IndexedValue field insertion)

```
<Insert atNode="MyIndexedLineSet2D" atField="colorIndex"
        position="2 7 BEGIN END 4"
        value="3 3 5 4 2" />
```

9.2.7.2.3 Insert Route

This is InsertRoute version of Insert

```
<Insert>
  <ROUTE fromNode="" fromField="" toNode="" toField=""/>
</Insert>
```

The following are examples of Insert used to insert a Route

Inserts a ROUTE without an ID

```
<Insert>
  <ROUTE fromNode="WhiteRect" fromField="emissiveColor"
         toNode="ACircle"       toField="emissiveColor"/>
</Insert>
```

Inserts a ROUTE with an ID

```
<Insert>
  <ROUTE DEF="myROUTE" fromNode="WhiteRect" fromField="emissiveColor"
         toNode="ACircle" toField="emissiveColor"/>
</Insert>
```

9.2.7.3 <Delete>

Delete command provides for node, Indexed value and Route deletion. For Delete atField defaults to value ‘children’ but position has no default.

```
<Delete atES_ID=”” atRoute=”” atNode=""
        atField="children" position="BEGIN | END | n"/>
```

Attributes atNode and atRoute are exclusive.

The atES_ID attribute indicates the destination of the command. It specifies a symbol which is the ID of the elementary stream into which the command should be encoded. In the case of conflicting atES_ID values, the attribute closest to the command or descriptor takes precedence.

9.2.7.3.1 Delete Node

This is the Delete Node version of the Delete command.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
<Delete atNode=””/>
```

The following is an example of Delete used to delete a node

Deletes a DEF'd node

```
<Delete atNode="MyIndexedLineSet2D"/>
```

9.2.7.3.2 Delete Indexed Value

This is the DeleteIndexedValue version of Delete for Nodes and simple non-Node fields

```
<Delete atNode="" atField="" position="BEGIN | END | n"/>
```

The following are examples of Delete used to delete both simple field values (non-Node values) and Nodes

Delete colorIndex value at the END of the colorIndex field

```
<Delete atNode="MyIndexedLineSet2D" atField="colorIndex" position=”END”/>
```

Deletes colorIndex values at positions 2,7,BEGIN, END and 4. Note that position 7 is the new position after the first value has been deleted at position 2 etc. Deletes are done in the order listed.

```
<Delete atNode="MyIndexedLineSet2D" atField="colorIndex"
        position="2 7 BEGIN END 4"/>
```

Deletes 2 colors (nodes), one at pos 7 and one at the END of the color field.

```
<Delete atNode="MyIndexedLineSet2D" atField="color" position="7 END"/>
```

9.2.7.3.3 Delete Route

This is the Delete Route version of Delete

```
<Delete atRoute=””/>
```

The following are examples of Delete used to delete Routes

Deletes a named Route

```
<Delete atRoute=”BlueRoute”/>
```

9.2.7.4 <Replace>

Replace command provides for node, field, Indexed value, Route and Scene replacement. For Replace neither atField nor position have default values.

```
<Replace atES_ID=”” atRoute=”” atNode=""
         atField="children" position="BEGIN | END | n" value=””>
  <!-- Nodes (including sub-trees) may go here and/or Routes -->
</Replace>
```

Attributes atNode and atRoute are exclusive.

The atES_ID attribute indicates the destination of the command. It specifies a symbol which is the ID of the elementary stream into which the command should be encoded. In the case of conflicting atES_ID values, the attribute closest to the command or descriptor takes precedence.

Whenever multiple nodes are replaced, the default for each position is END. If the position attribute contains more values than there are nodes or values replaced, then the extra position values are ignored. If the position attribute is not present or contains not enough values, the remaining nodes or values are replaced at END by default.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.7.4.1 Replace Node

This is the Replace Node version of the Replace command.

```
<Replace atNode="" />
```

The following are examples of Replace used to replace a node

Replaces existing MyGroup node with new sub-tree

```
<Replace atNode="MyGroup">
  <Group>
    <children>...</children>
  </Group>
</Replace>
```

Replaces existing shape MyRect node with new one MyCircle

```
<Replace atNode="MyRect">
  <Shape DEF="MyCircle">
    <geometry>...</geometry>
    <appearance><Appearance DEF="MyShapeStyle">...</appearance>
  </Shape>
</Replace>
```

Note: Unlike Insert it makes no sense to allow multiple specification of Nodes here as they would all replace the same target.

9.2.7.4.2 Replace Field

This is the Replace Field version of the Replace command. When position attribute is not specified the entire field is to be replaced.

```
<Replace atNode="" atField=" value="">
  <!—A Node may go here -->
</Replace>
```

The following are examples of Replace used to replace a node

Replaces colorIndex (replace entire MF field)

```
<Replace atNode="MyIndexedLineSet2D" atField="colorIndex"
         value= "6 5 6 7 7 2 1" />
```

Replaces emissiveColor field (replace SF field)

```
<Replace atNode="MyMaterial" atField="emissiveColor"
         value= "1.0 0.5 0.2" />
```

Replaces color field (replace SF field or type SFNode)

```
<Replace atNode="MyIndexedFaceSet" atField="color">
  <Color color="1.0 0.5 0.9   0.8 0.5 0.1   0.5 0.5 0.5"
</Replace>
```

Note: Specifying multiple field positions makes no sense (similar comment at end of Replace Node above)

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.7.4.3 Replace Indexed Value

This is the Replace Indexed Value version of the Replace command. Position attribute is specified and hence an indexed value replacement in an MFField is replaced.

```
<Replace atNode="" atField=" position="BEGIN | END | n" value=""/>
  <!-- Nodes may go here -->
</Replace>
```

The following are examples of Replace used to replace an indexed value

Replaces colorIndex value at the END of the colorIndex field to 6

```
<Replace atNode="MyIndexedLineSet2D" atField="colorIndex" position="END"
         value="6" />
```

Replaces colorIndex values 3,3,5,4,2 at positions 2,7,BEGIN, END and 4. Replaces are done in the order listed.

```
<Replace atNode="MyIndexedLineSet2D" atField="colorIndex"
         position="2 7 BEGIN END 4"
         value="3 3 5 4 2" />
```

This is still a replace indexed value (encodes to 5 IndexedValue BIFS Replace commands) even if the field had five elements and this is replacing them all. To specify a complete field replacement the position should be omitted and then a single BIFS Replace Field command would be encoded.

```
<Replace atNode="MyIndexedLineSet2D" atField="colorIndex"
         position="0 1 2 3 4"
         value="3 3 5 4 2" />
```

Replaces existing node in MySwitch choice field at position 5 with new one

```
<Replace atNode="MySwitch" atField="choice" position="5">
  <Shape>
    <geometry><Circle radius="100"/></geometry>
    <appearance>...</appearance>
  </Shape>
</Replace>
```

Replaces existing node in MyGroup choice field at position 3 and 7 with new ones

```
<Replace atNode="MyGroup" atField="children" position="3, 7">
  <Shape>
    <geometry><Circle radius="10"/></geometry>
    <appearance>...</appearance>
  </Shape>
  <Shape>
    <geometry><Rectangle width="10" heihgt="70"/></geometry>
    <appearance>...</appearance>
  </Shape>
</Replace>
```

9.2.7.4.4 Replace Route

This is the Replace Route version of the Replace command.

```
<Replace atRoute="">
  <ROUTE fromNode="" fromField="" toNode="" toField=""/>
</Replace>
```

The following are examples of Replace used to replace an indexed value

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Replaces existing route ‘ColorRoute” with a new route fromNode/Field to Node/Field.

```
<Replace atRoute=”ColorRoute”>
  <ROUTE fromNode="WhiteRect"   fromField="emissiveColor"
         toNode="ACircle"       toField="emissiveColor"/>
</Replace>
```

9.2.7.4.5 Replace Scene

This is the Replace Scene version of the Replace command.

```
<Replace>
  <Scene useNames=”true|false”>
  <!-- Nodes followed by -->
  <!-- ROUTEs go here -->
  </Scene>
</Replace>
```

The following are examples of Replace used to replace a scene

Replaces Scene.

```
<Replace>
  <Scene>
    <Group>
      <children>
        <Shape DEF="MyRect">
          <geometry>...</geometry>
          <appearance>...</appearance>
        </Shape>
        <Shape USE="MyRect"/>
      </children>
    </Group>
    <ROUTE DEF="MatColorRoute"
           fromNode="Mat1" fromField="emissiveColor"
           toNode="Mat2"   toField=emissiveCOlor" />
    <ROUTE fromNode="LineProp1" fromField="lineColor"
           toNode="LineProp2"   toField=lineColor" />
  </Scene>
</Replace>
```

ROUTES may appear only appear after nodes in a list immediately before the closing </Scene>. When encoded to MPEG-4 the ROUTES shall be included in the ReplaceScene in the order they appear in the document.

Note that MPEG-4 requires the top-level node to be Group, OrderedGroup, Layer2D or Layer3D. The author should ensure that the outermost node in a replace scene conforms to this requirement.

9.2.8 XMT-A Representation of Object Descriptors

9.2.8.1 Overview

Object descriptor is an MPEG-4 concept described in the Systems specification. The purpose of the object descriptors is to identify and describe elementary streams and to associate them appropriately to an audio-visual scene description. Object descriptors serve to gain access to ISO/IEC 14496 content.

An object descriptor is a collection of one or more elementary stream descriptors that provide the configuration and other information for the streams that relate to either an audio-visual object or a scene description. Object descriptors are themselves conveyed in elementary streams. Each object descriptor is assigned an identifier (object descriptor ID), which

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

is unique within a defined name scope. This identifier is used to associate audio-visual objects in the scene description with a particular object descriptor, and thus the elementary streams related to that particular object.

Elementary stream descriptors include information about the source of the stream data, in the form of a unique numeric identifier (the elementary stream ID) or a URL pointing to a remote source for the stream. Elementary stream descriptors also include information about the encoding format, configuration information for the decoding process and the sync layer packetization, as well as quality of service requirements for the transmission of the stream and intellectual property identification. Dependencies between streams can also be signaled within the elementary stream descriptors. This functionality may be used, for example, in scalable audio or visual object representations to indicate the logical dependency of a stream containing enhancement information, to a stream containing the base information. It can also be used to describe alternative representations for the same content (e.g. the same speech content in various languages).

To convert MPEG-4 descriptors to an XML representation the following guidelines were followed. The descriptor is first converted to an element with the same name. Then for simple fields (e.g bit(5), bit(8) etc) these are converted to attributes taking the value of the simple field, unless they are part of an if construct. In the latter case the field is converted to an attribute of an element having the name of the if condition flag. For arrays of descriptors e.g ES_Descriptor esDescr[1..255], these have been converted to an element having the name as the array and containing descriptors of the given type.

9.2.8.2 <Object Descriptor>

```
<ObjectDescriptor
  objectDescriptorID=""
  binaryID="">
  <URL URLstring="">
  <Descr>
    <esDescr>...</esDescr>
    <ociDescr>...</ociDescr>
    <ipmpDescrPtr>...</ipmpDescrPtr>
  </Descr>
  <extDescr>...</extDescr>
</ObjectDescriptor>
```

9.2.8.2.1 Description

The <ObjectDescriptor> element is the XMT-A representation of the ObjectDescriptor as described in subclause 7.2.6.3, ISO/IEC 14496-1.

The XMT-A Schema allows either the <URL> element or the <Descr> element and corresponds to the if (URL_Flag) construct. The <URL> element contains the URLstring and if present the descriptor will be binary encoded with the URL_Flag field bit set to 1 and the URLlength attribute set to the length of the URLstring. If the <Descr> element is present then URL_Flag field bit will be encoded as 0.

If the form of the url attribute in the media element is url=""od:MyOdIdentifier"", ie. the url starts with **od:** or **od://** then the url shall be coded as an Odid otherwise it shall be coded as a string (urlValue). The form as above assumes the SF/MFString representation. The part after **od:** or **od://** is in fact the objectDescriptorID (its id) of an Object Descriptor.

Example:

```
<AudioSource  url="od://PanelAudio"/>

<ObjectDescriptor objectDescriptorID="PanelAudio">
  . . .
</ObjectDescriptor>
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.8.3 <InitialObjectDescriptor>

```
<InitialObjectDescriptor
  objectDescriptorID=""
  binaryID="">
  <URL URLstring="""/>
  <Profiles
      ODProfileLevelIndication="None"
      sceneProfileLevelIndication="Simple2D_L1"
      audioProfileLevelIndication="Unspecified"
      visualProfileLevelIndication="254"
      graphicsProfileLevelIndication="254"
      includeInlineProfileLevelFlag="true|false"/>
  <Descr>
    <esDescr>...</esDescr>
    <ociDescr>...</ociDescr>
    <ipmpDescrPtr>...</ipmpDescrPtr>
  </Descr>
  <extDescr>...</extDescr>
</InitialObjectDescriptor>
```

9.2.8.3.1 Description

The <InitialObjectDescriptor> element is the XMT-A representation of the InitialObjectDescriptor as described in subclause 7.2.6.4, ISO/IEC 14496-1.

The `InitialObjectDescriptor` is a variation of the `ObjectDescriptor` specified in the previous subclause that allows to signal profile and level information for the content referred by it. It shall be used to gain initial access to ISO/IEC 14496 content.

The XMT-A Schema allows either the <URL> element or the <ProfDescr> element and corresponds to the if (URL_Flag) construct. The <URL> element contains the URLstring and if present the descriptor will be binary encoded with the URL_Flag field bit set to 1 and the URLlength attribute set to the length of the URLstring. If the <ProfDescr> element is present then URL_Flag field bit will be encoded as 0.

The attributes ODProfileLevelIndication, sceneProfileLevelIndication, audioProfileLevelIndication, visualProfileLevelIndication and graphicsProfileLevelIndication take either a numeric value from 0 to 255 inclusive or a string corresponding to the enumerated values as follows:

ODProfileLevelIndication	
Unspecified	254
None	255

sceneProfileLevelIndication	
Simple2D_L1	1
Simple2D_L2	2
Basic2D_L1	3
Core2D_L1	4
Core2D_L2	5
Main2D_L1	6
Main2D_L2	7
Main2D_L3	8
Advanced2D_L1	9
Advanced2D_L1	10
Advanced2D_L3	11
Unspecified	254

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

None	255

audioProfileLevelIndication	
Main_L1	1
Main_L2	2
Main_L3	3
Main_L4	4
Scalable_L1	5
Scalable_L2	6
Scalable_L3	7
Scalable_L4	8
Speech_L1	9
Speech_L2	10
Synthesis_L1	11
Synthesis_L2	12
Synthesis_L3	13
High_Quality_Audio_L1	14
High_Quality_Audio_L2	15
High_Quality_Audio_L3	16
High_Quality_Audio_L4	17
High_Quality_Audio_L5	18
High_Quality_Audio_L6	19
High_Quality_Audio_L7	20
High_Quality_Audio_L8	21
Low_Delay_Audio_L1	22
Low_Delay_Audio_L2	23
Low_Delay_Audio_L3	24
Low_Delay_Audio_L4	25
Low_Delay_Audio_L5	26
Low_Delay_Audio_L6	27
Low_Delay_Audio_L7	28
Low_Delay_Audio_L8	29
Natural_Audio_L1	30
Natural_Audio_L2	31
Natural_Audio_L3	32
Natural_Audio_L4	33
Mobile_Audio_Internetworking_L1	34
Mobile_Audio_Internetworking_L2	35
Mobile_Audio_Internetworking_L3	36
Mobile_Audio_Internetworking_L4	37
Mobile_Audio_Internetworking_L5	38
Mobile_Audio_Internetworking_L6	39
Unspecified	254
None	255

VisualProfileLevelIndication	
Simple_L3	1
Simple_L2	2
Simple_L1	3

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

Simple_Scalable_L2	4
Simple_Scalable_L1	5
Core_L2	6
Core_L1	7
Main_L4	8
Main_L3	9
Main_L2	10
N-Bit_L2	11
Hybrid_L2	12
Hybrid_L1	13
Basic_Animated_Texture_L2	14
Basic_Animated_Texture_L1	15
Scalable_Texture_L3	16
Scalable_Texture_L2	17
Scalable_Texture_L1	18
Simple_Face_Animation_L2	19
Simple_Face_Animation_L1	20
Simple_FBA_L2	21
Simple_FBA_L1	22
Basic_Animated_Texture_L2	23
Basic_Animated_Texture_L1	24
Hybrid_Profile_L2	25
Hybrid_Profile_L1	26
Advanced_Real_Time_Simple_L4	27
Advanced_Real_Time_Simple_L3	28
Advanced_Real_Time_Simple_L2	29
Advanced_Real_Time_Simple_L1	30
Core_Scalable_L3	31
Core_Scalable_L2	32
Core_Scalable_L1	33
Advanced_Coding_Efficiency_L4	34
Advanced_Coding_Efficiency_L3	35
Advanced_Coding_Efficiency_L2	36
Advanced_Coding_Efficiency_L1	37
Advance_Core_Profile_L2	38
Advance_Core_Profile_L1	39
Advanced_Scalable_Texture_L3	40
Advanced_Scalable_Texture_L2	41
Advanced_Scalable_Texture_L1	42
Unspecified	254
None	255

GraphicsProfileLevelIndication	
Simple2D_L1	1
Simple_2D_Text_L1	2
Simple_2D_Text_L2	3
Core_2D_L1	4
Core_2D_L2	5
Advanced_2D_L1	6

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Advanced_2D_L2	7
Unspecified	254
None	255

9.2.8.4 <ES_Descriptor>

```
<ES_Descriptor
  ES_ID=""
  binaryID=""
  streamPriority=""
  dependsOn_ES_ID=""
  OCR_ES_ID="">
  <URL URLstring=""/>
  <StreamSource>...</StreamSource>
  <decConfigDescr>...</decConfigDescr>
  <slConfigDescr>...</slConfigDescr>
  <ipiPtr>...</ipiPtr>
  <ipIDS>...</ipIDS>
  <ipmpDescrPtr>...</ipmpDescrPtr>
  <langDescr>...</langDescr>
  <qosDescr>...</qosDescr>
  <regDescr>...</regDescr>
  <extDescr>...</extDescr>
</ES_Descriptor>
```

9.2.8.4.1 Description

The <ES_Descriptor> element is the XMT-A representation of the ES_Descriptor as described in subclause 7.2.6.5, ISO/IEC 14496-1. In addition a <StreamSource> element is included to identify the source media to be used for the stream. See 9.2.12 for further information on associating source data with elementary streams.

The <URL> element contains the URLstring and if present the descriptor will be binary encoded with the URL_Flag bit set to 1 and the URLlength attribute set to the length of the URLstring.

The <streamDependence> contains the attribute dependsOn_ES_ID and if the element is present then the binary encoding will have the streamDependenceFlag field bit set to 1 with the value of the dependsOn_ES_ID coded.

The <OCRstream> contains the attribute OCR_ES_ID and if the element is present then the binary encoding will have the OCRstreamFlag field bit set to 1 with the value of the OCR_ES_ID coded.

9.2.8.5 <DecoderConfigDescriptor>

```
<DecoderConfigDescriptor
  objectTypeIndication="0"
  streamType=""
  upstream="false"
  bufferSizeDB="0"
  maxBitrate="0"
  avgBitrate="0">
  <decSpecificInfo>...</decSpecificInfo>
  <profileLevelIndicationIndexDescr>...</profileLevelIndicationIndexDescr>
</DecoderConfigDescriptor>
```

9.2.8.5.1 Description

The <DecoderConfigDescriptor> element is the XMT-A representation of the DecoderConfigDescriptor as described in subclause 7.2.6.6, ISO/IEC 14496-1.

The attribute objectTypeIndication can take either a numeric value from 0 to 255 inclusive or a string corresponding to the enumerated values as follows:

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

objectTypeIndication	
MPEG4Systems1	1
MPEG4Systems2	2
MPEG4Visual	32
MPEG4Audio	64
MPEG2VisualSimple	96
MPEG2VisualMain	97
MPEG2VisualSNR	98
MPEG2VisualSpatial	99
MPEG2VisualHigh	100
MPEG2Visual422	101
MPEG2AudioMain	102
MPEG2AudioLowComplexity	103
MPEG2AudioScaleableSamplingRate	104
MPEG2AudioPart3	105
MPEG1Visual	106
MPEG1Audio	107
JPEG	108
Unspecified	255

The attribute streamType can take either a numeric value from 0 to 63 inclusive or a string corresponding to the enumerated values as follows:

streamType	
ObjectDescriptor	1
ClockReference	2
SceneDescription	3
Visual	4
Audio	5
MPEG7	6
IPMP	7
OCI	8
MPEGJ	9

The attributes bufferSizeDB, maxBitrate and avgBitrate may directly specify values or allow values to be automatically supplied when set to the value “auto”. In the latter case the XMT-A shall determine appropriate values based on the media for that Elementary stream.

9.2.8.6 Decoder Specific Info

The decoder specific information constitutes an opaque container with information for a specific media decoder. In most cases the decoder specific info for a media stream can automatically be created as part of the encoding process, or may already exist in the correct format as a header information with the media data or as a separate entity.

In addition to a the explicit forms of DecoderSpecificInfo below, for example <BIFSConfig>, the following form is a generic DecoderSpecificInfo element that allows all other decoder specific info descriptors, including custom user private forms, to be encoded.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.8.6.1 <DecoderSpecificInfo>

```
<DecoderSpecificInfo
  type="auto" | "xml"
  src="">
  <param name="" value="">
  <param name="" value="">
  …
<DecoderSpecificInfo/>
```

9.2.8.6.1.1 Description

The <DecoderSpecificInfo> element is the XMT-A representation of a generic form of DecoderSpecificInfo as described in subclause 7.2.6.7, ISO/IEC 14496-1.

The attribute type="" allows either the DecoderSpecificInfo to be automatically generated by the XMT-A encoding tool(s) type="auto", or allows values to be given via xml, type="xml", in one of two ways using src="" or param name/value pairs.. Using src="" allows an external source containing information for the DecoderSpecificInfo to be identified. Using a list of name value pairs allows individual values to be given. If both src and name/value pairs are given the name/value pair will override any equivalent value that can be obtained from the src.

In order to allow the specification and easy interchange of binary data, the following two standard name/value pairs are defined:

name = "hexBytes", value is a list of hexadecimal byte value, e.g. "56;7A;0;25"

name = "bytes", value is a list of mixed decimal and hexadecimal byte values, e.g. "25; 0x56; 7; 0xA". Where all hexadeximal bytes must be prefixed with 0x.

In general it is expected that type="auto" will most commonly be specified and so no further standardized set of name/value pairs has been defined.

As an example, for a media stream that conveys a Jpeg image the required DecoderSpecificInfo is the JPEGDecoderConfig as described in subclause 7.2.6.7.1, ISO/IEC 14496-1. XMT-A will create the required info when a <DecoderSpecificInfo type="auto"/> element is placed within the <DecoderConfigDescriptor> <decSpecificInfo> element. The objectTypeIndication therein and streamType will guide the XMT-A to generate the correct DecoderSpecificInfo.

9.2.8.6.2 <BIFSConfig> and <BIFSv2Config>

```
<BIFSConfig
  nodeIDbits="10"
  routeIDbits="10">
  <commandStream
    pixelMetric="true|false" >
    <size
      pixelWidth="0"
      pixelHeight="0" />
  </commandStream>
  <AnimationMask
    randomAccess="true|false">
    ...
  </AnimationMask>
</BIFSConfig>
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
<BIFSv2Config
  use3DmeshCoding="false|true"
  usePredictiveMFField="false|true"
  nodeIDbits="10"
  routeIDbits="10"
  protoIDbits="10">
  <commandStream
    pixelMetric="true|false" >
    <size
      pixelWidth="0"
      pixelHeight="0" />
  </commandStream>
  <AnimationMask
    randomAccess="true|false">
    ...
  </AnimationMask>
</BIFSv2Config>
```

9.2.8.6.2.1 Description

The <BIFSConfig> and <BIFSv2Config> elements are the XMT-A representation of the BIFSConfig and BIFSv2Config as described in 8.5.

The attributes nodeIDbits, routeIDbits and protoIDbits may either be explicitly coded, in which case the number of bits specified must be used to encode the respective IDs, or any of these attributes may be given the value "auto". In the case "auto" is specified the XMT-A tool shall determine the number of bits that is sufficient to accommodate the number of IDs required and set the attribute value accordingly.

The <BIFSConfig> and <BIFSv2Config> elements can contain either a <commandStream> or an <AnimationMask>.

The <commandStream> contains the attribute pixelMetric and if the element is present then the binary encoding will have the isCommandStream field bit set to 1. The <size> contains the attributes pixelWidth and pixelHeight and if the element is present then the binary encoding will have the hasSize field bit set to 1.

The <AnimationMask> described a BIFS-Anim stream that animates the scene. The "randomAccess" attribute is false by default.

9.2.8.6.3 OCI Decoder Configuration

This is the decoder specific info for OCI streams.

```
<DecoderSpecificInfo>
  <param name="versionLabel" value="1" />
</DecoderSpecificInfo>
```

9.2.8.6.3.1 Description

This is the XMT-A representation of the OCIDecoderConfiguration as described in subclause 7.2.4.2.4, ISO/IEC 14496-1. The name/value pair may be either be explicitly coded as above or the OCI Decoder Configuration will also be created if type="auto" is specified instead.

9.2.8.6.4 <AFXConfigType>

```
<complexType name="AFXConfigType">
 <choice>
     <element name="A3DMCDecoderSpecific"/>
     <element name="CoordInterpCompDecoderSpecific"/>
     <element name="OriInterpCompDecoderSpecific"/>
     <element name="PosInterpCompDecoderSpecific"/>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    <element ref="xmta:MeshGridDecoderSpecific"/>
    <element ref="xmta:WMDecoderSpecific"/>
    <element name="OctreeImageDecoderSpecific"/>
    <element name="BBADecoderSpecific"/>
    <element name="PointTextureCompDecoderSpecific"/>
  </choice>
</complexType>
```

9.2.8.6.4.1 Description

AFXConfigType is the decoder specific information which is needed to decode 3DMC, Interpolator Compression, and MPEG-4 AFX related bitsreams in the BitWrapper node.

9.2.8.7 <SLConfigDescriptor>

```
<SLConfigDescriptor>
  <predefined value="0">
  <custom
    useAccessUnitStartFlag="true | false"
    useAccessUnitEndFlag="true | false"
    useRandomAccessPointFlag="true | false"
    hasRandomAccessUnitsOnlyFlag="true | false"
    usePaddingFlag="true | false"
    useIdleFlag="true | false"
    timeStampResolution=""
    OCRResolution=""
    timeStampLength=""
    OCRLength=""
    AU_Length=""
    instantBitrateLength=""
    degradationPriorityLength=""
    AU_seqNumLength=""
    packetSeqNumLength="16">
    <duration timescale="0"
              accessUnitDuration="0"
              compositionUnitDuration="0"/>
    <noUseTimeStamps startDecodingTimeStamp="0"
                     startCompositionTimeStamp="0"/>
  </custom>
</SLConfigDescriptor>
```

9.2.8.7.1 Description

The <SLConfigDescriptor> element is the XMT-A representation of the SLConfigDescriptor as described in subclause 7.3.2.3, ISO/IEC 14496-1.

9.2.8.8 <ContentIdentificationDescriptor>

```
<ContentIdentificationDescriptor
  protectedContent="false">
  <contentType contentType=""/>
  <contentIdentifier
    contentIdentifierType=""
    contentIdentifier=""/>
</ContentIdentificationDescriptor>
```

9.2.8.8.1 Description

The < ContentIdentificationDescriptor > element is the XMT-A representation of the ContentIdentificationDescriptor as described in subclause 7.2.6.10, ISO/IEC 14496-1.

The <contentType> contains the attribute contentType and if the element is present then the binary encoding will have the contentTypeFlag field bit set to 1.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The <contentIdentifier> contains the attributes contentIdentifierType and contentIdentifier, and if the element is present then the binary encoding will have the contentIdentifierFlag field bit set to 1.

The attribute contentType can take either a numeric value from 0 to 255 inclusive or a string corresponding to the enumerated values as follows:

ContentType	
Audio-visual	0
Book	1
Serial	2
Text	3
Item	4
Contribution	4
SheetMusic	5
SoundRecording	6
MusicVideo	6
StillPicture	7
MusicalWork	8
Others	255

The attribute contentIdentifierType can take either a numeric value from 0 to 255 inclusive or a string corresponding to the enumerated values as follows:

contentIdentifierType	
ISAN	0
ISBN	1
ISSN	2
SICI	3
BICI	4
ISMN	5
ISRC	6
ISWC-T	7
ISWC-L	8
SPIFF	9
DOI	10

9.2.8.9 <SupplementaryContentIndentificationDescriptor>

```
<SupplementaryContentIdentification
  languageCode="”
  supplContentIdentiferTitle = ""
  supplContentIdentiferValue = ""/>
```

9.2.8.9.1 Description

The <SupplementaryContentIdentification> element is the XMT-A representation of the SupplementaryContentIdentification as described in subclause 7.2.6.11, ISO/IEC 14496-1.

The languageCode attribute is a three character language code according to specification ISO 639-2:1998.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The supplContentIdentifierTitle attribute is a string (when encoded to binary the required descriptor field supplContentIdentifierTitleLength will be automatically derived from the attribute's string length).

The supplContentIdentifierValue attribute is a string (when encoded to binary the required descriptor field supplContentIdentifierValueLength will be automatically derived from the attribute's string length).

9.2.8.10 <IPI_DescrPointer>

```
<IPI_DescrPointer IPI_ES_Id="”/>
```

9.2.8.10.1 Description

The <IPI_DescrPointer> element is the XMT-A representation of the IPI_DescrPointer as described in Subclause 7.2.6.12, ISO/IEC 14496-1.

The attribute IPI_ES_ID is an IDREF to the ES_ID of the elementary stream whose ES_Descriptor contains the IP Information valid for this elementary stream.

9.2.8.11 <IPMP_DescriptorPointer>

```
<IPMP_DescriptorPointer IPMP_DescriptorID="”/>
```

9.2.8.11.1 Description

The <IPMP_DescriptorPointer> element is the XMT-A representation of the IPMP_DescriptorPointer as described in subclause 7.2.6.13, ISO/IEC 14496-1.

The attribute IPMP_DescriptorID is an IDREF to the IPMP_Descriptor.

9.2.8.12 <IPMP_Descriptor>

```
<IPMP_Descriptor
  IPMP_DescriptorID="0”
  binaryID=””>
  <IPMPS_URL  URLString=””/>
  <IPMPS_data type=”” IPMP_data=””/>
</IPMP_Descriptor>
```

9.2.8.12.1 Description

The intellectual property management and protection (IPMP) framework for ISO/IEC 14496 content consists of a normative interface that permits an ISO/IEC 14496 terminal to host one or more IPMP Systems. The IPMP interface consists of IPMP elementary streams and IPMP descriptors. IPMP descriptors may also be carried as part of an object descriptor stream.

The IPMP System itself is a non-normative component that provides intellectual property management and protection functions for the terminal. The IPMP System uses the information carried by the IPMP elementary streams and descriptors to make protected ISO/IEC 14496 content available to the terminal. An application may choose not to use an IPMP System, thereby offering no management and protection features.

The <IPMP_Descriptor> element is the XMT-A representation of the IPMP_Descriptor as described in subclause 2.6.14, ISO/IEC 14496-1.

The <IPMPS_URL> element contains the attribute URLString, and if the element is present then the binary encoding will have the IPMPS_Type field set to 0.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The <IPMPS_data> element contains the attributes type and IPMP_data, and if the element is present then the binary encoding will have the IPMPS_Type field set to the value of the type attribute.

9.2.8.13 <QoS_Descriptor>

```
<QoS_Descriptor>
 <predefined value="1">
   <qualifiers>...</qualifiers>
 </QoS_Descriptor>
```

9.2.8.13.1 Description

The QoS descriptor conveys the requirements that the elementary stream has on the transport channel and a description of the traffic that this elementary stream will generate.

The <QoS_Descriptor> element is the XMT-A representation of the QoS_Descriptor as described in subclause 7.2.6.15, ISO/IEC 14496-1.

The <predefined> element contains the attribute value, and if the element is present then the binary encoding will have the predefined field set to the value if the value attribute. This value can be from 1 to 255 inclusive.

The <qualifiers> element contains a list of custom QoS Qualifiers if a predefined is not used. If the element is present then the binary encoding will automatically have the predefined field set 0.

9.2.8.13.2 QoS Qualifiers

```
<QoSMaxDelay     value="0"/>
<QoSPrefMaxDelay value="0"/>
<QoSLossProb     value="0"/>
<QoSMaxGapLoss   value="0"/>
<QoSMaxAUSize    value="0"/>
<QoSAvgAUSize    value="0"/>
<QoSMaxAURate    value="0"/>
```

The QoS Qualifier elements above are the XMT-A representations of the QoS_Qualifiers as described in subclause 7.2.6.15.3, ISO/IEC 14496-1.

```
<QoSCustom  tag="" customData="0"/>
```

The QoS Qualifier elements QoSCustom is the XMT-A representation of a QoS_Qualifier as described in subclause 7.2.6.15.3, ISO/IEC 14496-1 where the tag can be set to values to create user private qualifiers.

```
<QoSRebufferingRatio  value="0"/>
```

The QoS Qualifier element above is the XMT-A representations of the QoS_Qualifier REBUFFERING_RATIO as described in subclause 7.2.6.15, ISO/IEC 14496-1.

9.2.8.14 <ExtensionDescriptor>

```
<ExtensionDescriptor tag="" customData=""/>
```

9.2.8.14.1 Description

This is a general purpose element to define a custom extension descriptor.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The <ExtensionDescriptor> element is the XMT-A representation of the ExtensionDescriptor as described in subclause 7.2.6.16, ISO/IEC 14496-1.

9.2.8.15 <RegistrationDescriptor>

```
<RegistrationDescriptor
  formatIdentifier=""
  additionalIdentificationInfo=""/>
```

9.2.8.15.1 Description

The registration descriptor provides a method to uniquely and unambiguously identify formats of private data streams.

The <RegistrationDescriptor> element is the XMT-A representation of the RegistrationDescriptor as described in subclause 7.2.6.17, ISO/IEC 14496-1.

9.2.8.16 Object Content Information Descriptors

Object content information (OCI) descriptors convey descriptive information about audio-visual objects. The main content descriptors are: content classification descriptors, keyword descriptors, rating descriptors, language descriptors, textual descriptors, and descriptors about the creation of the content. OCI descriptors can be included directly in the related object descriptor or elementary stream descriptor or, if it is time variant, it may be carried in an elementary stream by itself.

9.2.8.16.1 <ContentClassificationDescriptor>

```
<ContentClassificationDescriptor
  classificationEntity=""
  classificationTable=""
  classificationData=""/>
```

9.2.8.16.1.1 Description

The <ContentClassificationDescriptor> element is the XMT-A representation of the ContentClassificationDescriptor as described in subclause 7.2.6.18.3, ISO/IEC 14496-1.

9.2.8.16.2 <KeyWordDescriptor>

```
<KeyWordDescriptor
  languageCode=""
  isUTF8="true|false"
  <keyWord value=""/>
</KeyWordDescriptor>
```

9.2.8.16.2.1 Description

The <KeyWordDescriptor> element is the XMT-A representation of the KeyWordDescriptor as described in subclause 7.2.6.18.4, ISO/IEC 14496-1.

The descriptor can contain either 0 to 255 keywords. When the isUTF8 boolean is set true the keywords are coded as UTF8 and the isUTF8_string field in the binary coding is set to 1; otherwise the UTF16 coding is used. For each keyword the keyWordLength field will be binary coded automatically according to the encoded string length. Also the keyWordCount field will be binary coded automatically according to the number of keywords present.

9.2.8.16.3 <RatingDescriptor>

```
<RatingDescriptor
   ratingEntity=""
   ratingCriteria=""
   ratingInfo=""/>
```


Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

9.2.8.16.3.1 Description

The <RatingDescriptor> element is the XMT-A representation of the RatingDescriptor as described in subclause 7.2.6.18.5, ISO/IEC 14496-1.

9.2.8.16.4 <LanguageDescriptor>

```
<LanguageDescriptor languageCode = “”/>
```

9.2.8.16.4.1 Description

The <LanguageDescriptor> element is the XMT-A representation of the LanguageDescriptor as described in subclause 7.2.6.18.6, ISO/IEC 14496-1.

9.2.8.16.5 <ShortTextualDescriptor>

```
<ShortTextualDescriptor
  languageCode=””
  isUTF8=”true|false”>
  <event name=”” text=””/>
</ShortTextualDescriptor>
```

9.2.8.16.5.1 Description

The <ShortTextualDescriptor> element is the XMT-A representation of the ShortTextualDescriptor as described in Subclause 7.2.6.18.7, ISO/IEC 14496-1.

The descriptor contains an event name/text pair. When the isUTF8 boolean is set true eventName (from name attrubute) and eventText (from text attribute) are coded as UTF8 and the isUTF8_string field in the binary coding is set to 1; otherwise the UTF16 coding is used. The nameLength and textLength fields will be binary coded automatically according to the encoded string lengths.

9.2.8.16.6 <ExpandedTextualDescriptor>

```
<ExpandedTextualDescriptor
  languageCode=””
  isUTF8=”true|false”
  nonItemText = “”>
  <item description=“” text=””/>
</ExpandedTextualDescriptor>
```

9.2.8.16.6.1 Description

The <ExpandedTextualDescriptor> element is the XMT-A representation of the ExpandedTextualDescriptor as described in subclause 7.2.6.18.8, ISO/IEC 14496-1.

The descriptor can contain either 0 to 255 items. When the isUTF8 boolean is set true the UTF8 coding is used and the isUTF8_string field in the binary coding is set to 1; otherwise the UTF16 coding is used. For each itemDescription and itemText the itemDescriptionLength and itemLength fields will be binary coded automatically according to the encoded string lengths. Also the itemCount field will be binary coded automatically according to the number of items present. The nonItemText is also coded according to isUTF8 coding and the nonItemTextLength and textLength fields are binary coded automatically according to the respective content lengths.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.8.16.7 <ContentCreatorNameDescriptor>

```
<ContentCreatorNameDescriptor>
  <Creator
    languageCode=“”
    isUTF8=”true|false”
    name=“”/>
</ContentCreatorNameDescriptor>
```

9.2.8.16.7.1 Description

The <ContentCreatorNameDescriptor> element is the XMT-A representation of the ContentCreatorNameDescriptor as described in subclause 7.2.6.18.9, ISO/IEC 14496-1.

The descriptor can contain either 0 to 255 Creator elements. When the isUTF8 boolean is set true the name, in the contentCreatorName field, is coded as UTF8 and the isUTF8_string field in the binary coding is set to 1; otherwise the UTF16 coding is used. For each name the contentCreatorLength field will be binary coded automatically according to the encoded string length. Also the contentCreatorCount field will be binary coded automatically according to the number of creators present.

9.2.8.16.8 <ContentCreationDateDescriptor>

```
<ContentCreationDateDescriptor contentCreationDate=“”/>
```

9.2.8.16.8.1 Description

The <ContentCreatorNameDescriptor> element is the XMT-A representation of the ContentCreatorNameDescriptor as described in subclause 7.2.6.18.10, ISO/IEC 14496-1.

9.2.8.16.9 <OCICreatorNameDescriptor>

```
<OCICreatorNameDescriptor>
  <Creator
    languageCode=“”
    isUTF8=”true|false”
    name=“”/>
</OCICreatorNameDescriptor>
```

9.2.8.16.9.1 Description

The <OCICreatorNameDescriptor> element is the XMT-A representation of the OCICreatorNameDescriptor as described in subclause 7.2.6.18.11, ISO/IEC 14496-1.

The descriptor can contain either 0 to 255 Creator elements. When the isUTF8 boolean is set true the name, in the OCICreatorName field, is coded as UTF8 and the isUTF8_string field in the binary coding is set to 1; otherwise the UTF16 coding is used. For each name the OCICreatorLength field will be binary coded automatically according to the encoded string length. Also the OCICreatorCount field will be binary coded automatically according to the number of creators present.

9.2.8.16.10 <OCICreationDateDescriptor>

```
<OCICreationDateDescriptor OCICreationDate=“”/>
```

9.2.8.16.10.1 Description

The <OCICreationDateDescriptor> element is the XMT-A representation of the OCICreationDateDescriptor as described in subclause 7.2.6.18.12, ISO/IEC 14496-1.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.8.16.11 <SMPTECameraPositionDescriptor>

```
<SMPTECameraPositionDescriptor
  cameraID="">
  <parameter id="" value=""/>
</SMPTECameraPositionDescriptor>
```

9.2.8.16.11.1 Description

The <SMPTECameraPositionDescriptor> element is the XMT-A representation of the SMPTECameraPositionDescriptor as described in subclause 7.2.6.18.13, ISO/IEC 14496-1.

The descriptor can contain either 0 to 255 parameter elements. For each parameter an id (parameterID field) and parameter value (parameter field) will be binary coded automatically from the attributes. The parameterCount field will be binary coded automatically according to the number of parameters present.

9.2.8.17 <ExtensionProfileLevelDescriptor>

```
<ExtensionProfileLevelDescriptor
   profileLevelIndicationIndex=""
   ODProfileLevelIndication=""
   sceneProfileLevelIndication=""
   audioProfileLevelIndication=""
   visualProfileLevelIndication=""
   graphicsProfileLevelIndication=""
   MPEGJProfileLevelIndication=""/>
```

9.2.8.17.1 Description

The <ExtensionProfileLevelDescriptor> element is the XMT-A representation of the ExtensionProfileLevelDescriptor as described in subclause 7.2.6.19, ISO/IEC 14496-1.

For the attributes ODProfileLevelIndication, sceneProfileLevelIndication, audioProfileLevelIndication, visualProfileLevelIndication and graphicsProfileLevelIndication see 9.2.8.3.1.

The attribute MPEGJProfileLevelIndication takes either a numeric value from 0 to 255 inclusive or a string corresponding to the enumerated values as follows:

MPEGJProfileLevelIndication	
Personal_L1	1
Main_L1	2
Unspecified	254
None	255

9.2.8.18 <SegmentDescriptor>

```
<SegmentDescriptor start=""
  duration=""
  segmentName=""
/>
```

9.2.8.19 Object Descriptor Commands

The following describes the object descriptor commands:

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.8.19.1 <ObjectDescriptorUpdate>

```
<ObjectDescriptorUpdate atES_ID="">
   <OD>...</OD>
</ObjectDescriptorUpdate>
```

9.2.8.19.1.1 Description

The <ObjectDescriptorUpdate> element is the XMT-A representation of the ObjectDescriptorUpdate as described in subclause 7.2.5.5.2, ISO/IEC 14496-1.

9.2.8.19.2 <ObjectDescriptorRemove>

```
<ObjectDescriptorRemove objectDescriptorId=""/>
```

9.2.8.19.2.1 Description

The <ObjectDescriptorRemove> element is the XMT-A representation of the ObjectDescriptorRemove as described in subclause 7.2.5.5.3, ISO/IEC 14496-1.

9.2.8.19.3 <ObjectDescriptorExecute>

```
<ObjectDescriptorExecute objectDescriptorId=""/>
```

9.2.8.19.3.1 Description

The < ObjectDescriptorExecute > element is the XMT-A representation of the ES_DescriptorUpdate as described in subclause 7.2.5.5.8, ISO/IEC 14496-1.

9.2.8.19.4 <ES_DescriptorUpdate>

```
<ES_DescriptorUpdate atES_ID="" objectDescriptorId="">
  <esDescr>...</esDecsr>
</ES_DescriptorUpdate>
```

9.2.8.19.4.1 Description

The <ES_DescriptorUpdate> element is the XMT-A representation of the ES_DescriptorUpdate as described in subclause 7.2.5.5.4, ISO/IEC 14496-1.

9.2.8.19.5 <ES_DescriptorRemove>

```
<ES_DescriptorRemove atES_ID="" objectDescriptorId=""/>
```

9.2.8.19.5.1 Description

The <ES_DescriptorRemove> element is the XMT-A representation of the ES_DescriptorRemove as described in subclause 7.2.5.5.5, ISO/IEC 14496-1.

9.2.8.19.6 <IPMP_DescriptorUpdate>

```
<IPMP_DescriptorUpdate atES_ID="">
  <ipmpDescr>...</ipmpDecsr>
</IPMP_DescriptorUpdate>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.8.19.6.1 Description

The <IPMP_DescriptorUpdate> element is the XMT-A representation of the IPMP_DescriptorUpdate as described in subclause 7.2.5.5.6, ISO/IEC 14496-1.

9.2.8.19.7 <IPMP_DescriptorRemove>

```
<IPMP_DescriptorRemove atES_ID="" IPMP_DescriptorID=""/>
```

9.2.8.19.7.1 Description

The <IPMP_DescriptorRemove> element is the XMT-A representation of the IPMP_DescriptorRemove as described in subclause 7.2.5.5.7, ISO/IEC 14496-1.

9.2.9 XMT-A IPMP Streams

An IPMP stream is an elementary stream that passes time-varying information to one or more IPMP Systems. This is accomplished by periodically sending a sequence of IPMP messages along with the content at a period determined by the IPMP system.

```
<IPMP_Message atES_ID="">
  <IPMPS_URL  URLString=""/>
  <IPMPS_data type="" IPMP_data=""/>
</IPMP_Message>
```

9.2.9.1 Description

The <IPMP_Message> element is the XMT-A representation of the IPMP_Message as described in subclause 7.2.3.2.5, ISO/IEC 14496-1.

9.2.10 XMT-A OCI Streams

An OCI stream is an elementary stream that conveys time-varying object content information organized in a sequence of small, synchronized entities called OCI events that contain a set of OCI descriptors.

```
<OCI_Event atES_ID=""
  eventID=""
  absoluteTimeFlag="true|false"
  startingTime=""
  duration="">
  <OCIDescr>...</OCIDescr>
</OCI_Event>
```

9.2.10.1 Description

The <OCI_Event> element is the XMT-A representation of the OCI_Event as described in subclause 7.2.4.2.5, ISO/IEC 14496-1.

9.2.11 XMT-A MPEG-J Streams

An MPEG-J stream is an elementary stream that conveys the Java class files for and MPEG-4 MPEGlet. The source attribute references an external class, object, packaged file or other resource that comprises an MPEGlet or resources that can be loaded by the MPEGlet.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
<JavaStreamHeader atES_ID=""
  url=""
  version="0"
  isClassFlag="true|false"
  isPackaged="true|false"
  compressionScheme=""
  <classID ID="">
  <reqClassID><classID ID="">...</reqClassID>
</JavaStreamHeader>
```

9.2.11.1 Description

The <JavaStreamHeader> element is the XMT-A representation of the JavaStreamHeader as described in subclause 10.3.3.2.

The url attribute is a uriReference to a class or ZIP package etc that form the body of the access unit that the JavaStreamHeader prefaces. When the XMT-A encoder creates the access unit it will do so from the binary encoding of the stream header followed by the content of the url.

9.2.12 XMT-A Elementary Stream Data

Visual, audio, and MPEG-7 elementary stream data are referenced and associated with the Elementary Stream Descriptors using the <StreamSource> element. The <StreamSource> element is contained within the Elementary Stream Descriptor element.

Some elementary streams, such as BIFS and OD have textual representations with XMT-A that is used to create the streams via an encoding process. Other media sources, such as video and audio, have no textual representation of the media itself. However a variety of video or audio sources etc can be referenced and these sources do not necessarily have to be in the final target format for the stream. Conversion (transcoding or encoding) from source to target formats may be supported by the XMT-A authoring tool; basic support requires only that a tool can handle media in the correct target format. Transcoding and encoding externally referenced media etc is optional and may be authoring tool and platform dependent.

For elementary streams that are represented textually within the XMT-A document a <StreamSource> may still be used within the <ESDescriptor> to supply <EncodingHints>, for example for BIFS. For this case the url attribute would be omitted.

9.2.12.1 <StreamSource>

```
<StreamSource
  url="">
  <EncodingHints>...</EncodingHints>
</StreamSource>
```

url refers to the external source for the media data. Whether it be pre-encoded or requires encoding. The <EncodingHints> allow the source and/or target formats to be described so that when encoding is required the author can control various parameters of the encoding process where supported by the encoder.

<EncodingHints> cannot override any explicit values provided in <DecoderConfigDescriptor> including those in any <DecoderSpecificInfo> specified therein. An encoder shall ensure that the encoded stream source complies with the objectTypeIndication, streamType, bufferSizeDB etc. Some parameters however may be set to “auto”, eg bufferSizeDB, maxBitrate etc. The values used for these auto fields, when the descriptors are binary coded, will then be based on the results of the encoding process according to the <EncodingHints>. The entire <DecoderSpecificInfo> may also be automatically coded when specified using type=”auto”. Using auto allows XMT-A to encode any generate the necessary decoder configuration parameters without the author needing to explicitly provide these values.

9.2.12.2 Object Descriptor Streams

Object Descriptor commands are coded as timed elements and associated to a particular Elementary Stream Descriptor using the atES_ID attribute in the various commands.

By default atES_ID will take on the value of the ID of the first Elementary Stream Descriptor, in document order, contained in the Initial Object Descriptor that contains a DecoderConfigDescriptor of stream type ObjectDescriptorStream. Hence for

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

MPEG-4 content with a single ODStream the atES_ID will default as expected and minimize explicit coding. Any addtional ODStreams would require the commands to be explicitly coded for respective atES_ID to which the command should be associated.

9.2.12.3 Clock Reference Streams

No <StreamSource> is required. A Clock Reference stream is defined using the SLConfigDescriptor in the ElementaryStreamDescriptor that contains a DecoderConfigDescriptor of stream type CleockReferenceStream.

9.2.12.4 Scene Description Streams

BIFS commands are coded as timed elements and associated to a particular Elementary Stream Descriptor using the atES_ID attribute in the various commands .

By default atES_ID will take on the value of the ID of the first Elementary Stream Descriptor, in document order, contained in the Initial Object Descriptor that that contains a DecoderConfigDescriptor of stream type SceneDescriptionStream. Hence for MPEG-4 content with a single BIFS Stream the atES_ID will default as expected and minimize explicit coding. Any additional BIFS Streams, e.g. if there were a scaleable BIFS representation, would require the commands to be explicitly coded for the respective atES_ID to which the command should be associated.

9.2.12.5 IPMP Streams

<StreamSource> will refer to IPMP media data and an Elementary Stream Descriptor of type IPMPStream.

Alternatively IPMP message can be coded as timed elements and associated to a particular Elementary Stream Descriptor using the atES_ID attribute in the <IPMPMessage>.

9.2.12.6 Visual Streams

<StreamSource> will refer to a visual media and an Elementary Stream Descriptor that contains a DecoderConfigDescriptor of stream type VisualStream.

9.2.12.7 Audio Streams

<StreamSource> will refer to an audio media and an Elementary Stream Descriptor that contains a DecoderConfigDescriptor of stream type AudioStream.

9.2.12.8 MPEG-7 Streams

<StreamSource> will refer to MPEG-7 media data and an Elementary Stream Descriptor that contains a DecoderConfigDescriptor of stream type MPEG7Stream.

9.2.12.9 IPMP Streams

IPMP messages are coded as timed elements and associated to a particular Elementary Stream Descriptor, that contains a DecoderConfigDescriptor of stream type IPMPStream, using the atES_ID attribute in the <IPMPMessage>.

9.2.12.10 OCI Streams

OCI events are coded as timed elements and associated to a particular Elementary Stream Descriptor, that contains a DecoderConfigDescriptor of stream type ObjectContentInfoStream, using the atES_ID attribute in the <OCIEvent>.

9.2.12.11 MPEG-J Streams

MPEG-J streams are coded as timed elements, using the <JavaStreamHeader> element, and associated to a particular Elementary Stream Descriptor, that contains a DecoderConfigDescriptor of stream type MPEGJStream, using the atES_ID attribute in the <JavaStreamHeader>.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.13 XMT-A Deterministic mapping

In encoding XMT-A into binary some features may be encoded differently producing alternate binary representations that are all legally valid. In part to allow the author control where needed over mapping, and in part to allow a deterministic mapping for conformance the following 'devices' allow us to achieve this.

Elements are to maintained in document order, unless they are elements timed by a <par> element in which case they should be sorted into temporal order while maintaining the document order of any elements occurring at the same time. For independently coded streams this temporal sorting can be done on a per stream basis.

Following the sorting in step 1 any elements now are to be encoded in temporal order with any nested complex elements occurring at any given time to be encoded strictly in the order that they appear in the XMT document being converted to mp4. This is to ensure:

Elements to be coded at the same time in a single AU will thus be in the same order.

Multiple commands, or updates of fields will hence be in the same order.

Scene elements will then be in the same order under groups.

For attributes we cannot expect an XML parser to maintain the list of attributes in document order. So for this case, where it is needed, there are encoding hints on how to map attributes. For example, in BIFS this is needed as there are options for list or vector codings etc; whereas for the OD framework the fields are coded in a predetermined order as so it is not required. However, multi-value single attributes must preserve the order of the multiple values in the coding.

Elements are identified in the XMT by an id (a name). The name is convenient for the XMT as references can be made back to a human readable, meaningful name. These ids must often be coded into NodeIDs or ObjectDescriptor IDs etc. Normally the XMT authoring tool can create binary numeric IDs for all the names and ensure their uniqueness. To ensure a given conversion to a specific binary value the binaryID attribute has been added to elements whose names must be converted to binary. The authoring tool must respect the binary conversion wherever it is so explicitly given.

Some elements such as BIFS <Insert> can insert field values defined by attributes plus it can contain <ROUTE> commands to insert routes. For any elements such as these the command(s) contained in the attributes must be coded before any child elements.

Multiple commands, specified by a multi-value single attribute must be coded in the order that the multiple values occur.

Conditional SFCommandBuffer fields shall be coded using smallest possible number of padding bits (no extra bytes at the end). Also the length field shall be specified using the smallest number of bits that can accommodate the length, i.e minimize lengthBits.

9.2.14 XMT-A Animation

9.2.14.1 Overview

BIFS-Anim is a binary format used in MPEG-4 systems to transmit animation of objects in a scene. Each animated node is referred by its DEF identifier and one or many of its fields may be animated. BIFS-Anim is a key frame technique that specifies the value of each animated field frame by frame, at a defined frame rate. For better compression, each field value is quantized and adaptively arithmetic encoded.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

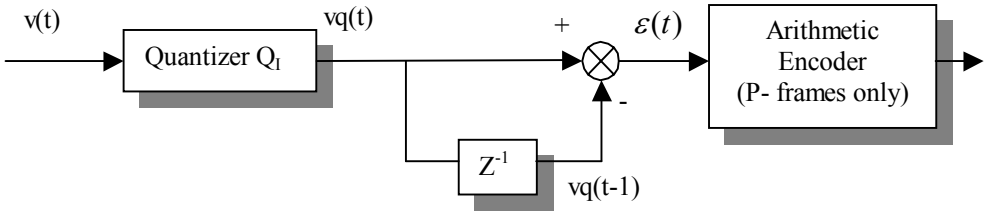


Figure 59 — BIFS Anim encoding process. Z^{-1} stands for one frame delay. The value *v(t)* at frame *t* is quantized with Quantizer Q_I and the difference $\varepsilon(t)$ with the previous quantized value *vq(t-1)* at frame *t-1* is adaptively arithmetic encoded.

Two kinds of frames are available: Intra and Predictive frames. I-frames contain raw quantized field values *vq(t)*, and P-frames contain arithmetically encoded difference field values $\varepsilon(t)=vq(t)-vq(t-1)$.

A BIFS-Anim frame can animate values by in different nodes nodes, i.e. each frame will specify the nodes that it animates, and will then animate all the fields of the specified nodes. As BIFS-Anim is a key-frame based system, a frame can be only I or P, consequently all field values in an animated node must be I or P coded, and each field of a given node is animated at the same frame rate.

This textual format is designed to allow authors to specify what can be encoded in a bitstream. It is allows specification of every encoding hint possible in BIFS-Anim.

There are two top level elements used in BIFS-Anim. The first is <AnimationMask>, which appears inside a <BIFSConfig> element and contains initializing information. The second is an <AnimationFrame> element, which must appear inside of a <par> element and holds the animation frame data.

9.2.14.2 <AnimationMask>

```
<AnimationMask
  randomAccess = "true|false">
  <ElementaryMask id="">
  </ElementaryMask>
  …
  <ElementaryMask ….> … </ElementaryMask>
</AnimationMask>
```

9.2.14.2.1 Description

The <AnimationMask> element is the XMT-A representation of the AnimationMask as described in subclause 8.5.4.

The randomAccess attribute is described in the BIFSConfig and BIFSv2Config as described in subclauses 8.5.2 and 8.5.3.

The <AnimationMask> tag holds a collection of one or more <ElementaryMask> tags.

9.2.14.3 <ElementaryMask>

```
<ElementaryMask atNode="">
    <InitialFieldsMask> … </InitialFieldsMask>
    <InitialFieldsMask> … </InitialFieldsMask>
     …
    <InitialFieldsMask> … </InitialFieldsMask>
</ElementaryMask>
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.14.3.1 Description

The <ElementaryMask> element is the XMT-A representation of the ElementaryMask as described in subclause 8.5.5.

The <ElementaryMask> tag holds a collection of <InitialFieldsMask> tags.

The atNode attribute holds the ID of the animated node.

9.2.14.4 <InitialFieldsMask>

```
<InitialFieldsMask
    id=""
   isTotal="true|false"
   indexList = "…"                // optional
   >
  <InitialAnimQP .. />
</InitialFieldsMask>
```

9.2.14.4.1 Description

The <InitialFieldsMask> element is the XMT-A representation of the InitialFieldsMask as described in subclause 8.5.6.

An <InitialFieldsMask> element shall contain exactly one <InitialAnimQP/> tag.

The id attribute shall hold the name of the field to be animated. The attributes isTotal and indexList concern only MFFields. The isTotal attribute shall be false by default. If isTotal is set to true, the indexList attribute shall be ignored. The indexList attribute holds a list of index value.

9.2.14.5 <initialAnimQP>

```
<InitialAnimQP
   type  = "1..13"
   useDefault = "true|false"              // optional
   useLowerBoundEfficientCoding = "…"     // optional
   useUpperBoundEfficientCoding = "…"     // optional
   lowerBound = "…"
   upperBound = "…"
   IminInt = "…"                  // optional, used only for type=13
   InbBits = "…"
   Pmin = "…"
   PnbBits = "…" />
```

9.2.14.5.1 Description

The <InitialAnimQP> element is the XMT-A representation of the InitialAnimQP as described in subclause 8.5.7.

The userLowerBoundEfficientCoding and userUpperBoundEfficientCoding are a comma, separated array of Boolean values with length that's equal to the getNbBounds() for the animated field. The default values for these fields is false.

An example of its use :

```
<AnimationMask>
  <ElementaryMask id = "COLOR0">
    <InitialFieldsMask id="color"
       indexList = "1 3 10"> <!--this MFField has more than 10 values-->
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        <InitialAnimQP
           type = "4"
           lowerBound = "0 0 0"
           upperBound = "1 1 1"
           useUpperBoundEfficientFloat = "true true true"
           useLowerBoundEfficientFloat = "true true true"
           InbBits = "8"
           PnbBits = "9"
           Pmin = "-256 -256 -256"/>
      </InitialFieldsMask>
   </ElementaryMask>
   <ElementaryMask id = "TX1">    <!-- a transform node -->
      <InitialFieldsMask id = "translation">  <!-- an SFVec3f -->
         <InitialAnimQP
           type  = "1"
           lowerBound = "-10.0 -10.0 -10.0"
           upperBound = "10.0 5.0 5.0"
           useUpperBoundEfficientFloat = "true true true"
           useLowerBoundEfficientFloat = "true true true"
           InbBits = "8"
           PnbBits = "9"
           Pmin = "-256 -256 -256"/>
      </ InitialFieldsMask>
   </ ElementaryMask>
</AnimationMask>
```

9.2.14.6 <AnimationFrame>

```
<AnimationFrame atES_ID=""
       animationStartCode = ""         // optional
       isIntra = "true|false"
       TimeCode = ""                  // optional
       SkipFrames = ""                    // optional
       >
       <FrameRate .. />                   // optional
       <AnimationFrameData> .. </AnimationFrameData>
    …

       <AnimationFrameData> .. </AnimationFrameData>
</AnimationFrame>
```

9.2.14.6.1 Description

The <AnimationFrame> element is the XMT-A representation of the AnimationFrame as described in subclause 8.8. It also contains the data skipFrames.

The skipFrames attribute holds an integer specifying the number of frames to skip.

The <AnimationFrame> element holds a collection of zero or more <AnimationFrameData> elements.

The atES_ID attribute indicates the destination of the AnimationFrame. It specifies a symbol which is the ID of the elementary stream into which the AnimationFrame should be encoded. In the case of conflicting atES_ID values in parent par elements, this attribute takes precedence.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.14.7 <FrameRate>

```
<FrameRate
       frameRate = "[0..255]"
       seconds  = "[0..15]"
       frequencyOffset  = "[0..1]"
       />
```

9.2.14.7.1 Description

The <FrameRate> element is the XMT-A representation of the FrameRate as described in subclause 8.8.

9.2.14.8 <AnimationFrameData>

```
<AnimationFrameData
       maskID = ""         // optional in place of id
       atNode = "">            // optional in place of maskID
       <AnimationField> … </AnimationField>
       …
       <AnimationField> … </AnimationField>
</AnimationFrameData>
```

9.2.14.8.1 Description

The <AnimationFrameData> element is the XMT-A representation of the functionality of the AnimationFrameData as described in subclause 8.8. However, in XMT-A, each <AnimationFrameData> describes data for just one animated node, where as in subclause 8.8, a mechanism is described to specify all the animated nodes at one.

An <AnimationFrameData> defines a node which is to be animated within a particular <AnimationFrame>.

The maskID or atNode attributes can optionally be used to specify animated node, but one must be specified. The nodes defined in the <ElementaryMask> element shall be referenced in masked using their order (starting with index 1). The animated nodes may alternatively be referenced using their DEF name.

The <AnimationFrameData> contains a sequence of zero or more <AnimationField> elements.

9.2.14.9 <AnimationField>

```
<AnimationField
       maskID = ""     // optional in place of id
       id = ""         // optional in place of maskID
       values = "">
       <animQP …./>        // optional
</ AnimationField>
```

9.2.14.9.1 Description

The <AnimationField> element is the XMT-A representation of the functionality of the AnimationField as described in subclause 8.8.

Inside an <AnimationFrameData>, all the fields specified in the <InitialFieldsMask> will be animated. For fields that don't need to be animated at the time of the current <AnimationFrame>, the previous value will have to be encoded. Thus all the animated fields don't have to be specified, but the missing ones will be encoded with their previous values.

The id attribute is the name of the animated field. As the case for <AnimationFrameData>, fields may be identified in the maskID attribute using their order (starting with index 1) in the list of animated field of the current node.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

An optional <animQP> element can be specified, but only for animationFrames set as Intra Frames.

Example:

```
<par begin="10.0s">
 <animationFrame isIntra = "true">
    <AnimationFrameData id = "TX1">
       <AnimationField maskID = "1" values = " 1.0 2.0 3.3"> # SFVec3f
        <animQP …/>
       </ AnimationField>
      </ AnimationFrameData>
  </animationFrame>
</par>
<par begin="10.5s">
   <animationFrame isIntra = "false">
      <AnimationFrameData maskID = "1"> # this is node COLOR0
       <AnimationField
        maskID = "1"
        values = " 1.0 1.0 0.9  0.5 0.5 0.5  0.0 0.0 1.0"> # MFColor
       </ AnimationField>
      </ AnimationFrameData>
      <AnimationFrameData maskID="2"> # this is node TX1
        <AnimationField
        maskID = "1"
            values = "1.1 2.1 3.4">
        </ AnimationField>
      </ AnimationFrameData>
   </animationFrame>
</par>
```

At time 10.0, only one of the nodes will be animated, but two nodes will be animated at time 10.5.

At time 10.5 we animate an MFField with 3 animated Indices. All the values of the animated indices are required.

9.2.14.10 <AnimQP>

```
   <AnimQP
      type  = "1..13"
      useDefault = "true|false"                          // optional
      useLowerBoundEfficientCoding = "true|false"          // optional
      useUpperBoundEfficientCoding = "true|false"          // optional
      lowerBound = ""                              // optional
      upperBound = ""                              // optional
      IminInt = ""                                 // optional
      InbBits = ""                                 // optional
      Pmin = ""                                    // optional
      PnbBits = ""                                 // optional
      />
```

9.2.14.10.1 Description

The <AnimQP> element is the XMT-A representation of the functionality of the AnimQP as described in subclause 8.8.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.14.10.2 FBA Animation

Face and Body animation parameters shall be placed in separate XML files. These files shall be referenced in the relevant ES_Descriptors with StreamSource elements.

9.2.15 Predictive MF Coding

The XMT-A <Predictive> element allows representation of advanced coding of multiple value fields.

9.2.15.1 <Predictive>

```
<Predictive>
       <PredictiveMFField name="">...</PredictiveMFField >
       <PredictiveMFField name="">...</PredictiveMFField>
    ...
</Predictive>
```

9.2.15.1.1 Description

The <Predictive> element is a container for the <PredictiveMFField> element, which can appear once for each numerical MFField in the node.

9.2.15.2 <PredictiveMFField>

```
<PredictiveMFField name="">
   <ArrayHeader>...</ArrayHeader>
   <ArrayOfValues>...</ArrayOfValues>
</PredictiveMFField >
```

9.2.15.2.1 Description

The < PredictiveMFField> element is the XMT-A representation of the PredictiveMFField as described in subclause 8.7.10. The name attribute specifies the name of the field whose values are represented. The <ArrayHeader>, and <ArrayOfValues> elements are described below.

9.2.15.3 <ArrayHeader>

```
<ArrayHeader NbBits  = "0..31" IntraMode = "0..2">
   <InitialArrayQP/>
</ArrayHeader>
```

9.2.15.3.1.1 Description

The <ArrayHeader> element is the XMT-A representation of the ArrayHeader as described in subclause 8.7.11. There is no need to specify a numberOfFields values as in the binary sequence, since it can be deduced from the field values. It contains an <InitialArrayQP> which holds the initial quantization parameters for the encoding.

9.2.15.4 <InitialArrayQP>

```
<InitialArrayQP
    NbBits  = "0..31" intraInterval=""     // coded only if IntraMode==1
    CompNbBits  = "0..31"
    vq= "" Pmin="" />                        // arrays of length getNbComp()
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.15.4.1 Description

The <InitialArrayQP> element is the XMT-A representation of the InitialArrayQP as described in subclause 8.7.13. The NbBits and intraInterval values shall appear only when the IntraMode of the enclosing <ArrayHeader> has value 1. The vq and Pmin values hold arrays of length getNbComp(), which is the number of components for the field that is being encoded.

9.2.15.5 <ArrayOfValues>

```
<ArrayOfValues IPPolicy  = "" arrayQPFlag="">
        <ArrayQP/>                  // optional ArrayQPs
        <ArrayQP/> . . .
        <ArrayQP/>
<ArrayOfValues/>
```

9.2.15.5.1 Description

The <ArrayOfValues> element is the XMT-A representation of some of the functionality in ArrayOfValues, as described in subclause 8.7.14. However, in XMT-A, the field values are not stored in this element; they are stored in the field element just as if predictive coding were not sured. The <ArrayOfValues> element is used to specify Intra and Predictive frames when the IntraMode of the <ArrayHeader> has value 2. This element is also used to specify when new Quantization parameters are sent in the stream using an <ArrayQP>

The IPPolicy attribute holds an integer array with values corresponding to the indices of the Intra frames in the stream. The arrayQPFlag is a numerical array with length equal to the number of Intra frames in the encoding. It specifies which Intra frames carry new quantization parameters.Note that the number of Intraframes can be depenedent on the value of IPPolicy (when IntraMode is 2), or it may depend on the intraInterval in the <InitialArrayQP>, when the IntraMode is 1. If IntraMode is 0, both the IPPolicy and arrayQPFlag attributes shall be ignored.

For each value of arrayQPFlag that is 1, there shall be a corresponding <ArrayQP> element that conveys a new set of quantization parameters for the encoding.

9.2.15.6 <ArrayQP>

```
<ArrayQP
      NbBits = "0..31" intraInterval=""         // coded only if IntraMode==1
      CompNbBits  = "0..31"
      vq= "" Pmin="" />                 // optional arrays
```

9.2.15.6.1 Description

The <ArrayQP> element is the XMT-A representation of the ArrayQP as described in subclause 8.7.13. The NbBits and intraInterval values shall appear only when the IntraMode of the enclosing <ArrayHeader> has value 1. The vq and Pmin values are optional arrays that hold arrays of length getNbComp(), which is the number of components for the field that is being encoded. The difference between the <ArrayQP> and <InitialArrayQP> is that the latter requires specification of CompNbBits, vq, and Pmin.

Example :

```
      <ScalarInterpolator
            key="0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0"
            keyValue="0 0.12 0.23 0.34 0.45 0.56 0.67 0.78 0.89 0.90 1.00">
        <Predictive>
          <PredictiveMFField name="key">
            <ArrayHeader NbBits=4 IntraMode=0>
              <InitialArrayQP CompNbBits=10 vq="0" Pmin="0">
            </ArrayHeader>
         </PredictiveMFField>
         <PredictiveMFField name="keyValue">
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            <ArrayHeader NbBits=4 IntraMode=2>
              <InitialArrayQP CompNbBits=10 vq=”0” Pmin=”0”>
            </ArrayHeader>
            <ArrayOfValues IPPolicy=”0 4 10” arrayQPFlag=”1,0”>
              <ArrayQP CompNbBits=14>
            </ArrayOfValues>
          </PredictiveMFField>
        </Predictive>
      </ScalarInterpolator>
```

9.2.16 XMT-A Carriage of node compressed information

9.2.16.1 Overview

A node may have a dedicated node compression scheme. This compressed representation may be carried in the BIFS stream or in a separate stream. The functionality of BitWrapper node is to encode a node data and transmit such an encoded bitstream through a scene description stream (BIFS stream) or a separate stream.

During the authoring stage, if an author wants to encode a node data and transmit it using the BitWrapper node, the author should have abilities to control encoding parameters for generating an encoded bitstream. The following textual format is designed to allow authors to specify what node can be encoded and how its encoding parameters can be controlled in generating an encoded bitstream.

9.2.16.2 <BitWrapper>

```
<complexType name="BitWrapperType">
       <all>
            <element ref="xmta:IS" minOccurs="0"/>
            <element name="node" form="qualified" minOccurs="0">
                  <complexType>
                        <group ref="xmta:SFWorldNodeType" minOccurs="0" />
                  </complexType>
            </element>
            <element ref="xmta:BitWrapperEncodingParameter"/>
       </all>
       <attribute name="type" type="xmta:SFInt32" use="optional" default="0"/>
       <attribute name="url" type="xmta:MFUrl" use="optional"/>
       <attribute name="buffer" type="xmta:SFString" use="optional"
default="&amp;quot;&amp;quot;"/>
       <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="BitWrapper" type="xmta:BitWrapperType"/>

<element name="BitWrapperEncodingParameter">
    <complexType>
       <choice>
          <element name="CoordinateInterpolatorEncodingParameter" minOccurs="0"
       maxOccurs="1">
            <complexType>
              <attribute name="keyQBits" type="xmta:numOfKeyQBits" use="optional"
            default="8"/>
              <attribute name="keyValueQBits" type="xmta:numOfKeyValueQBits"
              use="optional" default="16"/>
              <attribute name="transpose" type="xmta:transposeType" use="optional"
              default="&amp;quot;ON&amp;quot;"/>
              <attribute name="linearKeycoder" type="xmta:linearKeycoderType"
              use="optional" default="&amp;quot;LINEAR&amp;quot;"/>
             </complexType>
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
</element>
<element name="IndexedFaceSetEncodingParameter" minOccurs="0" maxOccurs="1">
  <complexType>
    <attribute name="coordQBits" type="xmta:numOfCoordQBits" use="optional"
default="10"/>
    <attribute name="normalQBits" type="xmta:numOfNormalQBits" use="optional"
default="9"/>
    <attribute name="colorQBits" type="xmta:numOfColorQBits" use="optional"
default="6"/>
    <attribute name="texCoordQBits" type="xmta:numOftexCoordQBits"
use="optional" default="10"/>
    <attribute name="coordPredMode" type="xmta:coordPredType" use="optional"
    default="2"/>
    <attribute name="normalPredMode" type="xmta:normalPredType" use="optional"
    default="0"/>
    <attribute name="colorPredMode" type="xmta:colorPredType" use="optional"
    default="0"/>
    <attribute name="texCoordPredMode" type="xmta:texCoordPredType"
    use="optional" default="0"/>
    <attribute name="errorResilience" type="xmta:errorResilienceType"
    use="optional" default="&quot;OFF&quot;"/>
    <attribute name="bitsPerPacket" type="xmta:SFInt32" use="optional"
    default="180"/>
    <attribute name="boundaryPrediction" type="xmta:boundaryPredictionType"
    use="optional" default="0"/>
  </complexType>
</element>
<element name="OrientationInterpolatorEncodingParameter" minOccurs="0"
maxOccurs="1">
  <complexType>
    <attribute name="keyQBits" type="xmta:numOfKeyQBits" use="optional"
    default="8"/>
    <attribute name="keyValueQBits" type="xmta:numOfKeyValueQBits"
    use="optional" default="16"/>
    <attribute name="preservingMode" type="xmta:preservingType" use="optional"
default="&quot;KEY&quot;"/>
    <attribute name="dpcmMode" type="xmta:orientationDpcmType" use="optional"
    default="0"/>
    <attribute name="aacMode_X" type="xmta:aacType" use="optional"
    default="&quot;BINARY&quot;"/>
    <attribute name="aacMode_Y" type="xmta:aacType" use="optional"
    default="&quot;BINARY&quot;"/>
    <attribute name="aacMode_Z" type="xmta:aacType" use="optional"
    default="&quot;BINARY&quot;"/>
    <attribute name="linearKeycoder" type="xmta:linearKeycoderType"
    use="optional" default="&quot;LINEAR&quot;"/>
  </complexType>
</element>
<element name="PositionInterpolatorEncodingParameter" minOccurs="0"
maxOccurs="1">
  <complexType>
    <attribute name="keyQBits" type="xmta:numOfKeyQBits" use="optional"
    default="8"/>
    <attribute name="keyValueQBits" type="xmta:numOfKeyValueQBits"
    use="optional" default="16"/>
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    <attribute name="preservingMode" type="xmta:preservingType" use="optional"
    default="&amp;quot;KEY&amp;quot;"/>
    <attribute name="dpcmMode_X" type="xmta:positionDpcmType" use="optional"
    default="0"/>
    <attribute name="dpcmMode_Y" type="xmta:positionDpcmType" use="optional"
    default="0"/>
    <attribute name="dpcmMode_Z" type="xmta:positionDpcmType" use="optional"
    default="0"/>
    <attribute name="aacMode_X" type="xmta:aacType" use="optional"
    default="&amp;quot;BINARY&amp;quot;"/>
    <attribute name="aacMode_Y" type="xmta:aacType" use="optional"
    default="&amp;quot;BINARY&amp;quot;"/>
    <attribute name="aacMode_Z" type="xmta:aacType" use="optional"
    default="&amp;quot;BINARY&amp;quot;"/>
    <attribute name="linearKeycoder" type="xmta:linearKeycoderType"
    use="optional"
    default="&amp;quot;LINEAR&amp;quot;"/>
    <attribute name="intra_X" type="xmta:intraType" use="optional"
    default="0"/>
    <attribute name="intra_Y" type="xmta:intraType" use="optional"
    default="0"/>
    <attribute name="intra_Z" type="xmta:intraType" use="optional"
    default="0"/>
  </complexType>
</element>
<element name="MeshGridEncodingParameter" minOccurs="0" maxOccurs="1">
  <complexType>
    <attribute name="nLevels" type="xmta:dim3u32" use="optional" />
    <attribute name="nSlices" type="xmta:dim3u32" use="optional" />
    <attribute name="sizeROI" type="xmta:dim3u32" use="optional" />
    <attribute name="modeROI" type="xmta:modeROIType" use="optional"
    default="&amp;quot;uniformDensity&amp;quot;"/>
    <attribute name="groupROI" type="xmta:dim3u32" use="optional" default="1 1
    1"/>
    <attribute name="hasConnectivityInfo" type="xmta:boolean" use="optional" />
    <attribute name="hasRefineInfo" type="xmta:boolean" use="optional" />
    <attribute name="hasRepositionInfo" type="xmta:boolean" use="optional" />
    <attribute name="hasGridInfo" type="xmta:boolean" use="optional" />
    <attribute name="meshType" type="xmta:meshType" use="optional" />
    <attribute name="sameBorderOrientation" type="xmta:boolean" use="optional"
    />
    <attribute name="uniformSplit" type="xmta:boolean" use="optional" />
    <attribute name="offsetAmplitude" type="xmta:bit32" use="optional" />
    <attribute name="cyclicMode" type="xmta:cyclicModeType" use="optional" />
    <attribute name="fullRefine" type="xmta:boolean" use="optional" />
    <attribute name="nRefineBits" type="xmta:bit5" use="optional" default="0"/>
    <attribute name="filterType" type="xmta:filterType" use="optional" />
    <attribute name="nQuantBits" type="xmta:bit5" use="required"/>
    <attribute name="gridCR" type="xmta:bit32" use="optional" default="1"/>
    <attribute name="maxError" type="xmta:SFFloat" use="optional" default="0"/>
  </complexType>
</element>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            <element name="WaveletSubdivisionSurfaceEncodingParameter" minOccurs="0"
            maxOccurs="1">
              <complexType>
                <attribute name="NbBpSC" type="xmta:bit5" use="optional" default="8"/>
                <attribute name="NbBPX" type="xmta:bit5" use="optional" default="14"/>
                <attribute name="NbBPY" type="xmta: bit5" use="optional" default="14"/>
                <attribute name="NbBPZ" type="xmta:bit5" use="optional" default="14"/>
                <attribute name="Wtype" type="xmta:bit2" use="optional" default="0"/>
                <attribute name="lift" type="xmta:bit1" use="optional" default="0"/>
                <attribute name="isInLocalCoordinates" type="xmta:bit1" use="optional"
                default="0"/>
                <attribute name="LengthNbBits" type="xmta:bit4" use="optional"
                default="15"/>
                <attribute name="isPartial" type="xmta:bit1" use="optional" default="0"/>
                <attribute name="NumberOfLevels" type="xmta:bit5" use="optional"
                default="4"/>
              </complexType>
            </element>
            <element name="PointTextureEncodingParameter" minOccurs="0" maxOccurs="1">
              <complexType>
                <attribute name="codingPercent" type="xmta:codingPercentType"
            use="optional" default="100"/>
              </complexType>
            </element>
          </choice>
        </complexType>
</element>
```

9.2.16.2.1 Description

The <BitWrapper> element is the XMT-A representation of the BitWrapper node as described in subclause 7.2.2.23. It also contains the XMT-A representation of the encoding parameters that can be used in generating the encoded bitstream of a specified node data defined in **node** element.

The **CoordinateInterpolatorEncodingParameter** element contains the encoding parameters used for encoding CoordinateInterpolator node data, in the case that **node** element contains CoordinateInterpolator node.

The **IndexedFaceSetEncodingParameter** element contains the encoding parameters used for enconding IndexedFaceSet node data using 3D Mesh Coding (3DMC), in the case that **node** element contains IndexedFaceSet node.

The **OrientationInterpolatorEncodingParameter** element contains the encoding parameters used for encoding OrientationInterpolator node data, in the case that **node** element contains OrientationInterpolator node.

The **PositionInterpolatorEncodingParameter** element contains the encoding parameters used for encoding PositionInterpolator node data, in the case that **node** element contains PositionInterpolator node.

The **PointTextureEncodingParameter** element contains the encoding parameter used for encoding PointTexture node data, in the case that **node** element contains PointTexture node.

The **MeshGridEncodingParameter** element contains the encoding parameter used for encoding MeshGrid node data, in the case that **node** element contains MeshGrid node.

The **WaveletSubdivisionSurfaceEncodingParameter** element contains the encoding parameter used for encoding WaveletSubdivisionSurface node data, in the case that **node** element contains WaveletSubdivisionSurface node.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

These respective encoding parameter elements should be exclusively used when the encoding is performed. Because the **node** element should contain one type of node data to be encoded at a time and should not contain more types at once.

When authors use the XMT-A representation of the BitWrapper element in order to transmit an encoded bitstream, there are two types of usage. One is to use the encoding parameter element and the other is not to use it. When using encoding parameters, authors can encode the original data (uncompressed one) that are contained in the **node** element of BitWrapper element. On the other hand, without using them, authors can directly specify the file name (or object descriptor ID) of an encoded bitstream in **url** or **buffer** attribute of BitWrapper element in the case that authors already have an encoded bitstream.

The attributes in each encoding parameter element are described in the following clauses.

9.2.16.3 <numOfKeyQBits>

```
<simpleType name="numOfKeyQBits">
   <restriction base="int">
      <minInclusive value="0"/>
      <maxInclusive value="31"/>
   </restriction>
</simpleType>
```

9.2.16.3.1 Description

The numOfKeyQBits indicates the quantization bit size of the key data. It is an integer type. The minimum value of numOfKeyQBits is 0 and the maximum is 31.

9.2.16.4 <numOfKeyValueQBits>

```
<simpleType name="numOfKeyValueQBits">
   <restriction base="int">
      <minInclusive value="0"/>
      <maxInclusive value="31"/>
   </restriction>
</simpleType>
```

9.2.16.4.1 Description

The numOfKeyValueQBits indicates the quantization bit size of the keyValue data. It is an integer type. The minimum value of numOfKeyValueQBits is 0 and the maximum is 31.

9.2.16.5 <transposeType>

```
<simpleType name="transposeType">
   <restriction base="string">
      <enumeration value="&quot;ON&quot;"/>
      <enumeration value="&quot;OFF&quot;"/>
   </restriction>
</simpleType>
```

9.2.16.5.1 Description

The transposeType is the flag for transpose mode or vertex mode. If the value is set to ON, the transpose mode is used. Otherwise, the vertex mode is used.

9.2.16.6 <linearKeycoderType>

```
<simpleType name="linearKeycoderType">
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   <restriction base="string">
      <enumeration value="&quot;LINEAR&quot;"/>
      <enumeration value="&quot;NOLINEAR&quot;"/>
   </restriction>
</simpleType>
```

9.2.16.6.1 Description

The linearKeycoderType is a string type. It indicates if the linear key coder is used or not.

9.2.16.7 <numOfCoordQBits>

```
<simpleType name="numOfCoordQBits">
   <restriction base="int">
      <minInclusive value="1"/>
      <maxInclusive value="24"/>
   </restriction>
</simpleType>
```

9.2.16.7.1 Description

The numOfCoordQBits indicates the quantization step used for geometry. The minimum value of numOfCoordQBits is 1 and the maximum is 24.

9.2.16.8 <numOfNormalQBits>

```
<simpleType name="numOfNormalQBits">
   <restriction base="int">
      <minInclusive value="3"/>
      <maxInclusive value="31"/>
   </restriction>
</simpleType>
```

9.2.16.8.1 Description

The numOfNormalQBits indicates the quantization step used for normals. The minimum value of numOfNormalQBits is 3 and the maximum is 31.

9.2.16.9 <numOfColorQBits>

```
<simpleType name="numOfColorQBits">
   <restriction base="int">
      <minInclusive value="1"/>
      <maxInclusive value="16"/>
   </restriction>
</simpleType>
```

9.2.16.9.1 Description

The numOfColorQBits indicates the quantization step used for colors. The minimum value of numOfColorQBits is 1 and the maximum is 16.

9.2.16.10 <numOfTexCoordQBits>

```
<simpleType name="numOftexCoordQBits">
   <restriction base="int">
      <minInclusive value="1"/>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      <maxInclusive value="16"/>
   </restriction>
</simpleType>
```

9.2.16.10.1 Description

The numOftexCoordQBits indicates the quantization step used for texture coordinates. The minimum value of NumOftexCoordQBits is 1 and the maximum is 16.

9.2.16.11 <coordPredType>

```
<simpleType name="coordPredType">
   <restriction base="int">
      <enumeration value="0"/>
      <enumeration value="2"/>
   </restriction>
</simpleType>
```

9.2.16.11.1 Description

The coordPredType is the type of prediction used to reconstruct the vertex coordinates of the mesh. The value is set to 0 if the no_prediction is used and 2 if the parallelogram_prediction is used.

9.2.16.12 <normalPredType>

```
<simpleType name="normalPredType">
   <restriction base="int">
      <enumeration value="0"/>
      <enumeration value="1"/>
      <enumeration value="2"/>
   </restriction>
</simpleType>
```

9.2.16.12.1 Description

The normalPredType indicates how normal values are predicted. The value is set to 0 if the no_prediction is used and 1 if the tree_prediction is used, and 2 if the parallelogram_prediction is used.

9.2.16.13 <colorPredType>

```
<simpleType name="colorPredType">
   <restriction base="int">
      <enumeration value="0"/>
      <enumeration value="1"/>
      <enumeration value="2"/>
   </restriction>
</simpleType>
```

9.2.16.13.1 Description

The colorPredType indicates how colors are predicted. The value is set to 0 if the no_prediction is used, 1 if the tree_prediction is used and 2 if the parallelogram_prediction is used.

9.2.16.14 <texCoordPredType>

```
<simpleType name="texCoordPredType">
   <restriction base="int">
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      <enumeration value="0"/>
      <enumeration value="2"/>
   </restriction>
</simpleType>
```

9.2.16.14.1 Description

The texCoordPredType indicates how colors are predicted. The value is set to 0 if the no_prediction is used and 2 if the parallelogram_prediction is used.

9.2.16.15 <errorResilienceType>

```
<simpleType name="errorResilienceType">
   <restriction base="string">
      <enumeration value="&quot;ON&quot;"/>
      <enumeration value="&quot;OFF&quot;"/>
   </restriction>
</simpleType>
```

9.2.16.15.1 Description

The errorResilienceType indicates the usage of error resilient mode. The value is set to OFF if the error resilience is not used and ON if the error resilience is used. Only if the value is set to ON, both boundaryPredictionType and bitsPerPacket shall be avaliable.

9.2.16.16 <bitsPerPacket>

The Syntax of bitsPerPacket is the same as the type of SFInt32 in the xmt-a schema .

9.2.16.16.1 Description

The bitsPerPacket indicates packet size for error resilient bitstream. This value determines the size of each partition in error resilient mode. The type of bitsPerPacket is SFInt32 .The default value is 360.

9.2.16.17 <boundaryPredictionType>

```
<simpleType name="boundaryPredictionType">
   <restriction base="int">
      <enumeration value="0"/>
      <enumeration value="1"/>
   </restriction>
</simpleType>
```

9.2.16.17.1 Description

The boundaryPredictionType indicates the type of boundary prediction. If the value is 0, the restricted prediction shall be used and if the value is 1, the extended prediction shall be used.

9.2.16.18 <preservingType>

```
<simpleType name="preservingType">
   <restriction base="string">
      <enumeration value="&quot;KEY&quot;"/>
      <enumeration value="&quot;PATH&quot;"/>
   </restriction>
</simpleType>
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.16.18.1 Description

The preservingType is a string type. It indicates if the current mode is key preserving mode or path preserving mode.

9.2.16.19 <orientationDpcmType>

```
<simpleType name="orientationDpcmType">
   <restriction base="int">
      <enumeration value="0"/>
      <enumeration value="1"/>
      <enumeration value="2"/>
   </restriction>
</simpleType>
```

9.2.16.19.1 Description

The orientationDpcmType indicates the order of DPCM used for each keyValue component(X, Y, Z, Theta) in OrientationInterpolator node. It is an integer type. The flags are set to 0 if 1st order DPCM is used, and 1 if 2nd order DPCM is used, and 2 if the orientation interpolator encoder automatically determines the order of DPCM.

9.2.16.20 <aacType>

```
<simpleType name="aacType">
   <restriction base="string">
      <enumeration value="&amp;quot;BINARY&amp;quot;"/>
      <enumeration value="&amp;quot;UNARY&amp;quot;"/>
   </restriction>
</simpleType>
```

9.2.16.20.1 Description

The aacType is a string type. It indicates if the current mode is BinaryAAC mode or UnaryAAC mode for each keyValue component(X, Y, Z, (Theta – OrientationInterpolator)).

9.2.16.21 <positionDpcmType>

```
<simpleType name="positionDpcmType”>
   <restriction base="int">
      <enumeration value="0"/>
      <enumeration value="1"/>
      <enumeration value="2"/>
   </restriction>
</simpleType>
```

9.2.16.21.1 Description

The positionDpcmType indicates the order of DPCM used for each keyValue component(X, Y, Z). It is an integer type. The flags are set to 0 if 1st order DPCM is used, and 1 if 2nd order DPCM is used, and 2 if SAD is used. SAD is an abbreviated word for summary of absolute difference. When sum1 is the sum of 1st order DPCM and sum2 is the sum of 2nd order DPCM, SAD is the method of selecting the minimum value of both sum1 and sum2 and determining the DPCM method having the minimum value.

9.2.16.22 <intraType>

```
<simpleType name="intraType">
   <restriction base="int">
      <enumeration value="0"/>
      <enumeration value="1"/>
   </restriction>
</simpleType>
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.16.22.1 Description

The intraType is used for a Position Interpolator Compression. It indicates if intra coding mode is used or not for each keyValue component(X,Y,Z).

9.2.16.23 <dim3u32>

```
<simpleType name="dim3u32">
   <restriction>
      <simpleType>
         <list itemType="unsignedInt"/>
      </simpleType>
      <length value="3"/>
   </restriction>
</simpleType>
```

9.2.16.23.1 Description

The dim3u32 is an array of 3 unsigned integers.

9.2.16.24 <modeROIType>

```
<simpleType name="modeROIType">
   <restriction base="string">
      <enumeration value="&quot;uniformDensity&quot;"/>
      <enumeration value="&quot;uniformSize&quot;"/>
   </restriction>
</simpleType>
```

9.2.16.24.1 Description

The modeROIType is a string type used for MeshGrid to specify the region of interest (ROI) mode. If uniformDensity mode, at each resolution level of the MeshGrid model the size of the ROI is adapted to have quasi the same amount of information within each ROI. If uniformSize mode, the size of the ROIs is kept the same resolution level as each resolution level of the model.

9.2.16.25 <meshType>

```
<simpleType name="meshType">
   <restriction base="string">
      <enumeration value="&quot;GENERIC_MESH&quot;"/>
      <enumeration value="&quot;TRI_MESH&quot;"/>
      <enumeration value="&quot;QUAD_MESH&quot;"/>
      <enumeration value="&quot;HEXA_MESH&quot;"/>
   </restriction>
</simpleType>
```

9.2.16.25.1 Description

The meshType is a string type used for MeshGrid to specify the type of mesh to encode. If GENERIC_MESH, the mesh may consist at the same time of triangles, quadrilaterals, pentagons, hexagons and heptagons. Otherwise if TRI_MESH, QUAD_MESH or HEXA_MESH the mesh is homogeneous and consists only of one type of polygon. The meaning of the values is described in more detail in ISO/IEC 14496-16:2004.

9.2.16.26 <cyclicModeType>

```
<simpleType name="cyclicModeType">
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   <restriction base="string">
      <enumeration value="&quot;CYCLIC_NONE&quot;"/>
      <enumeration value="&quot;CYCLIC_U&quot;"/>
      <enumeration value="&quot;CYCLIC_V&quot;"/>
      <enumeration value="&quot;CYCLIC_UV&quot;"/>
      <enumeration value="&quot;CYCLIC_W&quot;"/>
      <enumeration value="&quot;CYCLIC_UW&quot;"/>
      <enumeration value="&quot;CYCLIC_VW&quot;"/>
   </restriction>
</simpleType>
```

9.2.16.26.1 Description

The cyclicModeType is a string type used for MeshGrid to specify the cyclic behaviour of the mesh. The value is set to CYCLIC_NONE if the mesh is not cyclic and to one of the other 6 values when the mesh is cyclic. The meaning of the values is described in more detail in ISO/IEC 14496-16:2004.

9.2.16.27 <filterType>

```
<simpleType name="filterType">
   <restriction base="string">
      <enumeration value="&quot;SHORT_FILTER&quot;"/>
      <enumeration value="&quot;SMOOTH_FILTER&quot;"/>
   </restriction>
</simpleType>
```

9.2.16.27.1 Description

The filterType is a string type used for MeshGrid to specify the type of filter coefficients used in the wavelet transform of the reference-grid coordinates. The meaning of the values is described in ISO/IEC 14496-16:2004.

9.2.16.28 <BitWrapperEncodingHints>

```
<element name="BitWrapperEncodingHints">
   <complexType>
      <choice>
         <element name="BitWrapper3DMCEncodingHints">
            <complexType>
               <sequence>
                  <element name="sourceFormat">
                     <complexType>
                        <sequence>
                           <element ref="xmta:param" minOccurs="0" maxOccurs="unbounded"/>
                        </sequence>
                     </complexType>
                  </element>
                  <element name="targetFormat">
                     <complexType>
                        <sequence>
                           <element ref="xmta:param" minOccurs="0" maxOccurs="unbounded"/>
                        </sequence>
                     </complexType>
                  </element>
               </sequence>
            </complexType>
         </element>
         <element name="BitWrapperICEncodingHints">
            <complexType>
               <sequence>
                  <element name="sourceFormat">
                     <complexType>
                        <sequence>
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                  <element ref="xmta:param" minOccurs="0" maxOccurs="unbounded"/>
               </sequence>
            </complexType>
         </element>
         <element name="targetFormat">
            <complexType>
               <sequence>
                  <element ref="xmta:param" minOccurs="0" maxOccurs="unbounded"/>
               </sequence>
            </complexType>
         </element>
      </sequence>
   </complexType>
</element>
<element name="BitWrapperOctreeImageEncodingHints">
   <complexType>
      <sequence>
         <element name="sourceFormat">
            <complexType>
               <sequence>
                  <element ref="xmta:param" minOccurs="0" maxOccurs="unbounded"/>
               </sequence>
            </complexType>
         </element>
         <element name="targetFormat">
            <complexType>
               <sequence>
                  <element ref="xmta:param" minOccurs="0" maxOccurs="unbounded"/>
               </sequence>
            </complexType>
         </element>
      </sequence>
   </complexType>
</element>
<element name="BitWrapperMeshGridEncodingHints">
   <complexType>
      <sequence>
         <element name="sourceFormat">
            <complexType>
               <sequence>
                  <element ref="xmta:param" minOccurs="0" maxOccurs="unbounded"/>
               </sequence>
            </complexType>
         </element>
         <element name="targetFormat">
            <complexType>
               <sequence>
                  <element ref="xmta:param" minOccurs="0" maxOccurs="unbounded"/>
               </sequence>
            </complexType>
         </element>
      </sequence>
   </complexType>
</element>
<element name="OthersEncodingHints">
   <complexType>
      <sequence>
         <element name="sourceFormat">
            <complexType>
               <sequence>
                  <element ref="xmta:param" minOccurs="0" maxOccurs="unbounded"/>
               </sequence>
            </complexType>
         </element>
         <element name="targetFormat">
            <complexType>
               <sequence>
                  <element ref="xmta:param" minOccurs="0" maxOccurs="unbounded"/>
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                  </sequence>
               </complexType>
            </element>
         </sequence>
      </complexType>
   </element>
   <element name="BitWrapperPointTextureEncodingHints">
      <complexType>
         <sequence>
            <element name="sourceFormat">
               <complexType>
                  <sequence>
                     <element ref="xmta:param" minOccurs="0" maxOccurs="unbounded"/>
                  </sequence>
               </complexType>
            </element>
            <element name="targetFormat">
               <complexType>
                  <sequence>
                     <element ref="xmta:param" minOccurs="0" maxOccurs="unbounded"/>
                  </sequence>
               </complexType>
            </element>
         </sequence>
      </complexType>
   </element>
   <element name="BitWrapperWaveletSubdivisionSurfaceEncodingHints">
      <complexType>
         <sequence>
            <element name="sourceFormat">
               <complexType>
                  <sequence>
                     <element ref="xmta:param" minOccurs="0" maxOccurs="unbounded"/>
                  </sequence>
               </complexType>
            </element>
            <element name="targetFormat">
               <complexType>
                  <sequence>
                     <element ref="xmta:param" minOccurs="0" maxOccurs="unbounded"/>
                  </sequence>
               </complexType>
            </element>
         </sequence>
      </complexType>
   </element>
  </choice>
 </complexType>
</element>
```

9.2.16.28.1 Description

The <BitWrapperEncodingHints> element is used for specifying the stream filename and the stream format of an encoded bitstream that is transmitted as a separate bitstream using the BitWrapper element. The stream formats that can be transmitted by BitWrapper element are specified in this document and ISO/IEC 14496-16:2004.

9.3 XMT-Ω Format

The goal of the XMT- Ω format is to provide ease of use, facilitate content interchange and to be interoperable with the Synchronized Multimedia Integration Language (SMIL) 2.0, for which the specification was developed by the W3C SYMM WG.

The XMT-Ω format describes audio-visual objects and their relationships at a higher level than XMT-A. Here content requirements are expressed in terms of an author's intent rather than by coding explicit node and route connections. This permits an authoring tool to offer constructs at this higher level and to exchange them at this level with other authoring

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

tools. An authoring tool would compile the XMT-Ω format into MPEG-4 content by mapping the constructs into BIFS, OD, media streams, etc. together with any appropriate media compression and/or media conversions that may be required. Media sources expressed in XMT-Ω can be of a variety of formats native to the machine where the authoring tool is executing, and it is the responsibility of the tool, during the compilation phase, to convert the media to suitable formats, bit-rates, and so on.

In this high-level format, there is not necessarily only one mapping to MPEG-4 for each construct. MPEG-4 nodes and routes are very powerful tools and there can be more than one way to represent XMT-Ω constructs. Also, as MPEG-4 nodes can be 'wired' together with routes in many combinations, it is often difficult to reverse-engineer an author's intent from a collection nodes and routes. Faced with a presentation containing many nodes and routes, the re-authoring and maintenance of content can be challenging as the high-level view of that presentation must be inferred. The XMT-Ω, however, provides such a high-level view with high-level authoring constructs and thus facilitates content exchange, rapid content re-purposing or re-authoring and maintenance of content.

Recognizing though that some authors may wish to access low-level nodes/routes, this format allows the embedding of the XMT-A textual node and route definitions within an identified low-level nodes section. Interoperation between the two levels are also permitted.

9.3.1 XMT-Ω re-using SMIL

To capture content authors intentions at a high-level, the SMIL (Synchronized Multimedia Integration Language) is used as a basis for the abstraction of the XMT-Ω content representation. The version of SMIL upon which XMT-Ω is based on the SMIL 2.0 specification, (the follow-on to SMIL 1.0), that was developed by the W3C SYMM working group. For brevity, rather than constantly refer to SMIL 2.0 any references will be made simply as "SMIL" in the remainder of this document where there is no ambiguity.

SMIL is an XML-based language that allows authors to write interactive multimedia presentations. The main strengths of SMIL are that its constructs are self-describing, it is based on XML which provides an excellent format for interchange of data among different applications, it is relatively easy to author, and it is a language familiar to HTML users. It is also extensible so that new objects or metadata can easily be inserted in the representation.

The XMT-Ω format provides a new set of elements that expresses the high-level view of MPEG-4, while re-using a subset of modules defined by SMIL where the semantics are compatible. XMT-Ω is not specifically designed as a playback format, and is intended to be preprocessed to SMIL, WEB3D (VRML), or compiled to an MPEG-4 representation via appropriate translation software.

9.3.1.1 Re-using SMIL Modules

The SMIL language is composed of a number of functional areas that have broken down into a finer granularity using modules. These modules, comprising XML element and attributes, and attribute values, can then be combined and brought together in other host languages such as XMT-Ω. XMT-Ω is referred to as a host language since it integrates, or hosts, the modules within a larger set of XML representation.

SMIL provides guidelines and requirements for integration of the modules it defines into a host language. XMT-Ω, unless explicitly stated otherwise, will follow these guidelines and requirements and adhere to the semantics of the modules as well as their syntax. SMIL is a language that has been designed and then implementations follow. XMT-Ω, however, already has MPEG-4 as a implementation and is using this language to represent it, albeit at a high-level. Hence XMT-Ω must be mapped (compiled) into MPEG-4 and certain behaviors specified by SMIL may be difficult, or overly complicated, to maintain the semantics for all cases. This document will highlight such areas as necessary. In all cases it is the authoring tools responsibility to maintain correct semantics during the mapping. All the power of the MPEG-4 representation, including Scripts and MPEG-J, may be used by the authoring tool to achieve satisfactory mappings. An authoring tool will of course limit the use of MPEG-4 tools to those supported by the MPEG-4 profile and level for which the presentation is being created. Mapping for the semantics are both static and dynamic in nature. Static mappings capture the semantics for deterministic behavior that can be fully evaluated during the mapping. Dynamic mappings require runtime support of MPEG-4 player mechanisms and, for example, may be utilized to support non-deterministic behavior such as is involved in unpredictable user interactions.

Subsequent sections of this document will cover XMT-Ω and its use of SMIL modules in more detail. The following is a brief overview of the functional areas of SMIL and their re-use in XMT-Ω :

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Animation.
XMT-Ω will incorporate the SMIL Animation modules. These modules support dynamic updating, i.e. animation of attributes. SMIL semantics require the authored value as well as the animated value of an attribute to be available. MPEG-4 routes overwrite fields making such semantics the authoring tools responsibility to maintain.

Content Control.
Allows content choice and selection based on test attributes. May be mapped to MPEG-4 constructs such as alternate streams, for example selection based on language. Also may be used to select alternate content at compile time.

Layout
XMT-Ω will not incorporate SMIL Layout modules. Layout will however be defined that is consistent with the hierarchical tree structured spatial layout and groupings intrinsic to MPEG-4.

Linking.
XMT-Ω will not incorporate SMIL Linking modules. However support for linking that can be mapped to the Anchor node will be defined.

Media
XMT-Ω incorporates SMIL media modules plus it extends this set to include MPEG-4 specific media.

Metainformation
XMT-Ω incorporates SMIL metainformation modules. In addition XMT-Ω supports MPEG-7 meta representations.

Timing and Synchronization
SMIL contains extensive, comprehensive timing modules. XMT-Ω incorporates most of these modules.

Time Manipulations
XMT-Ω incorporates SMIL Time Manipulations module which permits time transformations.

Transitions
SMIL supports high-level transitions with effects defined by SMPTE as well as effects defined by SMIL . XMT-Ω incorporates some of these transition modules.

9.3.2 XMT-Ω Animation

SMIL Animation feature comprises two modules; BasicAnimation and SplineAnimation. XMT-Ω will include the features of both modules.

All the XMT-Ω Timing and Synchronization module attributes are supported to control the timelines of the elements of these Animation modules. As XMT-Ω incorporates the BasicTimeContainer module from Timing and Synchronization the fill attribute is also supported on all Animation elements.

Attributes of xMedia elements and their child elements, attribute of Media Augmentation elements, and attributes of the Media Group may all be animated.

9.3.2.1 BasicAnimation

BasicAnimation includes the <animate>, <set>, <animateMotion> and <animateColor> elements that provide the means to specify animated behavior.

9.3.2.1.1 <animate>

The <animate> element provides a means to animate an attribute over a list of values or using from, to and by.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
<animate
    id             = ID
    attributeName  = <attributeName>
    attributeType  = "XML" | "auto"
    targetElement  = IDREF
    values         = <list>
    calcMode       = "discrete | linear | paced"
    accumulate     = "sum | none"
    additive       = "sum | replace"
    from           = <value>
    to             = <value>
    by             = <value>
/>
```

id is an XML identifier.

attributeName is the name of the attribute on which the animation is enacted.

attributeType is for compatibility purposes with SMIL only.

targetElement is the identifier of the element containing the attribute to be animated.

values is a semicolon-separated list of one or more legal values with which to animate the attribute.

calcMode specifies the interpolation mode for the animation.

accumulate controls whether the animation is cumulative and each iteration builds upon the last iteration, or whether iterations simply repeat the same set of values.

additive controls whether the effect of the animation adds to any other animations that may currently be active on the attribute, or whether the animation value simply overrides any other lower priority animations. (See SMIL specification for priority determination).

from specifies a legal starting value for the animation if *values* is not specified.

end specifies a legal ending value for the animation if *values* is not specified.

by specifies a legal relative offset value for the animation if *values* is not specified.

9.3.2.1.2 <set>

The <set> element provides a means to set an attribute to a specific of value for a period of time.

```
<set
    id             = ID
    attributeName  = <attributeName>
    attributeType  = "XML" | "auto"
    targetElement  = IDREF
    to             = <value>
/>
```

The attributes of the <set> element are a subset of those in the <animate> element. See <animate> element for more information.

9.3.2.1.3 <animateMotion>

The <animateMotion> element provides a means to animate an element along a path using a list of values or using from, to and by. <animateMotion> has the same attributes as animate except that the attributeName and attributeType for the target attribute are not required. <animateMotion> acts upon predefined attributes of only certain elements.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If the targetElement is <transformation> or <group> the attribute acted upon is translation.

```
<animateMotion
    as per <animate> without attributeName and attributeType, but plus…
    origin = "default"
/>
```

9.3.2.1.4 <animateColor>

The <animateColor> element provides a means to animate a color attribute using a list of values or using from, to and by.

```
<animateColor
    as per <animate>…
/>
```

9.3.2.2 SplineAnimation

The SplineAnimation module adds the following attributes to BasicAnimation.

```
<animate
    as per BasicAnimation plus…
    calcMode   = "spline"
    keyTimes   = <list>
    keySplines = <list>
/>
```

The value **spline** is added to the supported values for calcMode. keyTimes is a semicolon-separated list of time values used to control the pacing of the animation. keySplines is a set of Bezier control points defining a cubic Bezier function, associated with the keyTimes list, that controls interval pacing.

Refer to the SMIL specification for a complete description of the SplineAnimation module. The module is incorporated into XMT-Ω in full conformance with that specification.

9.3.2.3 DragAnimation

The DragAnimation module is unique for XMT and defines interactive animation based on input sensor dragging. DragAnimation elements can be used with visual Xmedia elements. A visual element, when the input sensor is held down over it, can be the source of drag animation. DragAnimation for 2D xMedia elements can be plane or disc oriented allowing planar or rotational drag effects. For 3D the drag animation can be plane, cylinder or sphere allowing planar, rotational and cylindrical, or spherical drag effects.

Timing attributes determine when the begin, end, duration etc of the period during which dragging is enabled. During that period dragging can be effected using the pointing device.

Attributes listed below for the various drag animation elements have semantics that are defined in the respective nodes to which the drag animations will be mapped. Autooffset is common to all drag animations and allows the offset to automatically be set to the last output value when the pointing device button is released. Subseqeunt dragging then takes place from there. By default autooffset is true.

9.3.2.3.1 <dragPlane>

```
<dragPlane
    maxPosition = <value>
    minPosition = <value>
    autoffset   = "true | false"
    offset      = <value>
/>
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2005 – All rights reserved
Not for Resale

maxPosition specifies the maxPosition and has default value of "-1 -1"

minPosition specifies the minPosition and has default value of "0 0"

offset specifies the offset and has a default value of "0 0" for 2D contexts and "0 0 0" in 3D contexts.

9.3.2.3.2 <dragDisc>

```
<dragDisc
    maxAngle  = <angle>
    minAngle  = <angle>
    autoffset = "true | false"
    offset    = <angle>
/>
```

maxAngle specifies the maxAngle and has default value of "–1.0rad".

minAngle specifies the minAngle and has default value of "0".

offset specifies the offset and has a default value of "0".

9.3.2.3.3 <dragCylinder>

```
<dragCylinder
    diskAngle = <angle>
    maxAngle  = <angle>
    minAngle  = <angle>
    autoffset = "true | false"
    offset    = <angle>
/>
```

diskAngle specifies the diskAngle and has default value of "0.262rad".

maxAngle specifies the maxAngle and has default value of "–1.0rad".

minAngle specifies the minAngle and has default value of "0".

offset specifies the offset and has a default value of "0".

9.3.2.3.4 <dragSphere>

```
<dragSphere
    autoffset = "true | false"
    offset    = <angle axis>
/>
```

offset specifies the offset and has a default value of "0rad 0 1 0"

9.3.2.4 Mapping Animation to MPEG-4

XMT-Ω will incorporate the SMIL Animation modules however authoring tools will be made responsible to ensure SMIL Animation semantics are preserved.

Animation features can be mapped to MPEG-4 using various means. TimeSensors, Interpolators and Routes provide mechanisms to do value/keyTime animations. BIFS updates for replace field can be used for <set> and may alos be used for animation if the updates are frequent enough. BIFS Anim may also be used where appropriate. As long as the Animation semantics are maintained an authoring tool implementation is free to choose any appropriate MPEG-4 function(s) to fit the need. It would however be expected to choose a particular mapping to fit the MPEG-4 "tools"

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

according to a given MPEG-4 profile and level. More complex mappings may be achieved with MPEG-4 player functions that can do run-time computation and decisions, such as MPEG-J and Scripts.

The various drag animations can be mapped to DiscSensor, CylinderSensor, PlaneSensor, PlaneSensor2D and SphereSensor.

Unlike SMIL players, MPEG-4 players do not maintain both a base value and a dynamic presentation value for attributes (fields). MPEG-4 supports only field replacement, whether done by Routes, by BIFS command, or by animation streams. To preserve SMIL Animation semantics the XMT authoring tool will be required to manage and track the attributes state such that the semantics can be honored. However non-deterministic behavior, such as from user driven events, will make this task more difficult and may require the authoring tool to create more complex MPEG-4 constructs to manage this.

The following aspects a SMIL Animation will require attention by the authoring tool:

When a field is updated in MPEG-4 the prior value is not retained. When the effect of the animation ends the default behavior in SMIL is to revert back to the authored value. To achieve compatible behavior the authoring tool must add, for example additional BIFS commands, to ensure this happens. To do this the authoring tool must be aware of the authored value and may simulate the playback of the authored presentation so that it can insert a field replacement to put the value back to the authored value. For non-deterministic behavior this may include MPEG-4 Conditional nodes, MPEG-J, or Script nodes to do this that can be activated/run by the player from an appropriate trigger.

Where the XMT authoring environment can predict the former value then the authoring tool should honor SMIL semantics. What degree of complexity the authoring tool uses to map such behavior onto MPEG-4 is left to implementation. Where however such semantics cannot be preserved then the authoring tool should warn the author. In the latter case the tool should also suggest a change in the fill attribute value of the animation element. By changing to fill=”freeze” this will allow the final value from the animation to be kept until the media ends.

To achieve SMIL Animation compatibility the effect of certain attributes such as accumulate, additive, by, to, and certain calcModes e.g. spline and paced may be computed at compile time. Such computed results would then be used to generate say a series of timed BIFS commands, or a set of interpolator values for example that produce the desired effect. The degree of approximation in such computations is authoring tool implementation dependent. Depending on the target MPEG-4 player complexity it is conceivable that Scripts or MPEG-J could be used as an alternative so that computations could be done at playback time.

In MPEG-4 the result of multiple routes being fired simultaneously into a given field – fan-in – is undefined. If there are multiple animation elements acting simultaneously on an attribute (field) the authoring tool may not be able to generate an MPEG-4 presentation that combines them and maintains SMIL Animation semantics. If the resultant animation is predictable at authoring time or the authoring tool can manage whatever uncertainty remains at run-time then the animation may be converted to MPEG-4 to allow predictable behavior. If the authoring tool cannot provide a predictable outcome then a warning to the author should be generated.

Some MPEG-4 nodes do not allow some or all of their fields to be updated e.g. Box and FontStyle. Also Box and Rectangle are very similar nodes but for Box, the size field cannot be updated, whereas for Rectangle node the size field can be updated. To update a Box the whole geometry node would need replacing to change it’s size, whereas for Rectangle just the size field itself can be replaced. An XMT authoring tool must mange this mapping variability to MPEG-4 as XMT-Ω presents a consistent interface allowing both those size attributes to be animated even though there are underlying differences. For the animation element <set> the attribute size can be mapped to a single Field and Node replacement for Rectangle and Box respectively. <animate> can also be mapped the same way, with an appropriate rate of updates to achieve the animation effect. For Rectangle however there is also the choice of using an Interpolator in the scene to animate its size field. If an authoring tool cannot map an animation using the available MPEG-4 tools for the target profile and level then a warning to the author should be generated.

9.3.3 XMT-Ω Content Control

SMIL Animation feature comprises four modules; BasicContentControl, CustomTestAttributes, PrefetchControl and SkipContentControl. XMT-Ω will include the features of all four modules.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.3.3.1 BasicContentControl

SMIL provides a <switch> element and test attributes that permit content selection to be made from a set of alternative elements. Only the first acceptable element is chosen.

9.3.3.1.1 System test attributes

XMT-Ω incorporates the set of system test attributes as listed in the SMIL specification. For example systemBitrate and systemLanguage are test attributes defined by SMIL.

9.3.3.1.2 TermCap test attributes

XMT-Ω adds a set of test attributes that are to be mapped to the TermCap node, subclause 7.2.2.125. The values of the attributes are integers as defined in subclause 7.2.2.125.2. The attributes apply to xMedia objects where the mapping to MPEG will place the media objects under a switch connected to a TermCap node with the capability and its value set according to the termCap test attribute and its value.

termCapFrameRate
termCapColorDepth
termCapScreenSize
termCapGraphicsHardware
termCapAudioOutputFormat
termCapMaxAudioSamplingRate
termCapSpatialAudioCapability
termCapCPULoad
termCapMemoryLoad

The following table lists the above termCap test attributes. Either the numeric value or the string value (case sensitive) may be used.

termCapAttr	val	XMLstring value	Semantics
termCapFrameRate	0	unknown	Unknown or can't determine
	1	0-5fps	Less than 5fps
	2	5-10fps	5-10fps
	3	10-20fps	10-20fps
	4	20-40fps	20-40fps
	5	+40fps	More than4 0fps
termCapColorDepth	0	unknown	Unknown or can't determine
	1	1bpp	1bit/pixel
	2	grayscale	Grayscale
	3	3-12bppColor	Color, 3-12bits per pixel
	4	12-24bppColor	Color, 12-24bits per pixel
	5	+24bppColor	Color, more than24bits per pixel
termCapScreenSize	0	unknown	Unknown or can't determine
	1	0-200lines	Less than 200lines
	2	200-400lines	200-400lines
	3	400-800lines	400-800lines
	4	800-1600lines	800-1600lines
	5	+1600lines	1600 or more lines
termCapGraphicsHardware	0	unknown	Unknown or can't determine
	1	noAcceleration	No acceleration
	2	matrix	Matrix multiplication
	3	matrixTexture0-1M	Matrix multiplication + texture mapping (less than 1M memory)
	4	matrixTexture1-4M	Matrix multiplication + texture mapping (less than 4M memory)
	5	matrixTexture+4M	Matrix multiplication + texture mapping (more than 4M memory)
termCapAudioOutputFormat	0	unknown	Unknown or can't determine

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

	1	mono	Mono
	2	stereoSpeakers	Stereo speakers
	3	stereoHeadphones	Stereo headphones
	4	fiveChannelSurround	five-channel surround
	5	+fiveSpeakers	More than five speakers
termCapMaxAudioSamplingRate	0	unknown	Unknownorcan'tdetermine
	1	0-16000Hz	Less than 16000Hz
	2	16000-32000Hz	16000-32000Hz
	3	32000-44100Hz	32000-44100Hz
	4	44100-48000Hz	44100-48000Hz
	5	+48000Hz	48000Hz or more
termCapSpatialAudioCapability	0	unknown	Unknown or can't determine
	1	noSpatialAudio	No spatial audio
	2	panningOnly	Panning only
	3	azimuthOnly	Azimuth only
	4	full3DSpatialAudio	Full 3-D spatial audio
termCapCPULoad	0	unknown	Unknown or can't determine
	1	0-20%	Less than 20% loaded
	2	20-40%	20-40% loaded
	3	40-60%	40-60% loaded
	4	60-80%	60-80% loaded
	5	80-100%	80-100% loaded
TermCapMemoryLoad	0	unknown	Unknown or can't determine
	1	0-100KB	Less than 100KB free
	2	100-500KB	100KB-500KB free
	3	500KB-2MB	500KB-2MB free
	4	2-8MB	2MB-8MB free
	5	8-32MB	8MB-32MB free
	6	32-200MB	32MB-200MB free
	7	+200MB	More than 200MB free

The following elements support the use of termCap test attribute.

rectangle, circle, text, string, subtitles, points, lines, polygons, curve, img, video, audio, audioClip, box, cone, cylinder, sphere, mesh, group, applicationWindow, inline, xmtaMedia and useMacro

The <use> element is also supported if it is based on the above elements.

9.3.3.1.3 <switch>

The <switch> element is as per SMIL specification.

```
<switch>
    a list of alternate elements to select from according to test attributes…
</switch>
```

The following are a couple of examples to illustrate it's usage. For more information, and more examples, see the SMIL specification.

Selection between audio tracks in different languages.

```
<switch>
  <audio src="audio-english"  systemLanguage="en"/>
  <audio src="audio-francais" systemLanguage="fr"/>
  <audio src="audio-dutch"    systemLanguage="nl"/>
</switch>
```

Selection between audio tracks of various qualities suitable for different bit-rates.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
<switch>
  <audio src="audio-goodquality"   systemBitRate="44100"/>
  <audio src="audio-mediumquality" systemBitRate="22000"/>
  <audio src="audio"/>
</switch>
```

9.3.3.2 CustomTestAttributes

XMT-Ω incorporates the custom test attributes as defined by the SMIL specification. Custom test attributes allow authors to define their own test attributes.

9.3.3.2.1 <customAttributes>

The <customAttributes> element is as per SMIL specification. The element may be present in the document header, <head> section, and contains the definition of available custom test attributes.

9.3.3.2.2 <customTest>

The <customTest> element is as per SMIL specification. The element defines each custom test attribute.

```
<customTest
    id           = ID
    title        = <string value>
    defaultState = "true | false"
    override     = "allowed | not-allowed | uid-only"
    uid          = uri-reference
/>
```

id is an XML identifier.

title is a user defined label for the test attribute.

defaultState is the initial value for named custom test variable that may be overridden.

override allows the author to block overriding the value or limit the means by which overriding can be done

uid identifies a persistent store for the value of the custom test variable.

9.3.3.3 PrefetchControl

XMT-Ω incorporates the PrefetchControl module as defined by the SMIL specification. The <prefetch> element allows authors to influence the scheduling of media, primarily to allow the media to be presented quickly in interactive presentations with non-deterministic start times based on events.

9.3.3.3.1 <prefetch>

The <prefetch> element is as per SMIL specification.

```
<prefetch
    id        = ID
    mediaSize = <bytes-value   | percent-value>
    mediaTime = <clock-value   | percent-value>
    bandwidth = <bitrate-value | percent-value>
/>
```

id is an XML identifier.

mediaSize specifies how much of the media to prefetch as a function of its size.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

mediaTime specifies how much of the media to prefetch as a function of its duration.

bandwidth specifies how much of the network bandwidth to utilize when prefetching.

9.3.3.4 SkipContentControl

XMT-Ω incorporates the SkipContentControl module as defined by SMIL, and the elements containing the skip-content attribute are also as defined by SMIL.

9.3.3.5 Mapping Content Control to MPEG-4

Alternate content can be defined using SMIL Content control module.

Certain content, such as alternate languages can be selectable at run-time via MPEG-4 alternate streams identified with language descriptors.

Otherwise selections can be pre-compiled so that language specific MPEG-4 can be output.

Such run-time versus compile-time selection should be under control of the author during a publish stage that would compile the textual format into the required BIFS, OD and media streams etc.

Content control may also be expressed for ratings, bandwidth etc.

9.3.4 XMT-Ω Layout

XMT-Ω contains layout features that are based on the SMIL Layout features. For XMT-Ω Layout there are no SMIL modules incorporated. The Schema for XMT-Ω Layout is unique to XMT, but it is syntactically similar to SMIL Layout.

9.3.4.1.1 <layout>

The <layout> element is similar to that of the SMIL specification.

```
<layout
   type    = "xmt/xmt-basic-layout"
   metrics = "pixel | meter"
/>
```

type is a string specifying the layout type. The only current valid value “xmt/xmt-basic-layout”.

metrics is a string specifying the metrics used for media dimensions, position etc as per MPEG-4. The valid values are “pixel”, which is the default, or “meter”.

A single, optional <layout> element can be placed into the <head> of the document. When not present the default layout is as if a <layout> had been defined with a default topLayout.

9.3.4.1.2 <topLayout>

The <topLayout> element is similar to that of the SMIL specification. A single topLayout element may be present in the <layout>.

```
<topLayout
    backgroundColor = <color value>
    height          = <value>
    width           = <value>
    enableSubtitles = <"true | false">
/>
```

backgroundColor is a color value for the background of the topLayout. By default the color is “black” .

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

height is a string specifying the presentation preferred height in pixels. The attribute contains an optional value such as "480" that defines the height in pixels

width is a string specifying the presentation preferred width in pixels. The attribute contains an optional value such as "640" that defines the width in pixels.

Only if both a *width* and a *height* are specified will the preferred size flag in MPEG-4 systems decoder specific information descriptor be set true, otherwise it will be set false.

enableSubtitles is a boolean that controls whether any <subtitles> text is visible or not. This attribute may be animated and by default its value is true.

A <topLayout> may contain zero or more <regions> as defined below. The <topLayout> defines the screen for MPEG-4, see subclauses 7.1.1.2.2 to 7.1.1.2.4. The center point of the <topLayout> is the origin and is MPEG-4 point 0,0 for 2D.

xMedia objects, or <group>s that have no region attribute are placed in the <topLayout> and positioned according to the origin. The <group>, and the xMedia's <transformation> sub-element, can be used to position the media within the <topLayout> where the translation, rotation attributes position the objects just as the Transform/Transfrom2D nodes position the objects in MPEG-4.

9.3.4.1.3 <region>

The <region> element is a similar concept to that of the SMIL specification except that positioning of regions is based on center coordinates with a single attribute size containing both width and height. Having a single center and since size attribute facilitates animation of the regions to allow their positions and size to be changed (animated).

Multiple regions may be defined within the <topLayout>. Regions can be defined with regions to allow a hierarchical layout.

A media element, or <group> may be associated to a particular region by using the region attribute in the media, or <group>. The region is identified by its id or by its regionName. The optional regionName does not need to be unique and hence may be used in conjunction with content control module and alternate layouts to name multiple alternate regions the same. If there are alternate regions with the same name there must only be one that is finally selected by the content control, otherwise the layout is ambiguous.

xMedia objects and <group>s placed into the region are positioned off the center of the region where the translation, rotation attributes position the objects just as the Transform/Transfrom2D nodes position the objects in MPEG-4.

z-index is used to control the layerering of mutiple regions that are inside a single region or topLayout. If no z-index is specified then those regions have a highest order and will be on top. Regions having an identical z-index (and all those that where z-index is not specified are considered identical to each other) are layered in document order with the each new region being layered on top of the prior one in the document order.

```
<region
    id              = ID
    backgroundColor = <color value>
    regionName      = <string>
    translation     = <value>
    size            = <value>
    z-index         = <value>
    soundLevel      = <value>
/>
```

id is an XML identifier.

translation is the coordinates for the center of the region .

size is the width and height of the region.

z-index is used to resolve the layering of overlapping regions in the same region or in a topLayout.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

soundLevel is used to set the volume of audio elements placed into the region as per SMIL AudioLayout module.

9.3.4.2 Mapping Layout to MPEG-4

layout: metrics goes to BIFS config.

topLayout: width and height go to BIFS config. backgroundColor maps to MPEG-4 Background:skycolor or Background2D:backColor depending on whether the content is 2D or 3D. topLayout maps to a Layer2D node or OrderedGroup node for 2D content; or a Layer3D node or Group node for 3D content.

region: maps to layer2D or layer3D node

9.3.5 XMT-Ω Linking

XMT-Ω defines a single linking element <a> that is similar to the use of the same element in SMIL and HTML. In XMT-Ω this would be mapped to an MPEG-4 Anchor node to another MPEG-4 presentation.

9.3.5.1 <a>

```
<a
    id   = ID
    href = <uri-reference>
/>
```

id is an XML identifier.

href is a uri-reference to another valid mpeg-4 presentation.

The <a> linking element can contain one or more xMedia elements, <group> elements, xmtaMedia elements or <use> elements referring to any of the preceding types. The <a> element can be nested such that, for example, a <group> contained several child xMedia element can have an <a> element surrounding the group while an xMedia element(s) in the group also has an <a> element. If a media contained by more than one <a> element it is the one that is closest above the media in the scene hierarchy that will be activated when the media is clicked.

The following is an example where, if either of the two media elements circle or rectangle are clicked, the current presentation will end and the new presentation will be started.

```
<a href="dmif://NewMpeg4Presentation">
   <circle radius="25" .../>
   <rectangle size="100".../>
</a>
```

9.3.5.2 Mapping Linking to MPEG-4

Linking provides a high-level representation of the Anchor node which will be used in the mapping to MPEG-4.

9.3.6 XMT-Ω Extensible Media (xMedia) Objects

Although SMIL provides a useful abstraction for media objects in the Media Object Module, SMIL is more about multimedia player composition rather than multimedia object composition as MPEG-4 is about. For instance, SMIL would render text and let the text player concern itself about font, style, colors etc., whereas MPEG-4 text object includes font, style, alignment, and colors and as such is intimately aware of such detail. A combination of HTML+SMIL would include these text attributes but again HTML has its own media content model.
MPEG-4 also contains 2D elements similar to those that are described in SVG (Scalable Vector Graphics), for example Rectangle and Circle. SVG in fact uses modules from SMIL; content selection and animation. The animation module was jointly developed between SMIL and SVG.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The XMT-Ω format will abstract media objects to include behavior in a manner comparable to SMIL Animation, e.g. <set attributeName=”myRect.color” begin=”click” to =”#ff00ee”>. This example would map in MPEG-4 to a TouchSensor on the object that had a route that, via a valuator, set the color of a Rectangle object, named myRect, to the color #ff00ee.

The XMT- Ω syntax and semantics have been designed around extensible media (xMedia) objects. The XMT- Ω defines a set of xMedia elements as basic building blocks for multimedia objects. An xMedia object is defined by an element, such as <img>, or <rectangle>, which abstracts geometry, with media specific attributes and timing attributes. Elements and attributes defined in the XMT-Ω namespace contains a subset of what is defined in SMIL.

Spatial properties of an xMedia element can be further defined by a set of common child elements: <transformation>, <material>, <outline>, <chromakey>, <texture>, <light>, and <hotspots>.

Behavior that are associated with an xMedia element can be defined by a set of animation elements (see the animation section). Events that are associated with an xMedia element (e.g. a mouse click) can be expressed using timing attributes as in SMIL (see the timing section).

An xMedia element also abstracts BIFS, OD commands, media streams, etc.

9.3.6.1 Generic xMedia object

The following defines a generic 2D/3D xMedia object. For media object specific attributes, such as rectangle’s size or circle’s radius, see the individual xMedia object definitions below. Timing attributes are defined in the timing section.

<xMediaObject %media_specific_attributes
%timing_attributes

```
 <transformation
   center = ”cx cy cz”
   rotation = ”angle ax ay az”
   scale = ”sx sy sz”
   scaleOrientation = ”angle ox oy oz”
   translation = ”x y z”
   visibility = ”true | false”
   order = numeric_value
   billboard = true | false
   billboardAxis = “bx by bz”
/>
<material
   ambientIntensity =”i”
   diffuseColor = color
   color = color
   shininess = “s”
   specularColor = color
   transparency = “alpha”
   filled=”true | false”
>

   <outline
      color=”#00ff00”
      width=”2”
      style=”dash-dot”
   />

</material>
<chromakey
   isKeyed = ”true | false”
   isRGB   = “true | false”
   keyColor = “rgb | yuv”
   lowThreshold  = “l”
   highTHreshold = “h”
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      transparency = "alpha"
   />
  <texture
      src = image | video
      <encodingHints> Hints to encode source </encodingHints>
      repeatS = "true | false"
      repeatT = "true | false"
      center = "x y"
      rotation = "angle"
      scale = "x y"
      translation = "x y"
   />
   <light
      ambientIntensity = "i"
      color = color
      direction = "x y z"
      intensity = "i"
      on = "true | false"
   />
  <hotspots>
     %media_elements
   </hotspots>
</mediaObject>
```

Spatial properties of an xMedia object can be defined by its child elements, such as the transformation, material, outline, chromakey, texture, light elements. Behavior of an xMedia object upon user interactions are defined by the hotspots elements.

SMIL attributes *erase* and *sensitivity* are supported in XMT, for compatibility, but only for the SMIL default values of whenDone and opaque respectively.

9.3.6.1.1 <transformation>

```
   <transformation
     center = "cx cy cz"
     rotation = "angle ax ay az"
     scale = "sx sy sz"
     scaleOrientation = "angle ox oy oz"
     translation = "x y z"
     visibility = "true | false"
     order = numeric_value
     billboard = true | false
     billboardAxis = "bx by bz"
   />
```

A <transformation> element allows the object to be positioned, scaled etc (mapped to a Transform or Transform2D node). See subclauses 7.2.2.131 and 7.2.2.132 for definition. The center, rotation, scale, scaleOrientation and translation attributes are mapped to their MPEG-4 node counterparts (Note: rotation attribute maps to rotationAngle field for Transform2D node). Z-coordinates and axis for rotation and scale are only for 3D nodes.

The object can be shown and hidden (MPEG-4 object is mapped under an MPEG-4 switch node for this purpose) using a visibility attribute (familiar to DHTML authors).

Order (See subclause 7.2.2.90.2 OrderedGroup) is used to explicitly control the rendering order of the 2D media object within a <group> element or in the default group. The default group is the group that is used to map the region, or topLayout. Once one media object in a group has an order value all other objects will have a default order value of 0.0 if not explicitly defined. Order field is used primarily to control the rendering order of 2D media objects, since they have no z-index. For 3D objects, having a z-index, the order field is used only for the case of co-planar surfaces to determine which surface to render first. For 2D media objects without an order, or for objects of the same order value, the media objects will be rendered according to their temporal occurrence where later items are rendered on top. Items occurring at the same time will be rendered in the order they appear in the XMT document (document order).

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Billboard and billboardAxis are for 3D media objects only and define media object to be a billboard type as if a child of a Billboard node. See subclause 7.2.2.21 where billboardAxis is the axisOfRotation.

9.3.6.1.2 <material>

```
<material
   ambientIntensity =”i”
   diffuseColor = color
   color = color
   shininess = “s”
   specularColor = color
   transparency = “alpha”
   filled=”true | false”
/>
```

A <material> element specifies the material for a visual media. This is mapped either to a Material or Material2D node depending on whether the media object is a 3D or 2D type respectively - see subclauses 7.2.2.79 and 7.2.2.80 for definition. The color is mapped to emissiveColor, all other attributes are mapped to their MPEG-4 node counterparts. AmbientIntensity, diffuseColor, shininess and specularColor only for 3D media objects; filled attribute is only for 2D media objects. The <material> can contain an <outline> which is only for 2D media objects.

9.3.6.1.2.1 <outline>

```
<outline
   color=”#00ff00”
   width=”2”
   style=”dash-dot”
/>
```

The <**outline**> element contains the style (dotted, dashed, etc.), the thickness and color of a line that is drawn around the periphery of a 2D media object. This is mapped to LineProperties node - see subclause 7.2.2.75 for definition.

9.3.6.1.3 <chromakey>

```
<chromakey
   isKeyed = ”true | false”
   isRGB   = “true | false”
   keyColor = “rgb | yuv”
   lowThreshold  = “l”
   highThreshold = “h”
   transparency = “alpha”
/>
```

The <chromakey> element applies only to <img> or <video>. This defines a chroma key color for shaped image/video extraction. This is mapped to the MaterialKey node - see subclause 7.2.2.81 for definition.

9.3.6.1.4 <texture>

```
<texture
    src = image | video
    repeatS = “true | false”
    repeatT = “true | false”
    center = “x y”
    rotation = “angle”
    scale = “x y”
    translation = “x y”
/>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The <texture> element allows any 3D or 2D visual media object, except <img> and <video>, to be textured. This is mapped either to an ImageTexture or a MovieTexture node, along with a TextureTransform node depending on whether the src is an image or a video - see subclauses 7.2.2.65, 7.2.2.86 and 7.2.2.131 for definitions. The repeatS and repeatT attributes are mapped to the ImageTexture or MovieTexture node. Center, rotation, scale and translation are mapped to the TextureTransform node.

9.3.6.1.5 <light>

```
<light
   ambientIntensity = "i"
   color = color
   direction = "x y z"
   intensity = "i"
   on = "true | false"
/>
```

The <light> element is valid only for 3D media objects. This is mapped to a DirectionalLight - see subclause 7.2.2.48 for definition

9.3.6.1.6 <hotspots>

The <hotspots> element allows media elements, such as circle, rectangle etc or a <group> of objects to be used as hotspot links. The <hotspots> element allows these media elements to be added to any other media elements and designated as a hotspot. An <a> linking tag, see the linking section, or other animation element, see the animation section, would be associated with the hotspot media element to make it active. By nesting the hotspot media elements this way they will track the media object they are the hotspot for if the media object moves or is scaled etc. (The hotspots media elements are attached under the Transform/Transform2D that is mapped by the <transformation> element).

In MPEG-4, any shape can have a TouchSensor attached and can therefore be used as a hotspot. Both 2D and 3D shapes can be used and so a wider range of shapes is available for use in the same way that area can be used. In addition, the MPEG-4 shapes can be made translucent and partially visible; visible as translucent outlines etc. If the shapes use (share) the same appearance, then control of hotspot visibility can easily be provided in a presentation. It may be advantageous, at the authoring level, to identify shapes used as a Hotspot's. Note also the shapes can be timed and provide changing active buttons over time.

As the <hotspots> element contains xMedia objects, whose syntax permits the use of a region, it would create problems and makes no sense if the region attribute on an xMedia hotspot object specified a different region that the group it belonged to. To ensure this cannot happen the region attribute of an xMedia object will be ignored when it is contained in a <hotspots> element.

An object used as a hotspot, although is a media element, cannot contain further hotspots itself.

The media objects used as hotspots are translated according to the 0,0(,0) origin of the media object containing the hotspots and are transformed along with any transformation(s) that act upon that media object.

Media elements in the hotspots can also be enclosed by <anchor> element to allow hotspots to link to other content.

9.3.6.2 <rectangle>

```
<rectangle size="width height"/>
```

The <**rectangle**> element is the XMT-Ω representation of a rectangular media object.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.3.6.2.1 Element content

NOTE: All 2D attributes of the <**material**> element are applicable. Also, the `transparency` attribute of the <**material**> element is meaningful only if the object is filled, i.e., only if the `filled` attribute of the <**material**> element is specified. An unfilled object can be defined by specifying the <**outline**> element (see section on *The <outline> element*) and the `filled` attribute to be false. A filled object with an outline can be defined by specifying both the `filled` attribute to be true and using the <**outline**> element.

9.3.6.3 <circle>

```
<circle radius="radius"/>
```

The <**circle**> element defines the circular media object.

NOTE: See subsection on the *Element content* of *The <rectangle> element*.

9.3.6.4 <text>

```
<text src = "" >
</text>
```

This is a text element that is directly compatible with SMIL media object <text>.

Note: as the src attribute is a URI a text string can be coded directly in the text element using a data uri of the form such as <text src="data:text/plain,this is some text">

9.3.6.5 <string>

```
<string textLines="&quot;A first line of text&quot;;
                   &quot;A second line etc if needed and so on&quot;" >
  <fontStyle .../>
</string>
```

This maps to an MPEG-4 text node and provides a mechanism to include text strings directly rather than via an src.

See subclause 7.2.2.61 for definition of fontStyle which is equivalent to the MPEG-4 node

```
<fontStyle
  family = ""Helvetica Bold"; "SERIF | SANS | TYPEWRITER""
  horizontal = "true | false"
  justify = "BEGIN | FIRST | MIDDLE | END; BEGIN | FIRST | MIDDLE | END"
  language = ""
  leftToRight = "true | false"
  size = 24.0
  spacing = 1.0
  style = "PLAIN | BOLD | ITALIC | BOLDITALIC"
  topToBottom = "true | false"
/>
```

9.3.6.6 <subtitles>

Subtitles are text that are used for accessibility on audio streams. Under player client selection sub-titles may be enabled or disabled and are available as a separate object from text for this purpose. An attribute in <topLayout> called *enableSubtitles* is available to be animated for this purpose.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Like the <text> element the <subtitles> contains a <fontStyle>. The <fontStyle> may be directly here or a <use> that references a <fontstyle> in the <defs>. The latter case allows more than one <subtitles> text block to share the same font style.

```
<subtitles textLines="&amp;quot;A first line of subtitles&amp;quot;;
                      &amp;quot;A second line if wanted etc.&amp;quot;" >
  <fontStyle .../>
</subtitles>
```

9.3.6.7 <img>

```
<img src = "">
   <encodingHints> Hints to encode source </encodingHints>
</img>
```

The <img> element is directly compatible with SMIL media object <img>. The <img> element is an imageTexture mapped onto a Bitmap geometry.

Note: <img> is always mapped to Bitmap node. For other mappings an image can be textured onto a rectangle, or other visual media objects, using the <texture> element.

The <material>, <outline> and <texture> elements are not valid.

9.3.6.7.1 JPEG image sequence

For a sequence of images, when they are coded as JPEGs, it is possible to map into MPEG-4 the images to individual ObjectDescriptors or into the same ObjectDescriptor with one JPEG as a single timed Access Unit. The exact mapping to MPEG-4 is not dictated by XMT-Ω however <deliveryHints> may be used on each <img> to specify it is part of a common elementary stream. This is accomplished by using <param> with the name "ImageSequence" to denote the images are part of a common sequence were the sequence is identified using an identification in the value

<param name="ImageSequence" value="*A unique stream identification*"/>

For example:

```
<seq>
  <img src="firstImage.jpg" dur="5s">
    <deliveryHints>
      <param name="ImageSequence" value="slideShow1"/>
    </deliveryHints>
  </img>
  <img src="secondImage.jpg" dur="8s">
    <deliveryHints>
      <param name="ImageSequence" value="slideShow1"/>
    </deliveryHints>
  </img>
  <img src="thirdImage.bmp" dur="2s">
    <deliveryHints>
      <param name="ImageSequence" value="slideShow1"/>
    </deliveryHints>
  </img>
</seq>
```

In the above example the first two image sources were JPEG and the latter image source was a BMP. This will be re-coded as a JPEG during the mapping to MPEG-4 so that the <deliveryHints> for the image sequence can be honored.

9.3.6.7.2 Images as a PixelTexture

Although many images will be mapped to MPEG-4 as ImageTextures with an Elementary Stream conveying the pixels it is possible to map an image to a PixelTexture. For small images for use as icons, cursors etc mapping as a PixelTexture eliminates the need for an elementary stream for the image, plus mapping to PixelTexture allows non-rectangular

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

transparent images to be created. To specify that an <img> should be encoded as a PixelTexture the following <encodingHints> are to be used.

In the <targetFormat> specify <param> with a name of ”PixelTexture” and one value from the following =”mono”, ”monoAlpha”, “color” or “colorAlpha”.

9.3.6.8 <video>

```
<video src =”“ >
   <encodingHints> Hints to encode source </encodingHints>
</video>
```

The <video> element is directly compatible with SMIL media object <video>. The <video> element is a movieTexture mapped onto a Bitmap geometry.

Note: <video> is always mapped to Bitmap node. For other mappings a video can be textured onto a rectangle, or other visual media objects, using the <texture> element.

The <material>, <outline> and <texture> elements are not valid.

9.3.6.9 <points>

```
<points
   color = ”rgb; rgb; rgb”
   coord = ”x y z; x y z; x y z”
/>
```

See subclauses 7.2.2.98 PointSet and 7.2.2.99 PointSet2D for definition of the attributes which are equivalent to the MPEG-4 node fields. The color attribute is an array of colors one for each coordinate in the coord attribute. The coord attribute has either “x y” pairs or “x y z” triplets depending on whether the element is used as a 2D media element or 3D element. As per MPEG-4 Systems if no color is specified then the color used for the points will be as specified by the <material>.

9.3.6.10 <lines>

```
<lines
   color = ”rgb; rgb; rgb”
   coord = ”x y z; x y z; x y z”
   colorIndex = ”“
   colorPerVertex = ”true | false”
   coordIndex = ”“
/>
```

See subclauses 7.2.2.68 IndexedLineSet and 7.2.2.69 IndexedLineSet2D for definition of the attributes which are equivalent to the MPEG-4 node fields. The coord attribute has either “x y” pairs or “x y z” triplets depending on whether the element is used as a 2D media element or 3D element. As per MPEG-4 Systems if no color is specified then the color used for the lines will be as specified by the <material>.

9.3.6.11 <polygons>

```
<polygons
   coord = ”x y; x y; x y”
   coordIndex = ”“
   color = ”rgb; rgb; rgb”
   colorIndex = ”“
   colorPerVertex = ”true | false”
   texCoord = ”x y; x y; x y”
   texCoordIndex = ”“
/>
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

See subclause 7.2.2.67 IndexedFaceSet2D MPEG-4 node fields. The coord attribute has “x y” pairs as the element is used as a 2D media element. As per MPEG-4 Systems if no color is specified then the color used for the polygons will be as specified by the <material>.

<material> 2D attributes filled, color and alpha and <outline> apply.

9.3.6.12 <curve>

```
<curve
   points = "x y; x y; x y"
   fineness =
   type =
/>
```

See subclause 7.2.2.43 Curve2D for definition of the attributes which are equivalent to the MPEG-4 node fields.

9.3.6.13 <audio>

```
<audio src = "">
   <sound.../>
   <encodingHints> Hints to encode source </encodingHints>
</audio>
```

The <audio> element is directly compatible with SMIL media object <audio>.

For compatibility with SMIL the volume of the audio may be controlled through the soundLevel attribute of the region into which the audio is placed.

Alternatively a <sound> element may be used to control the sound intensity, location etc. When a <sound> element is used the soundLevel attribute of the region is ignored for that <audio> element.

The <sound> element is as follows

```
<sound
   location   = "x y z"
   intensity  = value
   spatialize = "true | false"
   direction  = "x y z"
   maxBack    = "value"
   maxFront   = "value"
   minBack    = "value"
   minFront   = "value"
   priority   = "value"
/>
```

where location (with x and y values), intensity and spatialize are valid for use in 2D contexts, all attributes being valid for use in 3D contexts.

Where <sound> is not used, intensity is set through the soundLevel attribute in the region, otherwise it is as if a default <sound> were present with default values for the other attributes. The default values being the same as the Sound or Sound2D node defaults dependent upon context.

9.3.6.14 <audioClip>

```
<audioClip src = "">
   <sound.../>
   <encodingHints> Hints to encode source </encodingHints>
</audioClip>
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

See subclause 7.2.2.10 AudioClip. Note that the MPEG-4 fields relating to timing are managed now via the XMT timing attributes.

The volume of the audioClip is either controlled through the soundLevel attribute of the region into which the audioClip is placed or via a <**sound**> child element, see <audio> element above for <**sound**> definition.

9.3.6.15 <box>

```
<box size="width height depth"/>
```

The <box> element defines a box media object.

NOTE: All attributes of the <**material**> and <**transformation**> element applies. The <**texture**> element applies while <**outline**> does not.

9.3.6.16 <cone>

```
<cone radius="r"
      height="h"
      hasSide="true|false"
      hasBase="true|false"/>
```

The <**cone**> element defines a 3D conical media object. This is mapped to the Cone node - see subclause 7.2.2.37 for definition. The attribute radius maps to field bottomRadius; hasSide and hasBase map to side and bottom respectively.

NOTE: See the subsection *Element content* of *The <box> element*.

9.3.6.17 <cylinder>

```
<cylinder radius="r"
          height="h"
          hasSide="true|false"
          hasBase="true|false"
          hasTop ="true|false"/>
```

The <**cylinder**> element defines a cylinder media object. This is mapped to the Cylinder node – see subclause 7.2.2.44 for definition. The attributes hasSide, hasBase and hasTop map to side, bottom and top respectively.

NOTE: See the subsection *Element content* of *The <box> element*.

9.3.6.18 <sphere>

```
<sphere radius="r"/>
```

The <**sphere**> element defines a spherical media object. This is mapped to the Sphere node - see subclause 7.2.2.118 for definition.

NOTE: See the subsection *Element content* of *The <box> element*.

9.3.6.19 <mesh>

```
<mesh
   coord = "x y z; x y z; x y z"
   coordIndex = ""
   color = "rgb; rgb; rgb"
   colorIndex = ""
   colorPerVertex = "true | false"
   normal = "x y z; x y z; x y z"
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      normalIndex = ""
      normalPerVertex = "true | false"
      texCoord = "x y; x y; x y"
      texCoordIndex = ""
      creaseAngle  = "0.0"
      ccw    = "true | false"
      convex = "true | false"
      solid = "true | false"
   />
```

See subclause 7.2.2.66 IndexedFaceSet MPEG-4 node fields. The coord attribute has “x y z” triplets as the element is used as a 3D media element.

9.3.6.20 <inline>

```
<inline src = "">
</inline>
```

See subclause 7.2.2.70 Inline. Allows a complete MPEG-4 presentation to be included (in-lined) in the scene.

9.3.6.21 <applicationWndow>

```
<applicationWindow  %timing_attributes
   src = ""
   description = ""
   parameter = ""
   size = "w h"
/>
```

An applicationWindow media object. This is mapped to the ApplicationWindow node - see subclause 7.2.2.7 for definition.

9.3.6.22 <delay>

```
<delay %timing_attributes/>
```

A delay (null media object). This object contains no sub-elements.

9.3.7 SMIL media object modules in XMT-Ω

The SMIL Media Object modules for MediaClipping, MediaClipMarkers and MediaDescription are incorporated into XMT-Ω.

9.3.7.1 MediaClipping

This module is fully incorporated. Using clipBegin and clipEnd attributes a sub-clip of a continuous media object can be defined as offsets from the start of the media that identify the beginning and end of a sub-clip.

9.3.7.2 MediaClipMarkers

This module is fully incorporated. This module defines clips using named time points rather than using clock values as used by clipBegin and clipEnd. Named time points can be mapped to MPEG-4 Segment Descriptor.

9.3.7.3 MediaDescription

This module is fully incorporated. This module contains attributes to provide a brief description of the content (abstract) the author, copyright and a title.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.3.8 XMT-Ω Media Augmentation objects

The following objects act like media objects in that they are part of scene graph hierarchy. The objects act to augment the media objects.

9.3.8.1 <fog>

```
<fog  %timing_attributes
   color = “”
   fogType = “LINEAR | EXPONENTIAL”
   visibilityRange = ””
/>
```

The <**fog**> element is a media augmentation object. This is mapped to the Fog node – see subclause 7.2.2.60 for definition.

9.3.8.2 <pointLight>

```
<pointLight
   ambientIntensity = “i”
   attenuation = “a0 a1 a2”
   color = color
   intensity = “i”
   location = “x y z”
   radius = “r”
   on = “true | false”
/>
```

The <**pointLight**> element is a media augmentation object. This is mapped to the PointLight node - see subclause 7.2.2.97 for definition.

9.3.8.3 <spotLight>

```
<spotLight
   ambientIntensity = “i”
   attenuation = “a0 a1 a2”
   beamWidth = angle
   color = color
   cutOffAngle = angle
   direction = “x y z”
   intensity = “i”
   location = “x y z”
   radius = “r”
   on = “true | false”
/>
```

The <**spotLight**> element is a media augmentation object. This is mapped to the SpotLight node - see subclause 7.2.2.120 for definition.

9.3.8.4 <backdrop>

```
<backdrop color=”” src = “”>
   <encodingHints> Hints to encode source </encodingHints>
</backdrop>
```

The <**backdrop**> element is a media augmentation object. This is mapped to the Background2D node - see sublclause 7.2.2.18 for definition.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A <backdrop> will either bind to the entire scene or to a <group> when the <backdrop> is a child element of a <group> and the <group> has a size. See subclause 9.3.9.1 for more detail.

9.3.8.5 <background>

```
<background
   skyColor=""
   skyAngle=""
   groundColor=""
   groundAngle=""
   backSrc = ""
   bottomSrc = ""
   frontSrc = ""
   leftSrc = ""
   rightSrc = ""
   topSrc = ""
   <encodingHints> Hints to encode source </encodingHints>
</background>
```

The **<background>** element is a media augmentation object. This is mapped to the Background node - see sublclause 7.2.2.17 for definition.

One or more <encodingHints> elements can be used to supply encoding hints for the image sources. There are a maximum of six images for the back, bottom, front, left, right and top. An encoding hint applies to the respective image(s) by supplying a name/value pair in the <targetFormat> element of the <EncodingHints> such as <param name="src" value="front"> which means that the encoding hints refer to the frontSrc image. The encoding hints will apply to more than one image if more that one name/value pair with name="src" is used within a single encoding hints target format.

A <background> will either bind to the entire scene or to a <group> when the <background> is a child element of a <group> and the <group> has a size. See subclause 9.3.9.1 for more detail.

9.3.9 XMT-Ω Media Group object

The <group> object acts like a media object. It groups other media and media augmentation objects together such that they can be manipulated as a single larger composite media object within the scene graph hierarchy.

9.3.9.1 <group>

Objects can be grouped and positioned through the group construct as defined in this section. This allows a group of objects to manipulated as a whole, e.g. moved together, scaled or made visibility.

```
<group %timing_attributes
  id =""
  center = "cx cy cz"
  rotation = "angle ax ay az"
  scale = "sx sy sz"
  scaleOrientation = "angle ox oy oz"
  translation = "x y z"
  visibility = "true | false"
  order = numeric_value
  billboard = true | false
  billboardAxis = "bx by bz"
  size = "w h"
  backgroundColor=""
  collide = "true | false"
/>
```

The <group> element has most of its attributes in common with <transformation> as defined in the media elements. The <group> acts like a <par> time container.

The size, background color and collide attributes are unique to <group>.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The size and background attributes may used to limit the rendering dimensions of the group and to provide a backgroundColor within those dimensions. The backgroundColor attribute is ignored when the size attribute is not present. A size of “-1.0 -1.0” will size the group to the dimensions of its parent; have one specified size and the other as dimension as –1 will cause the dimension that is –1 to be sized to its parent. The parent being either a <group> that is above this one in the hierarchy or if there are none then to the region or layout into which it is placed.

The collide attribute may be used in a 3D context and will cause collision detection amongst the children when set true. A collide event will be raised the group when a collision is detected.

Objects are rendered within the <group> according to the same rules as defined for the media objects <transformation> element.

```
<group id=”myGroup” begin=”0s”>
  <rectangle size=”10 10” begin=”2s” dur=”5s”/>
  <circle radius=”30” dur=”12s”/>
</group>
```

In the above example the circle appears first at t=0s. At t=2s the rectangle is added and appears on top of the circle. It is removed 5s later and the circle remains visible for a further 7s when the group ends after its children end at 12s.

```
<group id=”myGroup” begin=”0s”>
 <rectangle size=”50 50” dur=”5s”/>
 <circle radius=”30” dur=”12s”/>
</group>
```

In the above example the rectangle and circle both start at t=”0s”. The circle will now appear on top as the media objects in this case are rendered in document order.

Rendering order can however be different if order attribute is specified. In the following example the circle begins at the same time as the rectangle but the rectangle will be rendered on top as it has a greater order value than the circle.

```
<group id=”myGroup” begin=”0s”>
  <rectangle size=”10 10” dur=”5s”>
    <transformation order=”2.0”/>
  </rectangle>
  <circle radius=”30” dur=”12s”>
    <transformation order=”1.0”/>
  </circle>
</group>
```

A <group> can also be nested within another <group> to create any desired hierarchy. The order attribute in the <group> controls its rendering order within its parent.

The <group> element collects elements together as a group so that the set of objects can be acted upon as a whole using the attributes of the <group>. For example the complete group can be translated or scaled. The <group> element also supports the region attribute, like the xMedia objects, so that the group of objects placed in a given region. As the <group> element can contain xMedia objects, that may also have a region attribute, it would create problems if the region attribute on a child xMedia object specified a different region that the group it belonged to. To ensure this cannot happen the region attribute of an xMedia object will be ignored when its parent is a group element and the region will be determined solely by the region attribute on the <group> element.

If a <group> has a size attribute specified, if the size field is animated (<set> or <animate>), or if a <backdrop> or <background> is present as a child element of <group> then the <group> shall be mapped to a Layer2D or Layer3D as appropriate, or the mapping shall behave as if this were the case. For example in a very basic profile the Layer nodes may not be there, then the XMT-O can simulate the required mapping using nodes that are available or alert the author if an adequate mapping cannot be achieved. This mapping specification, to behave as if the <group> is mapped to Layer2D/3D, is principally to ensure that the <backdrop> or <background> will bind to the <group> and not to the entire scene. A <background> or <backdrop> that is not a child element of a <group> will form the background to the entire scene.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.3.10 XMT-Ω custom XMT-A Media objects

The following construct defines a custom media object using XMT-A nodes, descriptor commands etc. The construct allows complicated custom media objects and behavior to be specified and also allows for XMT-A capabilities to be incorporated that have no equivalent representation in XMT-Ω.

9.3.10.1 <xmtaMedia>

```
<xmtaMedia %simple_timing_attributes region=""
  id="" type="2D | 3D | Audio" hasOD="true | false">
  <nodes>
    <Group> or <OrderedGroup>
      <children>
        Visual, or Audio nodes etc
      </children>
    </Group> or </OrderedGroup>
    <ROUTE../>
    <ROUTE../>
  </nodes>
  <cmds>
    Zero or more <par> BIFS, OD, ES, IPMP commands etc and StreamSources
  </cmds>
</xmtaMedia>
```

An <xmtaMedia> element defines a custom media object. Attributes include support for simple timing support and region which allows the object to be placed in the optionally designated layout region. Simple timing is BasicInline Timing module as per SMIL, i.e. begin, end and dur with simple offset timing only where the intrinsic duration of an xmtaMedia element is 0s. The type attribute specifies whether the media is of type 2D, 3D or audio. This information allows XMT-Ω to create the necessary framework within which to place the media object. The hasOD attribute indicates whether there are OD commands contained in the xmtMedia that would require an OD stream to be defined in the InitialObjectDescriptor.

To control the custom media's position, its visibility etc the <xmtaMedia> element should be placed within an XMT-Ω <group> element. A duration must be specified for the media as its duration cannot necessarily be determined by the nodes and cmds.

The custom media definition is in two parts:

The first part, contained in the <nodes> element is an XMT-A <Group> or <OrderedGroup> node that may contain any valid XMT-A nodes as well as any ROUTEs needed to connect the nodes.

The second part, the optional <cmds> element, contains any necessary XMT-A Object Descriptor Framework elements to support or augment the nodes, as well as any BIFS commands to update the nodes. All BIFS commands, except for Replace Scene may be used. All OD Framework commands may be used and streams may be defined for MPEG-J, OCI, IPMP. The <par> element may be used to time the commands as defined by XMT-A. The time defined by the <par> element will be offset by the overall begin time of the <xmtMedia> object.

9.3.11 XMT-Ω MetaInformation

XMT-Ω incorporates the SMIL MetaInformation module as defined by SMIL.

9.3.11.1.1 <metadata>

The <metadata> element is as per SMIL specification.

```
<metadata>
    hosts RDF elements
</metadata>
```

The metadata element hosts RDF metadata as per the Resource Description Framework (RDF), see the SMIL specification for further information.

9.3.11.1.2 <meta>

The <meta> element is as per SMIL specification. The element specifies a single property/value pair in the name and content attributes, respectively.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
<meta
    id      = ID
    name    = <value>
    content = <value>
/>
```

id is an XML identifier.

name is the name of a property

content is the value of that named property

9.3.11.2 Mapping MetaInformation to MPEG-4

MetaInformation can be retained in the XMT document instance. MetaInformation can also be encoded as OCI.

9.3.12 XMT-Ω Structure

XMT-Ω defines a document structure that is similar, but not identical, to the SMIL structure. XMT-Ω has added a <defs> element into the <head>, much like SVG, to allow definitions for later re-use within the presentation.

9.3.12.1 <XMT-O>

<XMT-O> can contain a single <head> and <body> in that order.

9.3.12.2 <head>

The XMT-Ω can contain <meta>, <customAttributes>, <metadata>, <layout>, <transition>, <defs> and <macros>

9.3.12.3 <body>

And then there is the content itself. A key feature of the content is the high-level constructs for the audio-visual objects. These constructs not only describe the objects but also include behavior.

Like SMIL the <body> element has the timing semantics of a <seq> timing container.

9.3.13 XMT-Ω DEFS

XMT- Ω provides a means to define commonly used elements for a document.

9.3.13.1 <defs>

The XMT-Ω <head> can contain a <defs> element. Elements within the <defs> section can be as follows:

xMedia elements
xMedia augmentation elements
<group>
<xmtaMedia>
<material>, <chromakey>, <texture>, <light> sub-elements from xMedia elements
<outline> (from <material>)
<fontStyle> element (from <text>)

The id of an element allows and element, and any contained elements, to be referenced in the <body>.

```
<defs>
  <fontStyle id="myBold"  family="SERIF" size="24" style="BOLD"/>
  <rectangle id="bigRect" size="200 300">
    <material  id="myRed"  color="#FF0000" filled="true"/>
  </rectangle>
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    <circle id="smallCirc" radius="1">
      <use xlink:href="#myRed"/>
    </rectangle>
    <material  id="myWhite"  color="white" filled="true"/>
  </defs>
  <body>
    <string textLines="&amp;quot;My line of text&amp;quot;">
      <use xlink:href="#myBold"/>
    </string>
    <use xlink:href="#bigRect"/>
    <circle radius="50">
      <use xlink:href="#myWhite"/>
    </circle>
    <circle radius="20">
      <transformation translation="10 10"/>
      <use xlink:href="#myRed"/>
    </circle>
    <use xlink:href="#smallCirc"/>
  </body>
```

Elements within the <defs> section can also contain animation elements such as <set>, <animate> and <animateColor>. This would normally be used to provide materials whose colors pulsated, small spinning objects that can be re-used etc. The timeline for the animation elements is the document instance timeline that begins at t=0s. So for example setting a color in a material to red at 20s would do so at t=20s and would have the effect of turning all objects red that used the material. No timing is allowed on any other elements, except for the XMT-A <par> timing in an xmtaMedia object which is also relative to the document instance timeline that begins at t=0s, like the animation.

Only discrete xMedia elements are supported in the <defs> section; i.e. no continuous media such as video or audio.

The region attribute is not supported for elements in the <defs> section, the respective <use> will later define the region for that instance (as well as timing etc).

9.3.13.2 <use>

The <use> elements can only substitute in positions where the element(s) would occur if defined directly rather than via the <use>. A <use> can also occur in the <defs> section as part of a larger set of elements making a defined entity. but not as an element immediately under the <defs>, i.e an entity being defined can be created from a combination of original elements as well as other entities defined in the <defs> section.

When a <use> substitutes for an xMedia object it can have attributes that are supported by the xMedia object. These include timing attributes and region.

No child elements are permitted in the<use>; a <use> is identical to its definition. To size and position a <use> when it is an xMedia type the <use> may be placed under a <group> and the translation, scale, center attributes etc of the <group>

9.3.13.3 Mapping DEFS to MPEG-4

Elements in the <defs> section and the <use> maps to the MPEG-4 DEF and USE attributes.

9.3.14 XMT-Ω MACROS

XMT-Ω provides a means to define custom objects with virtual attributes and optional virtual values for these attributes. This facility provides the author a means to define a (complex) object and create many similar instances, say only varying only by a single virtual attribute value. Each high level object, or macro, can have many elements forming the object and the macro provides a mechanism to define virtual attributes by whch the object may be manipulated rather than using the actual attributes of the media objects that comprise the macro. This not only provides a level of abstraction but allows a one to many relationship between the virtual attribute and the actual attributes to which it refers. Abstract virtual values may be defined for each virtual attribute that allow the attribute to be manipulated using values that are more appropriate to the abstraction level.

9.3.14.1 <macros>

The XMT-Ω <head> can contain a <macros> element. Within the <macros> section individual <macro> objects can be defined:

,,,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2005 – All rights reserved
Not for Resale

9.3.14.2 <macro>

The XMT-Ω <macro> allows an object to be defined along with virtual attributes that are aliases for one or more actual attributes in the object itself. Such attributes can also have virtual values defined along with the corresponding actual values that the actual attributes in the object need to be set to.

Each <macro> can contain one or more <attrib> elements that define virtual attributes, followed by a single media <group> element that contains xMedia objects etc., as can normally be contained in a <group>. Timing and region attributes that could normally be present on a <group> element are not permitted when used in this <macro> context and are ignored if present. <use> elements are however permitted within the <group> to be part of the macro. Animation elements may be present to animate attributes within the <group> content.

Each virtual attribute, as defined by <attrib>, contains a <targets> element that lists the actual attributes of the elements contained within the <group> that are to be set when the virtual attribute value is set. <targets> contains a list of one or more <target> elements that identify an element and the attribute therein that is to be set. Each <target> can be identified by an *id* that allows it to be referenced for the creation of virtual values. Each <attrib> can contain zero or more virtual values following the <targets> list. If the actual attribute types are not identical then the virtual attribute can only be manipulated using virtual values and hence in this case at least one virtual value should be defined. Each virtual value element <value> contains a name attribute so as to create a named value. This named value can be used with the virtual attribute to set the virtual attribute value. A <value> contains a list of <put> elements that defines values to be set to each of the <target> attributes. A virtual value can be derived from another using a base="" attribute. In this case the <put> values override those listed from the virtual value upon which the derived value is based.

This is a very simple example defining a button in a 2D context. One virtual attribute is created called 'faceColor' and this is defined as being two actual target attributes in the object. In this case the value types are the same (both color) although they do not have to be. Then some (optional) abstract values are defined, one of which uses an already defined value as a base and makes a delta to it (brightpink is based off murkypink). Another virtual atribute 'dropShadow' shows a virtual attribute that is comprised of actual attributes whose types are not the same.

```
<macro name="button">
  <attrib name="faceColor">
    <targets>
      <target id="f1" targetElement="mat1"
                      attributeName="color"/>
      <target id="f2" targetElement="mat2"
                      attributeName="color"/>
    </targets>
    <value name="murkypink">
  <put target="f1" value="#ff8800"/>
  <put target="f2" value="#f55800"/>
    </value>
    <value name="brightpink" base="murkypink">
  <put target="f2" value="#ff0000"/>
    </value>
    <value name="muddyblue">
  <put target="f1" value="#ff88ff">
  <put target="f2" value="#5677ef"/>
    </value>
  </attrib>
  <attrib name="dropShadow">
    <targets>
      <target id="f11" targetElement="matd"
                       attributeName="color"/>
      <target id="f12" targetElement="t2dshadow"
                       attributeName="translation"/>
    </targets>
    <value name="bottomRightDark">
  <put target="f11" value="#020202"/>
  <put target="f12" value="4 -4"/>
    </value>
    <value name="topLeftLight">
  <put target="f11" value="#404040">
  <put target="f12" value="-5 5"/>
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    </value>
  </attrib>
  <group>
 Here is the actual xMedia shapes as already defined by XMT-Ω
 that form the actual object and has material elements with ids
 mat1 and mat2, etc. that are referred to above.
  </group>
</macro>
```

9.3.14.3 <useMacro>

To use a <macro> it can be instantiated using <useMacro> and the virtual attributes can be given values. Where the values are base types e.g. color, and all actual attributes are of the same type, any base type value from XMT- Ω can be used to instantiate it, in this case the color #00ff00. The virtual attribute can then be animated in the normal way using <set> and <animate>. And for animation too base type values can be used if the actual attributes are of the same type, and if not only virtual values. For example

```
<useMacro xlink:href="#button" faceColor="#00ff00">
  <set     begin="5s"  attributeName="faceColor"  to="murkypink"/>
  <animate begin="25s" attributeName="faceColor"  to="#0000ff" dur="20s"/>
  <set     begin="30s" attributeName="dropShadow" to="topLeftLight"/>
</useMacro>
```

Where the virtual attributes are a mix of many types the only values supported will be the virtual values, which will list the complete set of actual values to set in the actual attributes to make that virtual value.

All the common attributes that can be used on an xMedia element are present for <useMacro>. These attributes include timing, region etc.

A <useMacro> may be used anywhere a <group> can be used (the <macro> content being formed using <group>). <useMacro>s can also be present in the <defs> section.

9.3.15 XMT-Ω Timing and Synchronization

XMT- Ω Objects are timed using the SMIL Timing and Synchronization modules. We'll use the terms SMIL timing and XMT- Ω timing (or simply XMT timing) interchangeably in the rest of this document and distinguish them when necessary. The syntax and semantics of timing elements and attributes are as per SMIL specification unless otherwise specified.

SMIL Timing (and thus XMT- Ω timing) defines elements and attributes to coordinate and synchronize the presentation of media objects over time. Three synchronization elements, called as time containers, support timing by grouping their contained children into coordinated timelines:

The <seq> element plays the child elements one after another in sequence.
The <excl> element plays one child at a time, but does not impose any order.
The <par> element plays child elements as a group (allowing "parallel" playback).

XMT- Ω Timing attributes specify element's timing behavior (as per SMIL specification). The essential timing attributes are are :

begin: specifies the begin time of an element in a variety of ways, ranging from simple clock times to an event (e.g. a mouse-click) occurrence.

dur: specifies the duration of an element.
end: specifies the end time of an element in a variety of ways as in the begin time.
min: specifies the minimum active duration of an element.
max: specifies the maximum active duration of an element.

An example:

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
<par begin="0s" dur="15s">
  <rectangle . . . />
  <img . . . />
  <seq>
    <text. . . />
    <circle. . . />
  </seq>
</par>
```

In this example, <par> and <seq> are used to denote required timing and they will be mapped to BIFS updates at the required times along with OD and media streams as required by the xMedia objects.

The XMT-Ω also supports the following timing attributes:

repeatDur, repeatCount: specifies how long, and how often the element is to repeat.

fill: specifies how an element can be extended (e.g. by freezing the final state) to fill gaps in a presentation.

endsync: controls the active duration of the par and excl containers with respect to the active duration(s) of child element(s). It allows the time container to "co-end," or end together, with selected child element(s).

restart, restartDefault: controls restart behavior of an element

syncBehavior, syncBehaviorDefault: defines the runtime synchronization behavior for an element.

SyncMaster: if set to true, this forces other elements in the time container to synchronize their playback to this element.

SMIL Timing and Synchronization support is broken down into a number of semantically related elements, attributes and attribute values. XMT- Ω, in re-using the SMIL Timing and Synchronization modules, support most of the SMIL timing constructs, including time containers, sync-arcs, event based timing, stream dependencies, etc. The XMT- Ω also fully incorporates SMIL Time Manipulations module. The XMT-Ω supports the FlexTime model in MPEG-4 by reusing the SMIL timing modules with two additional XMT-Ω specific attributes, namely flexBehavior, flexBehaviorDefault.

As is the case with SMIL, not all XMT-Ω elements may specify their timing behavior using timing attributes. Those XMT-Ω elements that support timing attributes are: All xMedia elements, all animation elements, time container elements, and the group element.

9.3.15.1 SMIL Timing and Synchronization Modules in XMT- Ω

9.3.15.1.1 AccessKeyTiming

Fully incorporated: InputSensor node can be used to provide the necessary keyboard input function.

9.3.15.1.2 BasicInlineTiming

Fully incorporated but for restricted use of indefinte: dur attribute with all allowed values, and begin and end attributes with simple offset values.

Note that the "indefinite", while supported for dur attribute as the duration can be terminated by an event such as click, supporting this value for begin and end attributes Is not currently feasible for XMT and MPEG-4 as there is no defined external interface in MPEG-4 terminals to trigger the begin or end. SMIL, being a W3C standard, would support using the DOM to trigger the elements begin or end and hence in SMIL indefinite begin or end value is feasible.

9.3.15.1.3 BasicTimeContainers

Fully incorporated: par and seq elements, and fill, endsync attributes on them. The fill attribute on a time container describes its display behavior, and endsync describes its end conditions.

An example:

```
<par endsync="first">
  <video src="intro.mpg" dur="media"/>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    <audio src="intro.au" dur="media"/>
  </par>
```

In this example the par time container ends when any of the child elements ends its intrinsic media duration dur="media." The endsync attribute on a time container may also be "all," "last," or id-value to end the time container when all the child elements end, when the last one that has been activated ends, or when the child element specified by the id-value ends, respectively.

9.3.15.1.4 EventTiming

Fully incorporated: begin and end with event values. In addition to 2D interactive behavior, XMT- Ω also supports user interaction with 3D objects. These behaviors are specified in a manner similar to SMIL. An event source is routed to its event sink by using the timing attributes begin or end. As an example, the code fragment below shows that a circle appears when an object with an identifier m1 is clicked. The mouse click event associated with the "m1" object is routed to cause the circle to appear.

Example:

```
 <circle radius="20" begin="m1.click">
<material color="red"/>
 </circle>
```

Note: Begin and end times activated by user interaction, i.e. not defined at compile time, cannot be easily supported for media streams at the current level of MPEG-4. It requires the proposed MediaControl node. Support may be possible however for objects composed entirely from BIFS (such as circle, rectangle, etc.) using conditional nodes that can be activated at run-time via user interaction.

Note: In general the main time-line must be defined at compile-time as the scheduling of transmission cannot be managed interactively at run-time given today's MPEG-4. This situation is in changing as amendments being worked on will bring new functionalities. OD execution semantics may facilitate client-side data requests for transmission. The proposed MediaControl node and/or ServerCommands will also alleviate the situation and allow much more dynamic stream control during playback.

9.3.15.1.4.1 XMT events

In addition to supporting basic events like mouse click, mouse up, mouse down, mouse over and mouse out, XMT allows users to specify other behaviors beyond SMIL for interacting with MPEG-4 objects in both 2D and 3D spaces. These behaviors are specified as suffix to the identifier of an object, such as "M1.mousedrag", "M1.click", etc, with the exception of collision. The collide suffix will be associated with a collision group, instead of with an object.

Table 96 — XMT supported behaviors and suffixes

Behavior/Event	XMT defined event symbols	SMIL defined event symbols
Mouse click	click	activateEvent
Mouse up	mouseup	
Mouse down	mousedown	inBoundsEvent
Mouse out	mouseout	outOfBoundsEvent
Mouse over	mouseover	
Detecting visibility of an object	viewable	
Detecting proximity of an object	near	
Detecting collision of objects	collide	
Observed begin of object		beginEvent
Observed end of object		endEvent
Object has begun a repeat cycle		repeatEvent

The table above lists the XMT behavior and their corresponding suffixes. XMT has defined unique event symbols to support events not defined in SMIL, but also supports some of the event symbols defined by the SMIL language. In some cases there are two symbols for the same event, for example click and activateEvent. An authoring tool should support

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

both and treat them as synonyms. Click is potentially more readable but activateEvent is more interoperable with the SMIL language. Dependent on how the authored content is to be used an author can choose to use either.

Suffixes such as click, mouseup, mousedown, mouseout, mouseover, mousedrag and rotate are associated with the identifiers of both 2D and 3D objects. Collide applies to 3D objects.

To detect the collision of 3D media objects with other 3D media objects the objects are placed within a <group> that has the collide attribute set to true. The sample XMT fragment below shows that the audio starts playing when the two objects with identifiers s1 and b1 collide.

```
<head>
  <layout metrics="pixel">
     <topLayout width="640" height="480"/>
  </layout>
</head>
<body>
  <group id="CG1" collide="true">
    <sphere id="s1" radius="50">
      <transformation translation="10 10 10"/>
    </sphere>
    <box id="b1" size="10 10 10">
      <transformation translation="-10 10 10">
        <animateMotion to="10 10 10" dur="5s"/>
      </transformation>

    </box>
  </group>
  <audio src="myaudio.wav" begin="CG1.collide"/>
</body>
```

Additional constraints on an object's behavior/ event can be specified via escape to the XMT-A construct. For instance, we may want to specify the region within which an object can be dragged, or perhaps we want to specify the time duration during which certain events may occur, with the exception of collision event. This can be achieved by using the escape mechanism described below.

9.3.15.1.5 ExclTimeContainers

Partially incorporated: excl time container and fill, endsync attributes on it. <excl> supports only one stream playing according to selection. Certain mappings may be possible where such content is played under an MPEG-4 switch node where the switch selection is by user interaction.

An example:

```
 <par>
<video id="vid" …/>
<excl>
  <par begin="englishBtn.click">
    <audio begin=vid.begin   src="english.wav" />
  </par>
  <par begin="frenchBtn.click">
    <audio begin=vid.begin   src="french.wav" />
  </par>
  <par begin="koreanBtn.click">
    <audio begin=vid.begin   src="korean.wav" />
  </par>
</excl>
</par>
```

In this example, multiple languages are available for a video. Only one language is played at a time with the most recent selection replacing the previous language choice, if any. The three par elements are children of the excl contained and so at most only one can play at any given time. The audio child in each par is defined to begin when the video begins. Each audio can be active (or heard) only when its parent time container is active; which begins with a click on a button media

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

object somewhere. This means that when the par begins, the audio will start playing at some point in the middle of the audio clip in sync with the video. I.e. the audio's actual computed begin time is coincident with the start of the video, but it cannot play (become active and be heard) until its parent is active, begun by a click, and so the actual observed start is some point into the audio as if it began playing when the video began. So the language track for the video can be switched and stay in sync with the video.

Note that the priorityClass element to control the pausing behavior of children of the excl will not be incorporated by XMT. The child elements of an excl will be considered to be peers with the default peers behavior "stop."

9.3.15.1.6 FillDefault

Fully incorporated: the fillDefault attribute defines the default value for the display behavior for an element when its active duration ends. The element may be frozen at the final state, or it may no longer be

presented (i.e., its effect is removed from the presentation).

9.3.15.1.7 MediaMarkerTiming

Fully incorporated: begin and end with media marker values allow elements to begin and end based upon media markers contained in the source content.

Example:

```
<audio begin="longvideo.marker(#video1)" … />
```

The audio will begin at the time of the named marker #video1 in the longvideo element. This allows synchronization between streaming media segments and other elements using named markers. The named markers can map to proposed SegmentDescriptors in MPEG-4.

9.3.15.1.8 MinMaxTiming

Fully incorporatd: min and max attributes provide the author with a way to control the lower and upper bound of the element's active play duration.

Example:

```
<video  end="foo.click" min="5s" max="20s" … />
```

The video will end when foo is clicked, but it will play at least for 5 seconds (even if foo is clicked earlier) but no more than 20 seconds (i.e., the video will end at 20 seconds even if foo is not clicked.

9.3.15.1.9 MultiArcTiming

Fully incorporated: the begin and end attributes may have multiple semicolon-separated values.

Example:

```
<par>
    <video begin="5s; foo.end; foo.click"  …/>
  </par>
```

The video will begin 5 seconds after the par begins, when "foo" ends, or when "foo" is clicked, whichever happens first.

9.3.15.1.10 RepeatTiming

Fully incorporated: repeatDur, repeatCount attritutes to allow repeating an element for a given duration or number of iterations.

Example:

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
<audio src=”sound.wav” dur=”2.5s” repeatDur=”7s” …/>
```

The audio will play for a total of 7 seconds. It will play fully two times, followed by a fractional part (.5) of the 2 seconds.

9.3.15.1.11 RepeatValueTiming

Fully incorporated: begin and end with repeat values to allow elements to begin and end in response to repeat events with a specific iteration value.

An Example:

```
<audio begin=”foo.repeat(3)” … />
```

Audio begins when the foo element repeats the third time.

9.3.15.1.12 RestartDefault

Fully incorporated: restartDefault attribute for specifying default restart semantics.

9.3.15.1.13 RestartTiming

Fully incorporated: restart attribute for controlling the begin behavior of an element that has previously begun.

Example:

```
<audio begin=”foo.click” dur=”3s” restart=”whenNotActive” … />
```

This example will begin repeating the audio clip and play for 3 seconds when “foo” is clicked but restart will be triggered only if the click occurs when this element is not active. The restart attribute value can also be “always,” or “never,” as per SMIL specification.

9.3.15.1.14 SyncBehavior

Partially incorporated: syncBehavior attribute specifies the runtime synchronization behavior. Note that syncTolerance is not incorporated by XMT.

Example:

```
<par>
  <text … />
  <par id=”speech” syncBehavior=”canSlip” >
    <video syncBehavior=”locked”… />
    <audio syncBehavior=”locked”…/>
  </par>
</par>
```

The audio and video elements are defined with hard or “locked” sync to maintain lip sync, but the “speech” par time container is allowed to slip. If either the video or audio must pause due to delivery problems, the entire “speech” par will pause, to keep the entire timeline in sync. However the rest of the document, including the animation element will continue to play normally.

SMIL runtime synchronization behavior syncBehavior=”canSlip | locked” would map to independent MPEG-4 OCRs and MPEG-4 streams using a common OCR respectively. The syncTolerance attribute will not be supported by XMT.

9.3.15.1.15 SyncBehaviorDefault

Partially incorporated: syncBehaviorDefault attribute to specify default synchronization behavior for elements and all descendents. The syncToleranceDefault will not be supported.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.3.15.1.16 SyncbaseTiming

Fully incorporated: begin and end with syncbase values to allow elements to begin and end relative to each other.

Example:

```
<par>
  <video id=vid ….  />
  <img begin=vid.begin  end=vid.end …. />
</par>
```

The img will begin and end at the same time as the video.

9.3.15.1.17 SyncMaster

Partially incorporated: syncMaster attribute to specify the synchronization master for a timeline.

SMIL syncMaster may be used in MPEG-4 to define the stream that supplies the OCR reference.

9.3.16 XMT-Ω Time Manipulations

9.3.16.1 SMIL TimeManipulations Module

Fully incorporated : accelerate, decelerate, autoReverse, and speed attributes to allow time manipulations.

Example:

```
<seq speed="2.0">
  <video dur="10s" …/>
  <video dur="10s" …/>
</seq>
```

In this example the entire contents of the sequence will play twice as fast. Each video child will play at twice the normal rate, and so will only last for 5 seconds.

Example:

```
<img … >
  <animateMotion dur="8s" accelerate=".25" decelerate=".25" … />
</img>
```

A motion path will accelerate up from a standstill over the first 2 seconds, run at a faster than normal rate for 4 seconds, and then decelerate smoothly to a stop during the last 2 seconds. This makes the motion animation look more realistic.

9.3.16.2 FlexTime Support

MPEG-4 FlexTime model is fully supported by the XMT-Ω timing model by extending the SMIL timing behavior with flexBehavior and flexBehaviorDefault attributes. The flexBehavior attribute specifies preferred modes of stretching or shrinking of the playback duration of an element in order to satisfy synchronization requirements to compensate for unpredictable delays in a delivery system, utilizing some amount of flexibility given by the author. Note, that whereas stretching or shrinking the duration of continuous media, e.g. for video, implies respectively slowing down or speeding up playback, for discrete media such as a still image, shrinking or stretching is merely adjusting the rendering period to be shorter or longer.

As such the flexBehavior attribute specifies whether (and if so how) the corresponding element can remain in a flexible synchronization relationship to the parent time container. The flexBehavior works in conjunction with the min and max attributes, where the min and max values provide lower and upper bounds on the active duration as specified in the SMIL

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

timing model. The flextime support in XMT allows the “flexed” element to adjust its playback speed according to the preferred modes of stretching or shrinking, while allowing the length of the active playback duration to be anywhere between the min and max.

We will call an element “flexed” if it specifies the flexBehavior attribute with non-default values, where default values are stretch/shrink mode=”normal” indicating that no stretching or shrinking is to be done.

Note that the flextime support does not guarantee synchronization. Flexibility on a flexed element is bounded by the min and max, and thus any delay or wait needs to be timed within the limits. For instance, if a network congestion delay occurs on a flexed media element, its rendering can wait using the min/max flexibility provided that the rest of the synchronization requirements can still be met. But if there is a delay beyond an allowable limit, synchronization will simply be broken. For guaranteed synchronization, syncBehavior attribute can be used, which describes for each element whether it must remain in a hard sync relationship to the parent time container. But syncBehavior does not alter the active duration of an element as flexBehavior does. It delays the parent time container begin time until it is ready to proceed.

9.3.16.2.1 The flexBehavior attribute

The flexBehavior attribute allows authors to specify preferred ways to stretch (slow down) or shrink (speed up) the active duration of an element and its descendents to satisfy synchronization requirements.

The default value of flexBehavior is “default” which means that the value is specified by flexBehaviorDefault. If flexBehaviorDefault is not specifed it defaults to the value of “inherit; inherit”. A document that defines neither of these attributes will have flexBehavior=”normal ; normal” behavior for all timed elements.

flexBehavior::= stretchBehaviorValueList “;” shrinkBehaviorValueList | “default”
A semicolon separated pair of two lists, the first is the stretchBehaviorValue list and the second is the shrinkBehaviorValueList.
stretchBehaviorValueList ::= stretchBehaviorValue (“,” stretchBehaviorValueList)?
A comma separated list of stretchBehaviorValues in preferred order.
shrinkBehaviorValueList ::= shrinkBehaviorValue (“,” shrinkBehaviorValueList)?
A comma separated list of shrinkBehaviorValues in preferred order.
stretchBehaviorValue ::= “hold” | “linear” | “repeat” |”normal”
“hold” will hold on the last frame, or freeze the element in order to increase the active duration, “linear” will slow down the duration linearly (e.g., reducing the video frame rate), “repeat” will repeat the element duration to increase its active duration, and “normal” signals XMT that there is no flexBehavior extension to the SMIL based XMT timing behavior.
shrinkBehaviorValue ::= “stop” | “linear” | “normal”
“stop” will stop the element’s active duration in order to speed it up, “linear” will speed up the active duration, linearly (e.g., increasing the video frame rate), “normal” signals XMT that there is no flexBehavior extension to the SMIL based XMT timing behavior.

Example:

```
<par>
  <audio …/>
  <video min=”5s” max=”10s” flexBehavior=”linear, hold; linear, stop” …/>
</par>
```

In this example, the video can slow down or speed up to increase or decrease the active duration if synchronization requirements cannot otherwise be met. It can slow down preferably by linearly adjusting the frame rate, or if linear stretching is difficult, it can freeze the last frame and hold on to it to increase the active duration. Similarly, it can reduce the active duration by linearly adjusting the frame rate, but if linear speed up is not possible, it can stop when necessary.

Flexed elements can map to MPEG-4 using TemporalGroup and TemporalTransform nodes.

Consider a slightly more complicated example:

```
<seq begin=”0s” flexBehavior=”linear, hold; linear, stop” min=”14s” max=”18s”…>
  <img   min=”4s” dur=”6s” max=”12s” …/>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    <par   …/>
      <audio  dur=”10s”  min=”8s”  max=”15s”….>
      <video  dur=”10s”  min=”8s”  max=”15s” ….>
    </par>
  </seq>
```

In this example, when all is well, the img begins when the seq time container begins, displays for 6 seconds, and the par container begins and immediately audio and video will begin and play for 10 seconds, after which the par container ends and so does the seq container. The total active duration of the seq container is then 6+ 10=16 seconds which is within the bounds of min and max on the container, and hence everything is fine.

But suppose that the seq container begins at the specified time at zero second, but the img is delayed by 2 seconds, i.e., it arrives 2 seconds after the seq container begins. Note that in this example the specified begin times of the img and the seq container coincide, i.e., the timeline of the img’s active duration starts running as the seq container becomes active. The player has two choices at this point.

One is to display the img for as close to “optimal” (as specified in dur) duration 6 seconds after its arrival (making its active duration to 8 seconds) and continue with the rest. This is possible since there still is a chance (by looking ahead to the par container) to complete the par container in 10 seconds and thus finish the outermost seq container in its max duration, 18 seconds. Alternatively, it can compensate for the 2 second delay as much as it can up-front, by cutting the playback duration of the img, in this case to 4 seconds after its arrival (making its active duration to 6 seconds), and continue with the par container as if there was no delay assuming that the audio and video are there in time.

Note that a player can make either choice, whether to compensate for a delay as early as possible or to as late as possible. Suppose our player decides to choose the latter approach to compensate for a delay as soon as it can. But to make a point let’s further suppose that the audio arrives in time but the video is late. At this point the player has to take yet another adaptive action. There are two things the player has to do: to hold onto the audio in the (decoder) buffer and wait for the video to arrive so the audio and the video can begin together in sync. Assuming that the amount of audio buffer has been set to its maximum requirement, the player will decided how long it can wait for the arrival of the video by respecting the following constraints:

the max duration to which the active duration of the img can be stretched,
the min duration of the video, and
the max duration of the seq container.

In our example, the img can be stretched to as long as the max 12 seconds. But if the img is stretched to display for 12 seconds, and since the seq container specifies its max duration to be 18 seconds, this leaves only 6 seconds for the par container, and this violates the min constraints specified for the audio and video, min=”8s.”

Clearly, the constraints should be solved consistently. One solution is: to stretch the img’s active duration to 10 seconds, allowing the video to arrive as late as 10 seconds into the seq container begin time, holding onto the audio while waiting. There are three cases we need to consider depending on when the late video arrives:

it arrives before the max wait time: adjust the video audio active duration accordingly and speed up the playback according to the flexBehavior values,

it arrives at exactly the max wait time, the video/audio active duration becomes the min duration and speed up the playback accordingly.

the wait time expires: at this point synchronization is broken. The audio will play for the min duration at 10 seconds and the par container will end when audio’s active duration ends. The flextime model behaves as if the video node starts at that time, but the player will not render the video and will discard the stream if it arrives later.

9.3.16.2.2 The flexBehaviorDefault attribute

The flexBehaviorDefault attribute specifies default flexBehavior behavior for elements and all descendents.

Example:

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
<par flexBehaviorDefault=”normal; normal” …/>
```

By setting the flexBehaviorDefault to be “normal; normal” the author can specify that the default flexBehavior behavior on the par and its descendants is no stretching/shrinking (unless otherwise specified).

Example:

```
<par flexBehaviorDefault=”linear, repeat, hold; linear, stop” …/>
```

In this example, the par container and all its descendents, unless otherwise specified, will be flexed according to the preferred stretch/shrink modes as specified.

Like fillDefault the flexBehaviorDefault also supports the value “inherit” so that stretchBehaviorList and shrinkBehaviorList are inherited. Eg flexBehaviorDefault=”inherit; linear, stop” will inherit stretchBehaviorList with new shrinkBehaviorList defined and flexBehaviorDefault=”inherit; inherit” will inherit both sets of lists etc. The default value of flexBehaviorDefault is “inherit; inherit”

9.3.17 XMT-Ω Transitions

SMIL Transitions feature comprises three modules; BasicTransitions, InlineTransitions and TransitionModifiers. XMT-Ω will include the features of all BasicTransitions and InlineTransitions modules only.

9.3.17.1 BasicTransitions

9.3.17.1.1 <transition>

The <transition> element is as per SMIL specification. The element defines a single transition class.

```
<transition
    id             = ID
    type           = <value>
    subtype        = <value>
    startProgress  = <value>
    endProgress    = <value>
    direction      = ”forward” | ”reverse”
    fadecolor      = <color value>
/>
```

id is an XML identifier.

type is a type, or family, of transition

subtype is the subtype of transition within the family

startProgress is the amount of progress through the transition at which to begin execution, specified as a number between 0.0 and 1.0

endProgress is the amount of progress through the transition at which to end execution, specified as a number between 0.0 and 1.0

direction is the direction in which the transition will run

fadecolor if the type of the transition is “fade” with subtype “fadeToColor” or “fadeFromColor” then this attribute specifies the starting or ending color of the fade.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.3.17.1.2 <param>

The <param> element is as per SMIL specification. The element supports an authoring tool to define its own custom transitions by providing a generic mechanism to supply parameters to the transition to control the effects.

```
<param
    id    = ID
    name  = <value>
    value = <value>
/>
```

id is an XML identifier.

name is the name of a parameter that controls an aspect of the transition effect

value is the value to supply to that named parameter

9.3.17.1.3 Applying transitions to media elements

The transIn and transOut attributes are incorporated as per SMIL specification. A media with a transition specify transIn will occur at the beginning of a media elements active duration whereas with transOut it will occur at the end.

9.3.17.2 InlineTransitions

9.3.17.2.1 <transitionFilter>

The <transitionFilter> element is as per SMIL specification. The element animates the progress of a filter behavior (transition).

```
<transitionFilter
    id            = ID
    type          = <value>
    subtype       = <value>
    mode          = "in" | "out"
    fadecolor     = <color value>
    from          = <value>
    to            = <value>
    by            = <value>
    values        = <list>
    calcMode      = "discrete | linear | paced"
    targetElement = IDREF
/>
```

id is an XML identifier.

type is a type, or family, of transition

subtype is the subtype of transition within the family

fadecolor if the type of the transition is “fade” with subtype “fadeToColor” or “fadeFromColor” then this attribute specifies the starting or ending color of the fade.

from is the amount of progress through the transition at which to begin execution, specified as a number between 0.0 and 1.0

to is the amount of progress through the transition at which to end execution, specified as a number between 0.0 and 1.0

by is a relative offset value for the amount of progress of the transition

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

values is a semicolon-separated list of one or more legal values specifying the progress of the transition.

calcMode specifies the interpolation mode for the progress of the transition.

targetElement is the identifier of the element containing the attribute to be animated.

9.3.17.3 Mapping Transitions to MPEG-4

Fade transitions can be mapped using transparency and color animation.

Slide and push wipes are mapped by animating the position of the object(s).

Other effects can be achieved by animating with texture Transformations, Layer2D/3D, MatteTexture etc., potentially even in combinations.

9.3.18 XMT-Ω Tool considerations

MPEG-4 authoring format includes notation to allow hints for compression or other format conversions where needed. For example video compression may define compression parameters that allow frame rate and/or quality hints to be specified by the author for various bandwidths. This allows, for example, frame rate to be preferred over quality or vice versa. Video may also be converted to scalable layers. Images may not only be compressed but hints could be given to crop them to reduce size as appropriate.

Such authoring could result in either run-time or compile-time alternatives as also discussed in Content Control above. Such selection of mappings during compilation (encoding to MPEG-4) may be done with interactive prompts during the compilation phase, or may be based on project settings, and in conjunction with test attributes whose values can be specified in the <head> of the document.

9.3.19 XMT-Ω Examples

This section contains examples of the proposed high-level format and a mapping into MPEG-4 nodes.

9.3.19.1 Circle with finite duration color animation on mouse press

The following example shows a Circle whose color changes over a 2s duration beginning when the mouse button is pressed down.

Using the XMT-Ω format the circle is simply defined and is given an animate section that describes a linear interpolation using three color values that starts on a mouseDown event and has duration of 2s.

```
<circle id="myCircle" radius="20"
  <transformation
        visibility="true"
        translation="24 50"/>
  <material color="#ee0000" filled="true">
     <animateColor attributeName="color" dur="2s"
           begin="myCircle.click"
           values="#ee0000; #ffcc45; #ffffff" keyTimes="0; 0.3; 1"
           calcMode="linear" />
  </material>
</circle>
```

Using a VRML-like syntax the MPEG-4 nodes to map the XMT-Ω format description above could be done as follows. First there is a switch (MPEG-4 Switch as apposed to SMIL <switch>) so the entire object can be hidden or shown (visibility=true attribute above). The Circle is the basic pattern of Shape containing appearance and geometry, where geometry is a Circle and appearance contains a Material2D describing the color. The Circle (Shape) is set under a Transform2D to position it. To sense mouse activity we need a touchSensor and to define the duration of the color change we need a TimeSensor. Then there is a ColorInterpolator for the color changing. To make the behavior work we must route the TouchSensor to the TimeSensor, to start the TimeSensor when the mouse is pressed. Then we route the

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

fractional output of the TimeSensor to the input of the ColorInterpolator; and finally the output of the ColorInterpolator to the emissiveColor field of the Circles Material2D. Not forgetting of course to DEF the requisite nodes so we can setup the routes.

```
Switch  {
        whichchoice 0
        choice  [
            Transform2D  {
                translation 24 50
                children [
                    Shape {
                        appearance Appearance {
                            material DEF CircleMat Material2D {
                               emissiveColor 0.93 0.0 0.0
                               filled TRUE
                            }
                         }
                        geometry Circle {
                           radius 20
                        }
                    }

                  DEF Touch TouchSensor {}
                    DEF Timer  TimeSensor {
                        cycleInterval 2
                    }
                    DEF Coloring ColorInterpolator {
                         key [0.0 0.3 1.0]
                           keyValue [ 0.93 0.0   0.0,
                                      1.0  0.93 0.27,
                                      1.0  1.0   1.0 ]
                      }
                ]
            }
        ]
    }

Route Touch.touchTime to Timer.startTime
Route Timer.fraction_changed to Coloring.set_fraction
Route Coloring.value_Changed to CircleMat.emissiveColor
```

9.3.19.2 An example to show BIFS, OD and media stream mapping

```
<seq begin=”20s” dur=”10s”>
    <img id=”Image1” dur=”5s”  />
    <img id=”Image2” dur=”2s”  />
    <img id=”Image3”  />
</seq>
```

The above example shows a segment of a presentation that begins at 20s and has three images in shown one after the other. The first image starts at t=20s and lasts for the specified duration of 5s. At t=25s the second starts and lasts for 2s. The third and final image starts at t=27s and has its duration controlled by the overall <seq> time container that specifies the sequence lasts for 10s. As the first and second images consume 7s of the 10s there are 3s left for the third image.

This is mapped to the following MPEG-4 BIFS, OD and media streams. The mapping shows the sequence of commands but the detail of the nodes contained in the BIFS has been omitted for clarity.

t=20s OD Update for Image1 media stream
t=20s Media data for Image1
t=20s BIFS Command to insert nodes for Image1
t=25s OD Delete for Image1 media stream and OD Update for Image2 media stream

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

t=25s Media data for Image2
t=25s BIFS Command to delete nodes for Image1 and insert nodes for Image2
t=27s OD Delete for Image2 media stream and OD Update for Image3 media stream
t=27s Media data for Image3
t=27s BIFS Command to delete nodes for Image2 and insert nodes for Image3
t=30s BIFS Command to delete nodes for Image3
t=30s OD Delete for Image3 media stream

Note that the timing above assumes instantaneous delivery and that timing would require modification by an MPEG-4 server to assure that media data transmission times are allowed for.

9.3.20 XMT-Ω Data Types

9.3.20.1 Basic data types

Atrributes may be single values, a list of single values or a list of lists. A single value for example is a float, or a single compound datatype for example color, which is three floats, and translation which is two or three floats for 2D and 3D contexts respectively. A list is set of single values, where each value is separated by a semi-colon from the subsequent value. A list of lists is a set of lists, where each list is separated by a colon from the subsequent list.

9.3.20.1.1 Color

An XMT-Ω color value can be specified in one of the following forms

As a list of three floating point values each in the range 0.0 to 1.0, e.g. “1.0 0.5 0.75”. This is the same format as used in XMT-A

In the #RRGGBB hex-style form as is commonly used in html pages to specify color e.g. “#FF80C0”

As one of the standard color names from either the list of HTML colors defined by the W3C or from the set as defined by the XConsortium that includes the extended set of HTML colors used by the popular web browsers e.g. “red”, “peachpuff”, “LightGoldenRod”. In XMT-Ω no spaces are allowed in the names and the names are case-insensitive. The complete list of XConsortium defined color names can be found in the /usr/lib/X11/rgb.txt supplied with XWindows. This includes alternate spellings of “grey”, “gray” and colors with numeric suffices from 1 to 4 e.g. “blue1”, plus the complete range of gray from 1 through 100, e.g. “gray67”. Note: In the standard set of HTML colors there are four names that are also defined by XConsortium and the colors are different. For these identically named colors the HTML color value is used; the colors being purple, maroon, green and gray.

9.3.20.1.2 Angles

Angles are specified, by default, in degrees with a positive value resulting in anti-clockwise rotation. The suffix deg for degrees is optional but may be explicitly specified. Angles may also be specified in grad and radians using the suffices grad and rad respectively. The following examples all show a rotation of 90 degrees.

rotation = ”90”
rotation = ”90deg”
rotation = ”1.571rad”
rotation = “100grad”

9.3.20.1.3 Strings

Many attributes are specified as a sequence of characters – a string. For all cases other than those listed below the string can be entered directly as an attribute with no special treatment. However for attributes that may contain more than one string, where each string may contain spaces, each string must be surrounded by ". The attributes where such strings can occur are *textLines* attribute in <string> and <subtitles>, and *family* attribute from <fontstyle>. Quotes within a string must be escaped as per the example below.

textLines=”"Line one"”

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

textLines=""First line"; "A second line""
textLines=""He said, \"A string with an embedded quotes\" that must be escaped.""

9.3.21 XMT-Ω Node coverage

This section tabulates the MPEG-4 nodes and how they are/might be used in mapping from XMT-Ω to MPEG-4. This table is mainly for illustration and give an idea of what's covered by XMT-Ω. The actual use and mapping are subject to change and implementation variance (XMT-Ω defines intent and allows implementation some choice in handling the 'compilation into MPEG-4).

AcousticMaterial	
AcousticScene	
Anchor	Used in mapping <a> linking module
AnimationStream	May be used by Animation module
Appearance	Implicitly created when defining visual media elements
ApplicationWindow	Used to map <applicationWindow>
AudioBuffer	
AudioClip	Used to map <audioClip>
AudioDelay	
AudioFX	
AudioMix	
AudioSource	Used when mapping <audio>
AudioSwitch	
Background	Used to map <backdrop> and <region> or <group> backgroundColor
Background2D	Used to map <background> and <region> or <group> backgroundColor
BAP	
BDP	
Billboard	Part of <transformation> and <group>
Bitmap	Used to map <img> and <video>
Body	
BodyDefTable	
BodySegmentConnectionHint	
Box	Used to map <box>
Circle	Used to map <circle>
Collision	Part of <group>
Color	Use to map <lines>, <polygons>
ColorInterpolator	May be used by Animation module
CompositeTexture2D	
CompositeTexture3D	
Conditional	May be used by Animation module
Cone	Used to map <cone>
Coordinate	Use to map <points>, <lines>, <mesh>
Coordinate2D	Use to map <points>, <lines>, <polygons>
CoordinateInterpolator	May be used by Animation module
CoordinateInterpolator2D	May be used by Animation module
Curve2D	Used to map <curve>
Cylinder	Used to map <cylinder>
CylinderSensor	May be used by Animation module
DirectiveSound	
DiscSensor	May be used by Animation module
DirectionalLight	Used to map <light>
ElevationGrid	
Expression	
Extrusion	
Face	
FaceDefMesh	
FaceDefTables	
FaceDefTransform	

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

FAP	
FDP	
FIT	
Fog	Used to map <fog>
FontStyle	Used to map font styles for <string>
Form	
Group	Part of <transformation> and <group>
Hierarchical3DMesh	
ImageTexture	Used for <img> and <texture>
IndexedFaceSet	Used to map <mesh>
IndexedFaceSet2D	Used to map <polygons>
IndexedLineSet	Used to map <lines>
IndexedLineSet2D	Used to map <lines>
Inline	Used to map <inline>
InputSensor	May be used for keyboard input (accessKey timing)
Layer2D	Used to map <region> and <group> when used with size attribute
Layer3D	Used to map <region> and <group> when used with size attribute
Layout	
LineProperties	Used to map <outline>
ListeningPoint	
LOD	
Material	Used to map <material>
Material2D	Used to map <material>
MaterialKey	Used to map <chromakey>
MatteTexture	May be used for mapping of some transition effects
MediaBuffer	May be used to map <prefetch>
MediaControl	May be used for time manipulations and interactively played media
MediaSensor	May be used for MediaClipMarker module mapping
MovieTexture	Used for <video> and <texture>
NavigationInfo	
Normal	Used to map <mesh>
NormalInterpolator	May be used by Animation module
OrderedGroup	Part of <transformation> and <group>
OrientationInterpolator	May be used by Animation module
PerceptualParameters	
PixelTexture	May be used to map <img>
PlaneSensor	May be used by Animation module
PlaneSensor2D	May be used by Animation module
PointLight	Used to map <pointLight>
PointSet	Used to map <points>
PointSet2D	Used to map <points>
PositionInterpolator	May be used by Animation module
PositionInterpolator2D	May be used by Animation module
Proto	
ProximitySensor	May be used by Animation module (event source for animation)
ProximitySensor2D	May be used by Animation module (event source for animation)
QuantizationParameter	May be used to achieve higher compression for mapping
Rectangle	Used to map <rectangle>
ScalarInterpolator	May be used by Animation module
Script	May be used by Animation module for complex animations
Shape	Implicitly created when defining visual media elements
Sound	Used to map <audio>
Sound2D	Used to map <audio>
Sphere	Used to map <sphere>
SphereSensor	May be used by Animation module
SpotLight	Used to map <spotLight>
Switch	Part of <transformation> and <group> (used to map visibily attribute.) Also for <excl>

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TermCap	My be used for some Content Control module functionality mapping
Text	Used to map <text>, <string> and <subtitles>
TemporalGroup	Used to map time containers and flexible time manipulations
TemporalTransform	Used to map time manipuations and flexible time manipulations
TextureCoordinate	Used to map <polygons> and <mesh>
TextureTransform	Used to map <texture>
TimeSensor	May be used by Animation module (event source for animation). And for timing
TouchSensor	May be used by Animation module (event source for animation)
Transform	Part of <transformation> and <group>
Transform2D	Part of <transformation> and <group>
Valuator	May be used by Animation module
ViewPoint	
Viseme	
VisibilitySensor	May be used by Animation module (event source for animation)
WorldInfo	Maps small part of RDF

9.4 XMT-C Modules

9.4.1 MPEG-7

The MPEG-7 standard also known as "Multimedia Content Description Interface" aims at providing standardized core technologies allowing description of audiovisual data content in multimedia environments.

In order to achieve this broad goal, MPEG-7 standardizes:

Descriptors (D): representations of Features, that define the syntax and the semantics of each feature representation,

Description Schemes (DS), that specify the structure and semantics of the relationships between their components, which may be both Ds and DSs,

A Description Definition Language (DDL), to allow the creation of new DSs and, possibly, Ds and to allows the extension and modification of existing DSs,

System tools, to support multiplexing of description, synchronization issues, transmission mechanisms, file format, etc.

The following example illustrates the use of MPEG-7 descriptions within XMT. The example has been simplified for readability by removing necessary namespace qualifiers.

```
<seq begin=20s dur=10s>
      <img id=”Image1” dur=5s>
         <StructuredAnnotation>
           <Who>Fernado Morientes</Who>
                <When>Spain Sweden soccer match</When>
                <TextAnnotation xml:lang=’en-us’>
                       This was the first goal of this match.
                </TextAnnotation>
         </StructuredAnnotation>
      </img>
       <img id=”Image2” dur=2s>
         <Creation>
                <Creator>
           <role>presenter</role>
           <Individual>
               <GivenName>Ana</GivenName>
                             <LastName>Blanco</LastName>
                       </Individual>
                </Creator>
         </Creation>
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        </img>
         <img id=”Image3”>
           <Histogram HistogramNormFactor="1000">
             <HistogramValues>400 100 200 150 150 </HistogramValues>
           </Histogram>
        </img>
    </seq>
```

9.4.2 XMT-C Encoding, Delivery and Publication hints

One of the most important functionality XMT should provide is a giving a way of communication between the authoring entities and the bitstream generation entities. Since MPEG-4 scene will be composed of various media, number of encoders and bitstream generation tools will be involved to produce final bitstream. Therefore, bitstream generation entities including media encoders should be fully under the control of the authoring entities to generate the presentation as they wishes. Encoding Hints and Bitstream Generation Hints in this document will provide a way for the authoring entities to give their intent to the bitstream generation entities.

9.4.2.1 <EncodingHints>

Encoding hints are composed of sourceFormat and targetFormat. sourceFormat will contain the information about the raw media should be encoded by the media encoders. The targetFormat will hint the attributes about the bitstream generated by the media encoders.

```
<EncodingHints>
  <sourceFormat>
    <param name=”” value=””/>
    <param name=”” value=””/>
     . . .
  </sourceFormat>
  <targetFormat>
    <param name=”” value=””/>
    <param name=”” value=””/>
     . . .
  </targetFormat>
<EncodingHints />
```

9.4.2.1.1 <sourceFormat>

The following common <param> name value pairs have been predefined to promote interchangeability of encoding hints. Other name value pairs may be used as required to qualify aspects of source format.

name=”mimeType”

used to assist identification of media type for the source media. The value is the mimeType e.g value=”video/h263”

name=”colorFormat”

used to specify color format of the visual media. The value is the colorFormat e.g value=”YUV422”. Other values are YUV420, YUV411, YUV9, YUV12, RGB24, RGB16, RGB32 etc

name=”size”

used to specify the size of a visual source media, eg, value=”w, h” where w,h is the width and height of the visual media.

name=”length”

used to specify the length in time of a continuous source media, eg, value=”10000.89ms”. Units may be expressed in ms (milliseconds) or s (seconds)

name=”cropSize”

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

used to select a rectangular section from a visual media stream, e.g a video or image. The value is area to include as the target media value=”x, y, w, h” where x,y is a point relative to the top left corner of the visual media with a width and height. This may be used to crop a video during encoding. If no cropSize is specified the entire source visual is used.

name=”cropLength”

used to specify a sub-section of a continuous source media, eg, value=”1000ms, 2000.5ms”. Units may be expressed in ms (milliseconds) or s (seconds) and are the beginning and end of the sub-section. If no cropLength is specified the entire source visual is used.

9.4.2.1.2 <targetFormat>

The following common <param> name value pairs have been predefined to promote interchangeability of encoding hints. Other name value pairs may be used as required to qualify aspects of target format.

name=”size”

used to specify the size of the target visual media, eg, value=”w, h” where w,h is the width and height of the visual media.

name=”rateControl”

used to specify if rate control is used for the encoding process, eg. for video. value=”off” or “on”. Instead of “on” a particular form of rate control method can be specified if multiple mechanisms are supported, e.g. “TM5”.

name=”avgBitrate”

used to specify target average bit rate from the encoding process.

name=”maxBitrate”

used to specify target maximum bit rate from the encoding process.

name=”maxAUSize”

used to specify target maximum access unit size for the encoding process.

9.4.2.1.3 <BIFSEncodingHints>

The <BIFSEncodingHints> hints are provided principally to control the encoding process for BIFS to ensure that there can be a deterministic mapping for conformance purposes. In order to perform conformance tests for XMT-A the support of the <BIFSEncodingHints> is required and the encoding hint must be used by the encoding process wherever the BIFS provides a coding choice.

```
<BIFSEncodingHints
  SFNodeCoding   =”auto|mask|list”
  MFListCoding   =”auto|list|vector”
  SFScriptCoding =”auto|list|vector”
  ROUTESCoding   =”auto|list|vector”
/>
```

9.4.2.2 <DeliveryHints>

Delivery hints are to provide XMT with information that can be used to control or influence delivery of the MPEG-4 streams. Delivery hints are name value pairs in <param> element.

The following common <param> name value pairs have been predefined to promote interchangeability of encoding hints. Other name value pairs may be used as required to qualify aspects of source format.

name=”carouselRate”

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

used to specify the rate of carousel insertions into BIFS and OD streams to provide random access points for multicast

name=”carouselPercent”

used to specify the percent of total bandwidth to be used for carousel.

name=”maxPacketSize”

maximum packet size for delivery (transport).

name=”totalBandwidth”

total bandwidth of transport.

9.4.2.2.1 M4Mux

XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
   <xs:simpleType name="bit1">
      <xs:restriction base="xs:unsignedByte">
         <xs:maxInclusive value="1"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="bit3">
      <xs:restriction base="xs:unsignedByte">
         <xs:maxInclusive value="7"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="bit4">
      <xs:restriction base="xs:unsignedByte">
         <xs:maxInclusive value="15"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="bit5">
      <xs:restriction base="xs:unsignedByte">
         <xs:maxInclusive value="31"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="bit8">
      <xs:restriction base="xs:unsignedByte"/>
   </xs:simpleType>
   <xs:simpleType name="bit24">
      <xs:restriction base="xs:unsignedInt">
         <xs:maxInclusive value="16777215"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="bit16">
      <xs:restriction base="xs:unsignedShort"/>
   </xs:simpleType>
   <xs:simpleType name="bit32">
      <xs:restriction base="xs:unsignedInt"/>
   </xs:simpleType>
   <xs:annotation>
      <xs:documentation>
 ************************************************************
 *                                                          *
 *   Declaration of the M4Mux descriptors          *
 *                                                          *
 ************************************************************
</xs:documentation>
   </xs:annotation>
   <xs:complexType name="M4MUXDescriptorType">
```

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    <xs:complexContent>
      <xs:extension base="MandatoryM4MUXDescriptorType">
        <xs:all>
          <xs:element ref="M4muxTimingDescriptor" minOccurs="0"/>
          <xs:element ref="M4muxChannelDescriptor" minOccurs="0"/>
          <xs:element ref="M4muxBufferSizeDescriptor" minOccurs="0"/>
          <xs:element ref="M4muxCodeTableDescriptor" minOccurs="0"/>
        </xs:all>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="MandatoryM4MUXDescriptorType">
    <xs:all>
      <xs:element ref="M4muxIdentDescriptor" minOccurs="0"/>
    </xs:all>
  </xs:complexType>
  <xs:element name="M4muxChannelDescriptor">
    <xs:complexType>
        <xs:sequence>
      <xs:element ref="FMCMuxChannel" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="version_number" type="bit5" use="required"/>
      <xs:attribute name="current_next_indicator" type="bit1" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="FMCMuxChannel">
    <xs:complexType>
      <xs:attribute name="ES_ID" type="bit16" use="required"/>
      <xs:attribute name="M4MuxChannel" type="bit8" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="M4muxTimingDescriptor">
    <xs:complexType>
      <xs:attribute name="FCR_ES_ID" type="bit16" use="required"/>
      <xs:attribute name="FCRResolution" type="bit32" use="required"/>
      <xs:attribute name="FCRLength" use="required">
        <xs:simpleType>
          <xs:restriction base="bit8">
            <xs:maxInclusive value="64"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="FMXRateLength" use="required">
        <xs:simpleType>
          <xs:restriction base="bit8">
            <xs:maxInclusive value="32"/>
            <xs:minInclusive value="1"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="M4muxBufferSizeDescriptor">
    <xs:complexType>
      <xs:all>
        <xs:element ref="DefaultM4MuxBufferDescriptor"/>
        <xs:element ref="M4MuxBufferDescriptor"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="DefaultM4MuxBufferDescriptor">
    <xs:complexType>
      <xs:attribute name="FB_DefaultBufferSize" type="bit24" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="M4MuxBufferDescriptor">
    <xs:complexType>
      <xs:sequence>
```

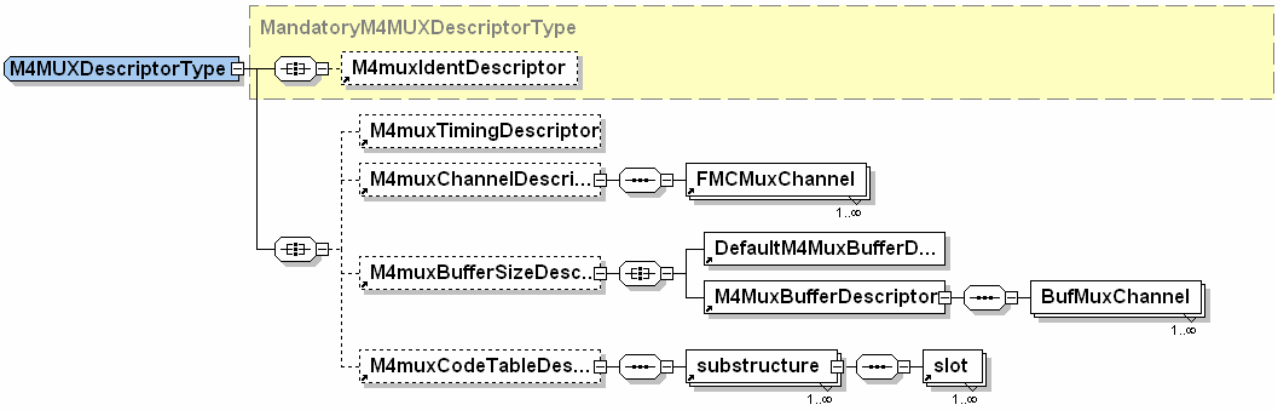

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

```
        <xs:element name="BufMuxChannel" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="M4MuxChannel" type="bit8" use="required"/>
            <xs:attribute name="FB_BufferSize" type="bit24" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="M4muxIdentDescriptor">
    <xs:complexType>
      <xs:attribute name="MuxID" type="bit8" use="required"/>
      <xs:attribute name="MuxType" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="0"/>
            <xs:enumeration value="1"/>
            <xs:enumeration value="8"/>
            <xs:enumeration value="9"/>
            <xs:enumeration value="10"/>
            <xs:enumeration value="11"/>
            <xs:enumeration value="12"/>
            <xs:enumeration value="13"/>
            <xs:enumeration value="14"/>
            <xs:enumeration value="15"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="Muxmanagement" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="0"/>
            <xs:enumeration value="1"/>
            <xs:enumeration value="8"/>
            <xs:enumeration value="9"/>
            <xs:enumeration value="10"/>
            <xs:enumeration value="11"/>
            <xs:enumeration value="12"/>
            <xs:enumeration value="13"/>
            <xs:enumeration value="14"/>
            <xs:enumeration value="15"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="M4muxCodeTableDescriptor">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="substructure" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="MuxCode" type="bit4" use="required"/>
      <xs:attribute name="version" type="bit4" use="required"/>
      <xs:attribute name="substructureCount" type="bit8" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="substructure">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="slot" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="slotCount" type="bit5" use="required"/>
      <xs:attribute name="repetitionCount" type="bit3" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="slot">
    <xs:complexType>
      <xs:attribute name="M4MuxChannel" type="bit8" use="required"/>
```

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        <xs:attribute name="numberOfBytes" type="bit8" use="required"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="M4MUXTable">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="M4MUXDescriptor" type="M4MUXDescriptorType" minOccurs="0"
maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```



Example of instance

```
<?xml version="1.0"?>
<!—M4MUX sample Instance -->
<M4MUXTable xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="XMT_m4mux_schema.xsd">
   <M4MUXDescriptor>
      <M4muxIdentDescriptor MuxID="1" MuxType="0" Muxmanagement="0"/>
      <M4muxTimingDescriptor FCR_ES_ID="36" FCRResolution="90000" FCRLength="16"
FMXRateLength="8"/>
      <M4muxChannelDescriptor version_number="1" current_next_indicator="1">
         <FMCMuxChannel ES_ID="2" FlexMuxChannel="10"/>
         <!-- ES 2 in channel 10 -->
         <FMCMuxChannel ES_ID="3" FlexMuxChannel="12"/>
         <!-- ES 3 in channel 12 -->
      </M4muxChannelDescriptor>
      <M4muxBufferSizeDescriptor>
         <DefaultFlexMuxBufferDescriptor FB_DefaultBufferSize="36000"/>
         <FlexMuxBufferDescriptor>
            <BufMuxChannel FlexMuxChannel="10" FB_BufferSize="40000"/>
            <!-- Buffer associated with channel  10 -->
            <BufMuxChannel FlexMuxChannel="12" FB_BufferSize="50000"/>
         </FlexMuxBufferDescriptor>
      </M4muxBufferSizeDescriptor>
      <M4muxCodeTableDescriptor MuxCode="2" version="3" substructureCount="2">
         <!--Number of sub-structures    -->
         <substructure slotCount="3" repetitionCount="1">
            <!-- Number of slots -->
            <slot flexMuxChannel="1" numberOfBytes="18"/>
            <slot flexMuxChannel="4" numberOfBytes="12"/>
            <slot flexMuxChannel="5" numberOfBytes="13"/>
         </substructure>
         <substructure slotCount="2" repetitionCount="0">
            <slot flexMuxChannel="10" numberOfBytes="18"/>
            <slot flexMuxChannel="12" numberOfBytes="12"/>
         </substructure>
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

```
    </M4muxCodeTableDescriptor>
  </M4MUXDescriptor>
  <!--another m4mux-->
  <M4MUXDescriptor>
    <M4muxIdentDescriptor MuxID="2" MuxType="0" Muxmanagement="0"/>
    <M4muxTimingDescriptor FCR_ES_ID="36" FCRResolution="90000" FCRLength="16"
FMXRateLength="8"/>
    <!--...-->
  </M4MUXDescriptor>
</M4MUXTable>
```

9.4.2.3 <PublicationHints>

Publication hints provide a mechanism to convert a generic authoring time placeholder for a stream into a specific reference for the media once it is deployed. When creating an mp4 file most streams will probably be referenced directly by Object Descriptor IDs with descriptors as necessary in the file. However if URLs are used then publication hint can be used to replace a generic placeholder by an actual url reference for when the content is created (encoded).

```
<PublicationHints>
  <param name="__pubhint1__"
         value="rtsp://www.somewhere.com/apath/bpath">
</PublicationHints>
```

In the above example the string "__pubhint1__" will be replaced by "rtsp://www.somewhere.com/apath/bpath" where it occurs in URI references.

9.4.3 XMT-C <authoring> Element

The <authoring> element provides a means to identify, at authoring time, additional information to facilitate authoring and exchange of information between authors and authoring tools. Authoring elements are defined for the following purposes:

specifying sets of elements to allow the grouping of objects into semantically coherent sets; these sets may be orthogonal to scene hierarchy.

adding properties to objects: these properties can be defined by the author

The <authoring> element contains one or more of the <metaSet> and <metaProperty> elements. The <authoring> element can be placed in any XMT element.

```
<authoring id=""
   scope="local" | "subtree"
/>
```

The scope attribute allows the semantics to be restricted to the element where the <authoring> is defined (local) or the element plus all its children. Using such rules an element may have inherited metaProperty's etc as well as ones that are defined inside that element. If an element would thus have multiple metaProperty's with the same name the one that is defined closest to the element (the closest in scope) shall be is regarded as the correct value.

9.4.3.1 Specifying a <metaSet> of elements

```
<metaSet id="" description="" set="" />
```

This element defines a set. The description is intended to be used to identify the purpose of the set. All elements belonging to this set will use the attribute set="" to refer to their membership in one or more sets.

All XMT-A elements can contain the set="" attribute. The attribute may contain a list of ids when the element belongs to more than one set. Sets can be made up of any elements including nodes, and/or object descriptors etc. The <metaSet> element itself also contains the set attribute and hence sets of sets can be created.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A set is whatever an author and/or authoring tool deem relevant. Hence the definition of a set is up to the author and/or authoring tool alone. Certain sets may have meaning to a specific authoring tool and/or author. When files are exchanged from one authoring tool to another authoring tool, if the latter does not provide the same 'higher-level' support for the set(s) they should, in any case, be preserved by that tool. Hence if the file is edited again using the former tool the set will still be there (its relevance may have been altered however; e.g. if the set was a set of all red shapes and a new one was added by the latter tool the set would not have been updated to reflect that fact if the latter tool did not understand the higher meaning or the author did not 'manually' update the set – assuming the latter tool provided such a facility).

The following example shows a set defined by the author of related colors. An authoring tool may allow the author to vary the intensity or all related colors by the same amount. The example includes the second element belonging to two sets, the related colors set and also the thick Lines set.

```
<metaSet id="relatedColors" description="Colors of related objects" />

<Shape>
...
   <Material2D set="relatedColors" ...
<Shape DEF="SecondShape">
...
   <Material2D set="relatedColors, thickLines" ...
```

9.4.3.2 Specifying a <metaProperty>

```
<metaProperty id="" name="" value="" />
```

A metaProperty is an additional author and/or authoring tool defined property for any element. The element for which the extra property is being defined will contain this element.

The example shows a **PositionInterpolator** where two additional authoring properties have been defined

```
<PositionInterpolator ...>
  <authoring>
    <metaProperty name="movement"  value="spiral"/>
    <metaProperty name="direction" value="clockwise"/>
  </authoring>
</PositionInterpolator>
```

9.5 XMT Schemas

The current schemas for XMT can be found in the separate electronic attachment to this document.

There are three parts to the XMT schema: XMT-A, XMT-O and XMT-C. The respective schemas can be found in the zip file in folders that are likewise named.

9.6 Informative: XMT/X3D Compatibility

The current schemas for XMT can be found in the separate electronic attachment to this document: The XMT-A representation follows the same rules as the compromise form of X3D and hence is compatible with it. MPEG-4 adds to this representation some extra attributes and elements for deterministic binary encoding and to augment authoring. However, these extra attributes and elements are optional.

An X3D document would go directly into a <Scene> element. MPEG-4 consists of a <Header> and a <Body> which contains the <Replace><Scene> BIFS command. This is because MPEG-4 can carry many media streams and can dynamically update the BIFS, and thus the MPEG-4 <Scene> element holds the MPEG-4 representation of all the BIFS, commands, OD framework etc. inside the <Replace> command.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The table below compares X3D and MPEG-4 representation to illustrate the high degree of compatibility and the small amount of change to go from X3D to MPEG-4 or vice versa (within the subset of elements that is contained in both standards of course).

X3D	XMT-A

<Header>
  <meta>
  </meta>
</Header>
<Scene>
  <!-- The scene contents -->
</Scene>
	<Header>
	  <meta>
	  </meta>
	  <InitialObjectDescriptor/>
	</Header>
	<Body>
	  <Replace>
	  <Scene>
	  <!-- The scene contents -->
	  </Scene>
	  </Replace>
	</Body>

To fully convert the document from X3D to XMT-A, or vice versa, the outer <X3D> or <XMT-A> element with schema namespace reference will need to be altered accordingly,

Note: X3D <Scene> does not need to have a <Group> at the top level, while MPEG-4 requires a top-level node such as <Group>, <OrderedGroup>, <Layer2D> or <Layer3D> as the root of the scene graph. If the X3D scene does not have a single <Group> as the root it will be also be necessary to add this when converting to XMT-A. Note that X3D image, video and audio sources are referred directly by urls. While MPEG-4 can express the urls in an identical manner it is more likely that a conversion would create ObjectDescriptors for these media types and replace the source url references by ObjectDescriptor Ids.

Note: X3D allows commas as white space, using them as visual separators. XMT does not allow the use of commas as white space.

9.7 Informative: The usage of XMT-A BitWrapper element in authoring side

There are 4 cases using XMT-A representation of BitWrapper node. As described in subclause 7.2.2.23.2, the functionality of BitWrapper node is to encode data for the **node** field and to transmit an encoded bitstream through a scene description stream (BIFS stream) or a separate stream. The **url** field carries the out-band (=separate) bitstream and the buffer field carries the in-band bitstream such as BIFS bitstream.

For each scenario (in-band and out-band), authors can use the encoding parameters for encoding the original data that is contained in the **node** field of the BitWrapper node. Alternatively the author simply directly specifies the file name (or object descriptor ID) of an encoded bitstream in **url** or **buffer** field of the BitWrapper node if he already has an encoded bitstream.

Here, it will show how authors can create XMT-A format files (XMT), which contains BitWrapper element in order to carry an encoded bitstream in each case and generate mp4 files from XMT-A format files.

9.7.1 Example: Using BitWrapper and pre-encoded bitstream

One can create the XMT input file that has BitWrapper node and pre-encoded bitstream in it. From this input file, one also can generate a mp4 file. The whole processes are explained in the following figure.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

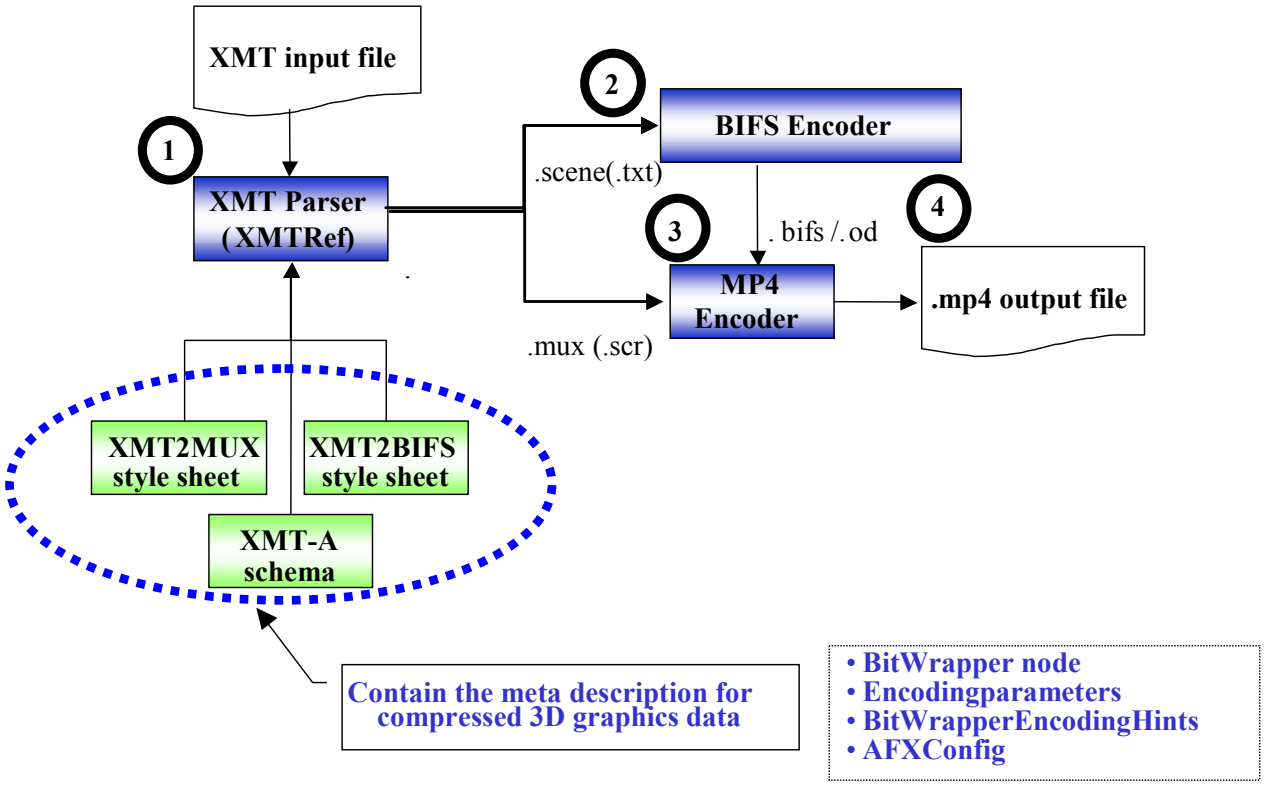


Figure 60 — XMT to mp4: Using BitWrapper and pre-encoded bitstream

The XMT parser takes a XMT file as an input and produces two output files with the help of XMT-A schema description and two XMT style sheets (XMT2BIFS, XMT2MUX) in the number 1(XMTParser) of the figure. Those two output files (.scene and .mux) can be used as input files of BIFS encoder and MP4 encoder respectively in the number 2(BIFS Encoder) and 3(MP4 Encoder). Finally, MP4 encoder produces a mp4 file with the additional input files from BIFS encoder (.bifs and .od) in the number 4(.mp4 output file).

The following two examples show how the above processes are applied in real situation with the MPEG-4 reference software.

9.7.1.1 In-band scenario: Using pre-encoded bitstream

One can create a text file and a script file (Both are IM1 textual format) that carry encoded in-band bitstream. The encoded bitstream is already made and provided to authors. Therefore, one does not need to have original data in the **node** field and its encoding parameters in BitWrapper node. In this case, the name of file given by the **buffer** field indicates the pre-encoded bitstream.

In “ICOriBufferWithoutEP.xmt”,

```
<Header>
   <InitialObjectDescriptor objectDescriptorID="1" binaryID="1" >
    …
      <Descr>
         <esDescr>
            <ES_Descriptor ES_ID="xyz1" binaryID="301">
               <decConfigDescr >
                  <DecoderConfigDescriptor     streamType="3" objectTypeIndication="2"
bufferSizeDB="500000">
                     <decSpecificInfo>
                        <BIFSv2Config nodeIDbits="10" routeIDbits="10" PROTOIDbits="10">
                        </BIFSv2Config>
                     </decSpecificInfo>
```


Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

```
               </DecoderConfigDescriptor>
            </decConfigDescr>
            <slConfigDescr>
               <SLConfigDescriptor>
                  <custom useAccessUnitStartFlag="TRUE" useAccessUnitEndFlag="TRUE"
                          timeStampResolution="100" timeStampLength="14" />
               </SLConfigDescriptor>
            </slConfigDescr>
            <StreamSource url="ICOriBufferWithoutEP.bif"> </StreamSource>
         </ES_Descriptor>
      </esDescr>
   </Descr>
</InitialObjectDescriptor>
</Header>
<Body>
   <Replace>
      <Scene>
         <Group>
            <children>
               …
               <Transform translation="0 -2.373 -7.635" DEF="ROT">
                  <children>
                     <Shape>
                        <appearance> … </appearance>
                        <geometry DEF="MY_BOX">
                           <Box size="30.0 10.0 30.0"></Box>
                        </geometry>
                     </Shape>
                     <BitWrapper type="0" buffer="sample.aac">
                        <node>
                           <OrientationInterpolator DEF="Camera01-ROT-INTERP">
</OrientationInterpolator>
                        </node>
                     </BitWrapper>
                  </children>
               </Transform>
               <TimeSensor cycleInterval="6.0" loop="TRUE" DEF="TimeSource"> </TimeSensor>
            </children>
         </Group>
         <ROUTE fromNode="TimeSource" fromField="fraction_changed" toNode="Camera01-ROT-
INTERP"
                toField="set_fraction"> </ROUTE>
         <ROUTE fromNode="Camera01-ROT-INTERP" fromField="value_changed" toNode="ROT"
                toField="rotation"> </ROUTE>
      </Scene>
   </Replace>
</Body>
```

The above xmt file is taken by the XMT parser and the following two files(.scene file and .mux file) are produced by the XMT parser with the help of XMT-A schema description and two XMT style sheets(XMT2BIFS, XMT2MUX).

In “ICOriBufferWithoutEP.scene”

```
…
BitWrapper {
   node DEF  Camera01-ROT-INTERP OrientationInterpolator { }
   type 0
   buffer "sample.aac"
}
…
```

In “ICOriBufferWithoutEP.mux”

```
InitialObjectDescriptor {
   …
   esDescr [
      ES_Descriptor {
         ES_ID   301
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      decConfigDescr DecoderConfigDescriptor{
         objectTypeIndication  2
         streamType  3
         bufferSizeDB  500000
         decSpecificInfo BIFSv2Config {
            PROTOIDbits  10
            nodeIDbits  10
            routeIDbits  10
         }
      }
      slConfigDescr SLConfigDescriptor{
         useAccessUnitStartFlag  TRUE
         useAccessUnitEndFlag  TRUE
         timeStampResolution  100
         timeStampLength  14
      }
      muxInfo MuxInfo {
         fileName       "ICOriBufferWithoutEP.bif"
         streamFormat BIFS
      }
   }
]
}
```

9.7.1.2 Out-band scenario: Using pre-encoded bitstream

One can create a text file and a script file (Both are IM1 textual format) that carry encoded out-band bitstream. The encoded bitstream is already made and provided to an author. Therefore, one does not need to have original data in the node field and its encoding parameters in BitWrapper node. In this case, the name of the file given by the parameter value of <sourceFormat> element in <BitWrapperEncodingHints> element indicates the pre-encoded bitstream.

In “ICOriUrlWithoutEP.xmt”,

```
<Header>
   <InitialObjectDescriptor objectDescriptorID="1" binaryID="1" >
      …
      <Descr>
         <esDescr>
            <ES_Descriptor ES_ID="xyz" binaryID="201">
               <decConfigDescr >
                  <DecoderConfigDescriptor     streamType="1"  objectTypeIndication="2"
                          bufferSizeDB="10000000">
                  </DecoderConfigDescriptor>
               </decConfigDescr>
               <slConfigDescr>
                  <SLConfigDescriptor>
                     <custom useAccessUnitStartFlag="TRUE" useAccessUnitEndFlag="TRUE"
                             timeStampResolution="100" timeStampLength="14"  />
                  </SLConfigDescriptor>
               </slConfigDescr>
               <StreamSource url="ICOriUrlWithoutEP.od"> </StreamSource>
            </ES_Descriptor>
         </esDescr>
         <esDescr>
            <ES_Descriptor ES_ID="xyz1" binaryID="301">
               <decConfigDescr >
                  <DecoderConfigDescriptor     streamType="3" objectTypeIndication="2"
bufferSizeDB="500000">
                     <decSpecificInfo>
                        <BIFSv2Config nodeIDbits="10" routeIDbits="10" PROTOIDbits="10">
</BIFSv2Config>
                     </decSpecificInfo>
                  </DecoderConfigDescriptor>
               </decConfigDescr>
               <slConfigDescr>
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                <SLConfigDescriptor>
                   <custom useAccessUnitStartFlag="TRUE" useAccessUnitEndFlag="TRUE"
                           timeStampResolution="100" timeStampLength="14" />
                </SLConfigDescriptor>
             </slConfigDescr>
             <StreamSource url="ICOriUrlWithoutEP.bif"> </StreamSource>
          </ES_Descriptor>
       </esDescr>
    </Descr>
  </InitialObjectDescriptor>
</Header>
<Body>
  <Replace>
    <Scene>
       <Group>
          <children>
             …
             <Transform translation="0 -2.373 -7.635" DEF="ROT">
                <children>
                   <Shape>
                      <appearance> … </appearance>
                      <geometry DEF="MY_BOX">
                         <Box size="30.0 10.0 30.0"></Box>
                      </geometry>
                   </Shape>
                   <BitWrapper type="0" url="12">
                      <node>
                         <OrientationInterpolator DEF="Camera01-ROT-INTERP">
</OrientationInterpolator>
                      </node>
                   </BitWrapper>
                </children>
             </Transform>
             <TimeSensor cycleInterval="6.0" loop="TRUE" DEF="TimeSource"> </TimeSensor>
          </children>
       </Group>
       <ROUTE fromNode="TimeSource" fromField="fraction_changed" toNode="Camera01-ROT-
INTERP"
              toField="set_fraction"> </ROUTE>
       <ROUTE fromNode="Camera01-ROT-INTERP" fromField="value_changed" toNode="ROT"
              toField="rotation"> </ROUTE>
    </Scene>
  </Replace>
  <ObjectDescriptorUpdate>
    <OD>
       <ObjectDescriptor objectDescriptorID="12" binaryID="12">
          <Descr>
             <esDescr>
                <ES_Descriptor ES_ID="PI" binaryID="211">
                   <decConfigDescr >
                      <DecoderConfigDescriptor  streamType="3" objectTypeIndication="5"
                              bufferSizeDB="50000">
                         <decSpecificInfo>
                            <AFXConfig>
                               <OriInterpCompDecoderSpecific>
</OriInterpCompDecoderSpecific>
                            </AFXConfig>
                         </decSpecificInfo>
                      </DecoderConfigDescriptor>
                   </decConfigDescr>
                   <slConfigDescr>
                      <SLConfigDescriptor>
                         <custom  timeStampResolution="100" />
                      </SLConfigDescriptor>
                   </slConfigDescr>
                   <StreamSource>
                      <BitWrapperEncodingHints>
                         <BitWrapperICEncodingHints>
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                        <sourceFormat>
                           <param value="sample.aac"> </param>
                        </sourceFormat>
                     </BitWrapperICEncodingHints>
                  </BitWrapperEncodingHints>
               </StreamSource>
            </ES_Descriptor>
         </esDescr>
      </Descr>
   </ObjectDescriptor>
</OD>
</ObjectDescriptorUpdate>
</Body>
```

The above xmt file is taken by the XMT parser and the following two files(.scene file and .mux file) are produced by the XMT parser with the help of XMT-A schema description and two XMT style sheets(XMT2BIFS, XMT2MUX).

In “ICOriUrlWithoutEP.scene”,

```
…
BitWrapper {
   node DEF  Camera01-ROT-INTERP OrientationInterpolator { }
   type 0
   url 12
}
…
UPDATE OD [
   ObjectDescriptor {
      objectDescriptorID 12
      muxScript ICOriUrlWithoutEP.mux
   }
]
```

In “ICOriUrlWithoutEP.mux”,

```
InitialObjectDescriptor {
   objectDescriptorID  1
   …
   esDescr [
      ES_Descriptor {
         ES_ID   201
         decConfigDescr DecoderConfigDescriptor{…}
         slConfigDescr SLConfigDescriptor{…}
         muxInfo MuxInfo {
            fileName   "ICOriUrlWithoutEP.od"
            streamFormat BIFS
         }
      }
   ]
   esDescr [
      ES_Descriptor {
         ES_ID   301
         decConfigDescr DecoderConfigDescriptor{…}
         slConfigDescr SLConfigDescriptor{…}
         muxInfo MuxInfo {
            fileName   "ICOriUrlWithoutEP.bif"
            streamFormat BIFS
         }
      }
   ]
}


ObjectDescriptor {
   objectDescriptorID  12
   esDescr [
      ES_Descriptor {
         ES_ID   211
         decConfigDescr DecoderConfigDescriptor{
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            …
            decSpecificInfo AFXConfig {
               afxext  OriInterpCompDecoderSpecific { }
            }
         }
         slConfigDescr SLConfigDescriptor{…}
         muxInfo MuxInfo {
            fileName "sample.aac"
            streamFormat InterpolatorCompression
         }
      }
   ]
}
```

9.7.2 Example: Using BitWrapper, original data(uncompressed) and its encoding parameters

One can create the XMT input file that has BitWrapper node, original data (not pre-encoded one) and its encoding parameters in it. From this input file, one also can generate a mp4 file. The processes are explained in the following figure.

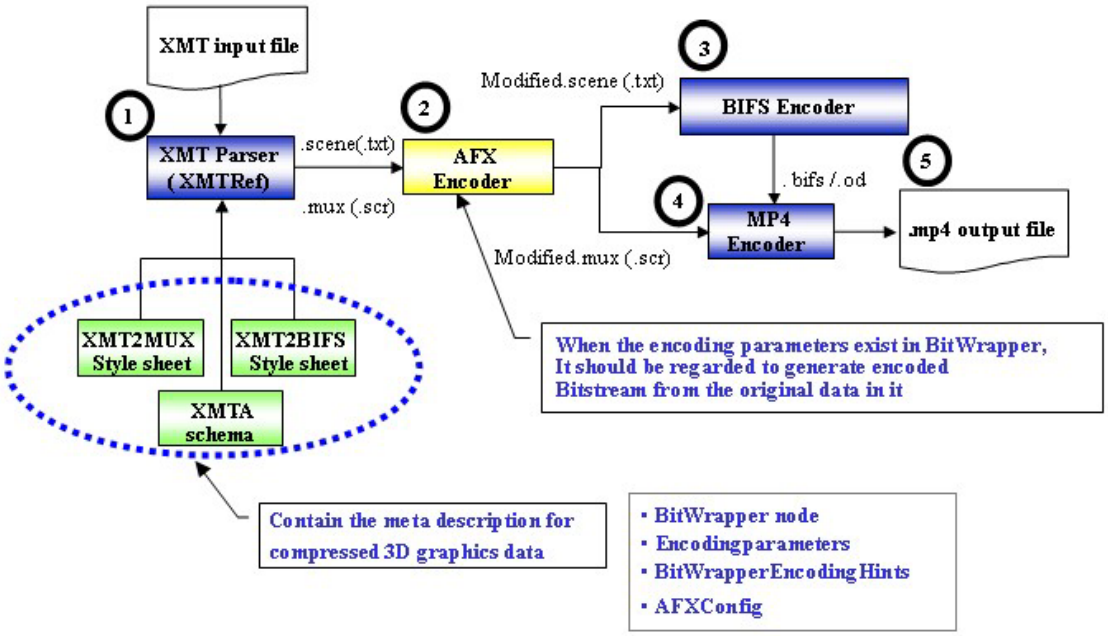


Figure 61 — From XMT to mp4: Using BitWrapper, original data and its encoding parameters

From the figure, a XMT parser takes a XMT file as an input and produces two output files with the help of XMT-A schema description file and two XMT style sheets (XMT2BIFS, XMT2MUX) in the XMTParser (circle 1). However, the two output files (.scene and .mux) cannot be used as input files of BIFS encoder and MP4 encoder directly, because the XMT parser just converts the description from XMT to IM1 textual format and does not encode the original data with the provided encoding parameters in the BitWrapper node. Therefore, we need to have some means to encode the original data of node field in the BitWrapper node with encoding parameters. It is called AFX encoder (circle 2) and is located between the XMT parser and the BIFS encoder, the MP4 encoder. The BitWrapper node currently supports to encode node data related to AFX tools (14496-16:2003), three interpolators (Coordinate, Orientation and Position) and 3DMC.

The AFX encoder reads the original data of node field, the type of node, its encoding parameters in the BitWrapper node and the name to be used as a name of encoded bitstream (The name is given by buffer field in in-band scenario and by parameter value of <sourceFormat> element in out-band scenario). Then, it produces an encoded bitstream and recreates modified output files (.scene and .mux). Modified .scene file contains the encoded bitstream instead of containing original data and its encoding parameters in the BitWrapper node in the AFX Encoder. The remaining processes are the same as the previous case in the subclause 9.7.1.

The following two examples show how the above processes are applied in real situation.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.7.2.1 In-band scenario: Using encoding parameter

One can create a text file and a script file (Both are IM1 textual format) that carry encoded in-band bitstream. The encoded bitstream can be made by encoding the original data which are contained in the node field of BitWrapper node and stored in the encoded bitstream file. The name to store the encoded bitstream is given by the buffer field in BitWrapper node. In the encoding, the encoding parameters that are specified in xmt file should be used by authors.

In “ICOriBufferWithEP.xmt”,

```
<Header>
   <InitialObjectDescriptor objectDescriptorID="1" binaryID="1" >
      …
      <Descr>
         <esDescr>
            <ES_Descriptor ES_ID="xyz1" binaryID="301">
               <decConfigDescr >
                  <DecoderConfigDescriptor streamType="3" objectTypeIndication="2"
bufferSizeDB="500000">
                     <decSpecificInfo>
                        <BIFSv2Config nodeIDbits="10" routeIDbits="10" PROTOIDbits="10">
</BIFSv2Config>
                     </decSpecificInfo>
                  </DecoderConfigDescriptor>
               </decConfigDescr>
               <slConfigDescr>
                  <SLConfigDescriptor>
                     <custom useAccessUnitStartFlag="TRUE" useAccessUnitEndFlag="TRUE"
                             timeStampResolution="100" timeStampLength="14" />
                  </SLConfigDescriptor>
               </slConfigDescr>
               <StreamSource url="ICOriBufferWithEP.bif"> </StreamSource>
            </ES_Descriptor>
         </esDescr>
      </Descr>
   </InitialObjectDescriptor>
</Header>
<Body>
   <Replace>
      <Scene>
         <Group>
            <children>
               …
               <Transform translation="0 -2.373 -7.635" DEF="ROT">
                  <children>
                     <Shape>
                        <appearance> … </appearance>
                        <geometry DEF="MY_BOX">
                           <Box size="30.0 10.0 30.0"></Box>
                        </geometry>
                     </Shape>
                     <BitWrapper type="0" buffer="sample.aac">
                        <node>
                           <OrientationInterpolator  DEF="Camera01-ROT-INTERP"
                              key="0.0 0.2 0.4 0.6 0.8 1.0"
                              keyValue="0.0 0.0 1.0 0.0, 0.0 0.0 1.0 3.14, 0.0 0.0 1.0 6.28,
1.0 0.0 1.0 0.0, 1.0 0.0
                                        1.0 3.14, 1.0 0.0 1.0 6.28">
                           </OrientationInterpolator>
                        </node>
                        <OrientationInterpolatorEncodingParameter keyQBits="10"
keyValueQBits="14"
                                 dpcmMode="0">
                        </OrientationInterpolatorEncodingParameter>
                     </BitWrapper>
                  </children>
               </Transform>
               <TimeSensor cycleInterval="6.0" loop="TRUE" DEF="TimeSource"> </TimeSensor>
            </children>
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        </Group>
        <ROUTE fromNode="TimeSource" fromField="fraction_changed" toNode="Camera01-ROT-
INTERP"
               toField="set_fraction"> </ROUTE>
        <ROUTE fromNode="Camera01-ROT-INTERP" fromField="value_changed" toNode="ROT"
               toField="rotation"> </ROUTE>
     </Scene>
   </Replace>
</Body>
```

The above xmt file is taken by the XMT parser and the following two files(.scene file and .mux file) are produced by the XMT parser with the help of XMT-A schema description and two XMT style sheets(XMT2BIFS, XMT2MUX).

In “ICOriBufferWithEP.scene”,

```
…
BitWrapper {
   node DEF  Camera01-ROT-INTERP OrientationInterpolator {
      key [ 0.0 0.2 0.4 0.6 0.8 1.0]
      keyValue [ 0.0 0.0 1.0 0.0, 0.0 0.0 1.0 3.14, 0.0 0.0 1.0 6.28, 1.0 0.0 1.0 0.0, 1.0
0.0 1.0 3.14, 1.0 0.0 1.0 6.28]
   }
   OrientationInterpolatorEncodingParameter {
      keyQBits   10
      keyValueQBits14
      dpcmMode   0
   }
   type 0
   buffer "sample.aac"
}
…
```

In “ICOriBufferWithEP.mux”,

```
InitialObjectDescriptor {
   objectDescriptorID  1
   …
   esDescr [
      ES_Descriptor {
         ES_ID   301
         decConfigDescr DecoderConfigDescriptor{…}
         slConfigDescr SLConfigDescriptor{…}
         muxInfo MuxInfo {
            fileName   "ICOriBufferWithEP.bif"
            streamFormat BIFS
         }
      }
   ]
}
```

As shown above, this “.scene” file could not be directly used as an input to BIFS encoder because it contains uncompressed original data in the node field. Therefore, the AFX encoder encodes the node data in the node field with the given encoding parameters and stores it in the given file name in the buffer field.

And the expected outputs of AFX encoder would be like this;

In “ICOriBufferWithEP_Modified.scene”

```
…
BitWrapper {
   node DEF  Camera01-ROT-INTERP OrientationInterpolator { }
   type 0
   buffer "sample.aac"
}
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In “ICOriBufferWithEP_Modified.mux”,

```
InitialObjectDescriptor {
   objectDescriptorID  1
   …
   esDescr [
      ES_Descriptor {
         ES_ID   301
         decConfigDescr DecoderConfigDescriptor{…}
         slConfigDescr SLConfigDescriptor{…}
         muxInfo MuxInfo {
            fileName   "ICOriBufferWithEP_modified.bif"
            streamFormat BIFS
         }
      }
   ]
}
```

9.7.2.2 Out-band scenario: Using encoding parameter

One can create a text file and a script file (Both are IM1 textual format) that carry encoded out-band bitstream. The encoded bitstream can be made by encoding the original data which are included in the node field of BitWrapper node and stored in the encoded bitstream file. The name to store the encoded bitstream is given by the parameter value of <sourceFormat> element in <BitWrapperEncodingHints> element. In the encoding, the encoding parameters that are specified in xmt file by an author should be used.

In “ICOriUrlWithEP.xmt”,

```
<Header>
   <InitialObjectDescriptor objectDescriptorID="1" binaryID="1" >
      …
      <Descr>
         <esDescr>
            <ES_Descriptor ES_ID="xyz" binaryID="201">
               <decConfigDescr >
                  <DecoderConfigDescriptor streamType="1" objectTypeIndication="2"
bufferSizeDB="10000000">
                  </DecoderConfigDescriptor>
               </decConfigDescr>
               <slConfigDescr>
                  <SLConfigDescriptor>
                     <custom useAccessUnitStartFlag="TRUE" useAccessUnitEndFlag="TRUE"
                             timeStampResolution="100" timeStampLength="14"  />
                  </SLConfigDescriptor>
               </slConfigDescr>
               <StreamSource url="ICOriUrlWithEP.od"> </StreamSource>
            </ES_Descriptor>
         </esDescr>
         <esDescr>
            <ES_Descriptor ES_ID="xyz1" binaryID="301">
               <decConfigDescr >
                  <DecoderConfigDescriptor streamType="3" objectTypeIndication="2"
bufferSizeDB="500000">
                     <decSpecificInfo>
                        <BIFSv2Config nodeIDbits="10" routeIDbits="10" PROTOIDbits="10">
                        </BIFSv2Config>
                     </decSpecificInfo>
                  </DecoderConfigDescriptor>
               </decConfigDescr>
               <slConfigDescr>
                  <SLConfigDescriptor>
                     <custom useAccessUnitStartFlag="TRUE" useAccessUnitEndFlag="TRUE"
                             timeStampResolution="100" timeStampLength="14" />
                  </SLConfigDescriptor>
               </slConfigDescr>
               <StreamSource url="ICOriUrlWithEP.bif"> </StreamSource>
            </ES_Descriptor>
         </esDescr>
      </Descr>
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   </InitialObjectDescriptor>
</Header>
<Body>
   <Replace>
      <Scene>
         <Group>
            <children>
               …
               <Transform translation="0 -2.373 -7.635" DEF="ROT">
                  <children>
                     <Shape>
                        <appearance> … </appearance>
                        <geometry DEF="MY_BOX">
                           <Box size="30.0 10.0 30.0"></Box>
                        </geometry>
                     </Shape>
                     <BitWrapper type="0"  url="12">
                        <node>
                           <OrientationInterpolator DEF="Camera01-ROT-INTERP"
                              key="0.0 0.2 0.4 0.6 0.8 1.0"
                              keyValue="0.0 0.0 1.0 0.0, 0.0 0.0 1.0 3.14, 0.0 0.0 1.0 6.28,
1.0 0.0 1.0 0.0, 1.0 0.0
                                       1.0 3.14, 1.0 0.0 1.0 6.28">
                           </OrientationInterpolator>
                        </node>
                        <OrientationInterpolatorEncodingParameter keyQBits="10"
   keyValueQBits="14"
                                 dpcmMode="0">
                        </OrientationInterpolatorEncodingParameter>
                     </BitWrapper>
                  </children>
               </Transform>
               <TimeSensor cycleInterval="6.0" loop="TRUE" DEF="TimeSource"> </TimeSensor>
            </children>
         </Group>
         <ROUTE fromNode="TimeSource" fromField="fraction_changed" toNode="Camera01-ROT-
INTERP"
                toField="set_fraction"> </ROUTE>
         <ROUTE fromNode="Camera01-ROT-INTERP" fromField="value_changed" toNode="ROT"
                toField="rotation"> </ROUTE>
      </Scene>
   </Replace>
   <ObjectDescriptorUpdate>
      <OD>
         <ObjectDescriptor objectDescriptorID="12" binaryID="12">
            <Descr>
               <esDescr>
                  <ES_Descriptor ES_ID="PI" binaryID="211">
                     <decConfigDescr >
                        <DecoderConfigDescriptor streamType="3" objectTypeIndication="5"
                                 bufferSizeDB="50000">
                           <decSpecificInfo>
                              <AFXConfig>
                                 <OriInterpCompDecoderSpecific>
</OriInterpCompDecoderSpecific>
                              </AFXConfig>
                           </decSpecificInfo>
                        </DecoderConfigDescriptor>
                     </decConfigDescr>
                     <slConfigDescr>
                        <SLConfigDescriptor>
                           <custom  timeStampResolution="100" />
                        </SLConfigDescriptor>
                     </slConfigDescr>
                     <StreamSource>
                        <BitWrapperEncodingHints>
                           <BitWrapperICEncodingHints>
                              <sourceFormat>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                        <param value="sample.aac"> </param>
                     </sourceFormat>
                  </BitWrapperICEncodingHints>
               </BitWrapperEncodingHints>
            </StreamSource>
         </ES_Descriptor>
      </esDescr>
   </Descr>
</ObjectDescriptor>
</OD>
</ObjectDescriptorUpdate>
</Body>
```

The above xmt file is taken by the XMT parser and the following two files(.scene file and .mux file) are produced by the XMT parser with the help of XMT-A schema description and two XMT style sheets(XMT2BIFS, XMT2MUX)

In “ICOriUrlWithEP.scene”

```
…
BitWrapper {
   node DEF  Camera01-ROT-INTERP OrientationInterpolator {
   key [ 0.0 0.2 0.4 0.6 0.8 1.0]
   keyValue [ 0.0 0.0 1.0 0.0, 0.0 0.0 1.0 3.14, 0.0 0.0 1.0 6.28, 1.0 0.0 1.0 0.0, 1.0
0.0 1.0 3.14, 1.0 0.0 1.0 6.28]
}
OrientationInterpolatorEncodingParameter {
   keyQBits   10
   keyValueQBits14
   dpcmMode   0
}
type 0
   url 12
}
…
UPDATE OD [
   ObjectDescriptor {
      objectDescriptorID 12
      muxScript ICOriUrlWithEP.mux
   }
]
```

In “ICOriUrlWithEP.mux”,

```
…
InitialObjectDescriptor {
   objectDescriptorID  1
   …
   esDescr [
      ES_Descriptor {
         ES_ID   201
         decConfigDescr DecoderConfigDescriptor{…}
         slConfigDescr SLConfigDescriptor{…}
         muxInfo MuxInfo {
            fileName   "ICOriUrlWithEP.od"
            streamFormat BIFS
         }
      }
   ]
   esDescr [
      ES_Descriptor {
         ES_ID   301
         decConfigDescr DecoderConfigDescriptor{…}
         slConfigDescr SLConfigDescriptor{…}
         muxInfo MuxInfo {
            fileName   "ICOriUrlWithEP.bif"
            streamFormat BIFS
         }
      }
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   ]
}
ObjectDescriptor {
   objectDescriptorID  12
   esDescr [
      ES_Descriptor {
         ES_ID   211
         decConfigDescr DecoderConfigDescriptor{
            …
            decSpecificInfo AFXConfig {
               afxext  OriInterpCompDecoderSpecific { }
            }
         }
         slConfigDescr SLConfigDescriptor{…}
         muxInfo MuxInfo {
            fileName   "sample.aac"
            streamFormat InterpolatorCompression
         }
      }
   ]
}
```

As shown above, this ".scene" file could not be directly used as an input to BIFS encoder because it contains uncompressed original data in the node field. Therefore, the AFX encoder encodes the node data in the node field with the given encoding parameters and stores it in the given file name in the <sourceFormat> element in <BitWrapperEncodingHints> element.

And the expected outputs of AFX encoder would be like this;

In "ICOriUrlWithEP_Modified.scene",

"

```
…
BitWrapper {
        node DEF Camera01-ROT-INTERP OrientationInterpolator { }
        type 0
        url 12
}
…
UPDATE OD [
        ObjectDescriptor {
                objectDescriptorID 12
                muxScript ICOriUrlWithEP_Modified.mux
        }
]
```

"

In "ICOriUrlWithEP_Modified.mux",

```
…
InitialObjectDescriptor {
   objectDescriptorID  1
   …
   esDescr [
      ES_Descriptor {
         ES_ID   201
         decConfigDescr DecoderConfigDescriptor{…}
         slConfigDescr SLConfigDescriptor{…}
         muxInfo MuxInfo {
            fileName   "ICOriUrlWithEP_modified.od"
            streamFormat BIFS
         }
      }
   ]
   esDescr [
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      ES_Descriptor {
         ES_ID   301
         decConfigDescr DecoderConfigDescriptor{…}
         slConfigDescr SLConfigDescriptor{…}
         muxInfo MuxInfo {
            fileName   "ICOriUrlWithEP_modified.bif"
            streamFormat BIFS
         }
      }
   ]
}
ObjectDescriptor {
   objectDescriptorID  12
   esDescr [
       ES_Descriptor {
         ES_ID   211
         decConfigDescr DecoderConfigDescriptor{
            …
            decSpecificInfo AFXConfig {
               afxext  OriInterpCompDecoderSpecific { }
            }
         }
         slConfigDescr SLConfigDescriptor{
            …
         }
         muxInfo MuxInfo {
            fileName   "sample.aac"
            streamFormat InterpolatorCompression
         }
      }
   ]
}
```

10 MPEG-J

10.1 Architecture

10.1.1 Parametric versus Programmatic System

10.1.1.1 Overview of a Parametric MPEG-4 System

Figure 62 shows an example of the basic MPEG-4 player, a parametric system. MPEG-4 coded data from storage/network goes through a DMIF and a demultiplex layer. In the demultiplex layer, M4Mux packets pass through the Sync Layer resulting in unformatted SL-PDUs of each media type (coded audio, video, speech or facial animation streams) which are then buffered in the respective decoder buffers and are offered to the corresponding media decoders. Also, the SL-PDUs corresponding to scene description representation are input to the BIFS decoder, the output of which goes to the Scene Graph. The output of the media decoders as well as Scene Graph feeds the Compositor and the Renderer, which may respond to (very basic) user interaction such as mouse clicks etc. The output of Compositor and Renderer is the scene for presentation.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

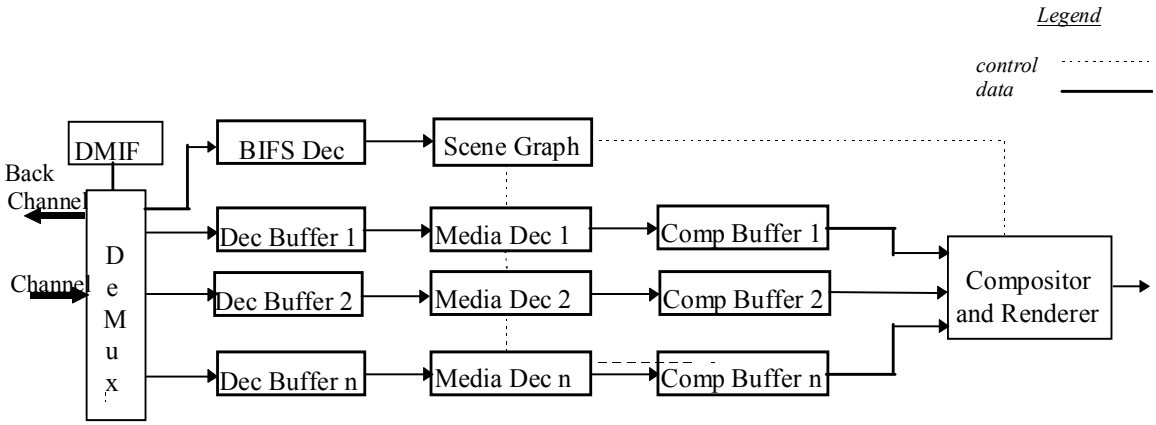


Figure 62 — An MPEG-4 Player

10.1.2 The MPEG-J System

The MPEG-J is a programmatic system, which specifies interfaces for interoperation of an MPEG-4 media player with Java code. By combining MPEG-4 media and safe executable code, content creators may embed complex control mechanisms with their media data to intelligently manage the operation of the audio-visual session. Java byte code is delivered to the MPEG-4 terminal as a separate elementary stream. There, it will be directed to the MPEG-J run time environment, which includes a Java Virtual Machine, from where the MPEG-J program will have access to the various components of the MPEG-4 player. Figure 63 shows an example of the components of the MPEG-J operating environment.

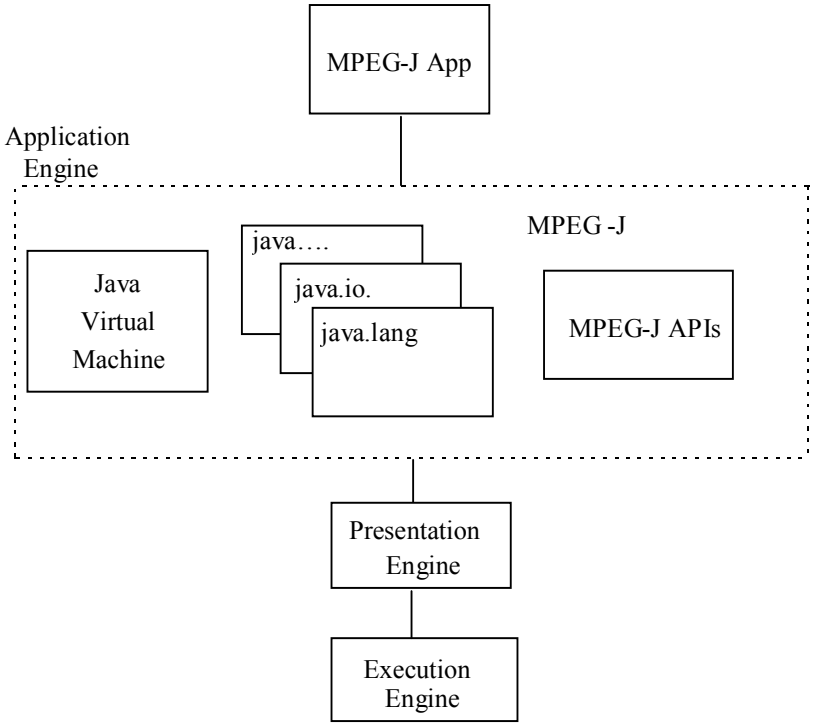


Figure 63 — MPEG-J Software architecture

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The software architecture of MPEG-J takes into consideration the resources available on the underlying platform. The architecture involves the isolation of distinct *components*, the design of interface that reflects them, and the characterization of interactions between components. Such components include:

Execution and Presentation Resources: It is assumed that the decoding and presenting resources are limited. This component abstracts access to information on such static and dynamic resources in the player. It abstracts notification during changes in such resources. Further, it provides for some minimal control of the same.

Decoders: This component abstracts the media decoders used to decode the received media streams. The programmatic control and their manipulation to add extra functionality is also abstracted by this component.

Network Resources: Since the device receives media streams, this component abstracts the control of such streams. It also abstracts the media pipeline, which transports and presents the stream.

Scene Graph: An MPEG-4 session has a Scene Graph which spatially and temporally represents audio visual objects This component abstracts access and control of the scene graph.

The MPEG-J APIs specified in this document are the interfaces that reflect the above said components. A block diagram of the MPEG-J player controlling an MPEG-4 player environment is shown in Figure 64. The lower half of this drawing depicts the parametric MPEG-4 player of Figure 62 and is also referred to as the Presentation Engine. The upper half of Figure 64 illustrates this the MPEG-J system controlling the Presentation Engine is also referred to as the Application Engine.. The APIs shown in Figure 64 will be specified later in this document.

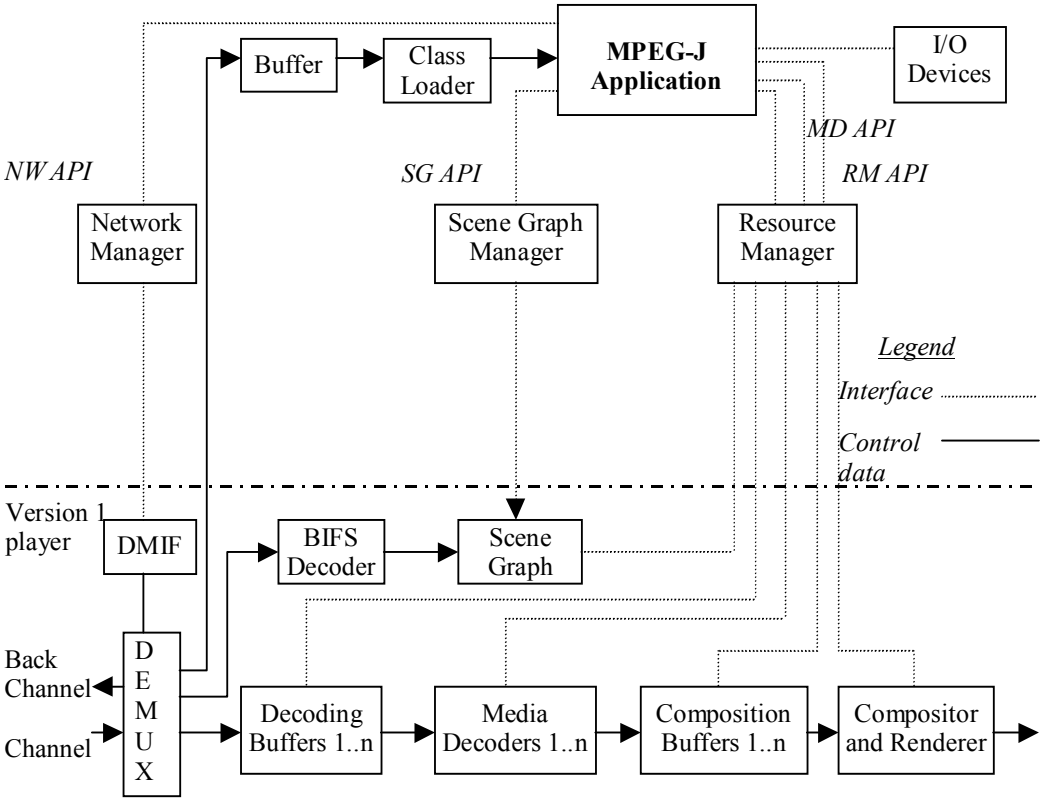


Figure 64 — Architecture of an MPEG-J enabled MPEG-4 System

10.2 MPEG-J Session

The MPEG-J session need not be started till it is clear that MPEG-J application programs will be received and are to be executed. This can be recognized by the presence of one of more of MPEG-J Elementary streams.

10.2.1 Walkthrough of an MPEG-J Session Start-up

The MPEG-J session is initiated when the MPEG-4 player receives an MPEG-J Object Descriptor. The player takes the following steps:

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

It opens the MPEG-J elementary stream via a DMIF channel. The MPEG-J Elementary stream is a SL packetized stream, similar to all other MPEG-4 streams.

It delivers the Access Units to the class loader, which loads the classes.

The MPEG-J "decoder" receives the arriving Access Units (it "decodes" them). There can be more than one class with an entry point within one MPEG-J stream. Each time such a class containing an entry-point is received (a "run" method) execution will start there as a new thread.

10.2.2 Local and Remote MPEG-J Applications

MPEG-J application that uses the MPEG-J APIs to control the underlying MPEG-4 player can either be local or remote. In the case of a remote application that is received in the MPEG-J Elementary Stream, it must implement the MPEGlet interface. The lifecycle and the security model of such an application (MPEGlet) are described in the next two sections. However, this does not apply to local applications.

10.2.3 MPEG-J Elementary Stream, Object Descriptor, and the Name Scope

10.2.3.1 MPEG-J Elementary Stream

The MPEG-J data comprising of class files or object data is streamed to the MPEG-J terminal as an MPEG-J Elementary Stream. The class files and all the associated data in such a stream can be optionally packaged together. Further, the class files in the stream (irrespective of whether it is packaged or not, can be compressed. The stream type of such an elementary stream is uniquely defined in Table 5 streamType Values, ISO/IEC 14496-1.

10.2.3.2 MPEG-J Object Descriptor

The MPEG-J elementary stream and the application programs (MPEGlets) derive their scope and properties from its Object Descriptor, which in turn is scoped by the initOD or the updateOD of the scene.

10.2.3.3 The Name Scope of an MPEG-J Stream

The Name Scope of the MPEGlets in an MPEG-J Stream is derived from the Object Descriptor of that MPEG-J Elementary Stream. Similar to the node identifiers in the scene graph, all the identifiers used by an MPEGlet in an Elementary Stream are interpreted within the name scope of that Elementary Stream and its Object Descriptor. Therefore, all the rules that restrict the name scope of an inline scene apply to the MPEG-J session also.

The name scope of an MPEGlet is determined by the managers it receives from the MpegjTerminal. The MPEGlet must pass a reference to itself in the constructor of the MpegjTerminal to identify the name scope used by the managers. A local application may use the zero-argument constructor of the MpegjTerminal to imply that the managers should use the root name scope.

10.2.4 Life Cycle of an MPEGlet

The life cycle of an MPEGlet is very similar to that of an applet. The MPEGlet interface has init(), run(), stop(), and destroy() methods. When an MPEGlet is received in the bitstream, it is loaded after the Start-Loading Time Stamp and before the Load-By Time Stamp as described in subclause 10.3.2. At the time instant specified by the Load-By Time Stamp, the init() method of the MPEGlet is executed. This is where all the initializations for the MPEGlet are typically done. After initializing the MPEGlet, the run method is called as a separate thread. Similar to a Java applet, the stop() and destroy() methods are specified in the MPEGlet interface. If the MPEG-J player receives another MPEGlet in the bitstream, it is initialized and started as a different thread.

10.2.5 Security Model of an MPEGlet

The security model of an MPEGlet is very similar to that of an applet. However, the security manager implemented on the player can add or relax the security restrictions. By default all the security restrictions that apply to applets apply to the MPEGlets too. These default security restrictions of an MPEGlet are:

MPEGlets cannot load libraries or define native methods.

MPEGlets can use only their own Java code, MPEG-J APIs, and the Java APIs the underlying platform provides.

An MPEGlet cannot normally read or write files on the host that is executing it.

An MPEGlet cannot start any program on the host that is executing it.

An MPEGlet cannot read certain system properties except through the Terminal Capability APIs

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.3 Delivery of MPEG-J Data

The MPEG-J application programs are delivered to the MPEG-4 player as MPEG-4 elementary streams defined in this document. The MPEG-J data could be classes, serialized objects, or any associated data (in the case of packaged form). Serialized objects and other auxiliary data are expected to accompany classes that have knowledge about handling those objects.

10.3.1 Issues in Delivery of Byte Code

The MPEG-J data (classes or objects) must be delivered in a timely fashion to the player. A header is used along with the class files or objects (serialized) to ensure this. This header, which is called the Java Stream Header, is attached to each class file or object data before it is passed on to the Sync Layer. After packetization, any “time aware” transport mechanism, like M4Mux, RTP, and even MPEG2 transport stream, can be used to transport the data to the client side.

10.3.1.1 Packet Loss

Packet loss in the case of Java byte code streaming will be a problem for the execution of the programs. The possible options for dealing with data loss are:

Retransmission of the entire class at regular intervals in the absence of a back channel. This would also help to facilitate random access points in the case of media. However, this may not be possible when there are a large number of clients or when the class (or object) is huge, making retransmission prohibitive.

When a back channel is present this loss can be signaled to the server and the lost packet can be retransmitted. There are a number of error resilient schemes, with built-in redundancy, available to recover from a partial loss of data. For e.g., schemes like forward error correction can be used. However, currently none of these schemes are mandated in an MPEG-J stream.

Packet Loss is not handled at the MPEG-J layer. It is assumed that the underlying transport layer is reliable enough to ensure that there is no packet loss.

10.3.1.2 Security

To ensure the safety of the client, the byte code needs to be authentic. There are a number of security schemes that can be used to ensure the authenticity of the byte code. Any of these schemes can be accommodated in an IPMP Descriptor or an IPMP stream.

10.3.1.3 Compression

The Java byte code can be optionally compressed for bandwidth efficiency using the Zip compression mechanism. Files can be both compressed and uncompressed using the java.util.zip package. The underlying compression technique in Zip is ZLIB.[1)]

10.3.1.4 Class Dependency

If a given class depends on other classes, the classes that it depends on have to be loaded before the dependent class can be loaded. Similarly before an object can be instantiated, the class of which it is an instance must be loaded first. One way of doing this would be using a packaging scheme e.g. JAR to package all the interdependent class files together. However, this may not always be the optimal solution, especially in lossy transport channels as a single packet loss could result in a loss of the entire package. As an alternative a simple class-dependency mechanism is provided in the Java Stream header below. In this mechanism all the dependent classes of a particular class are listed in the header of that class file. It is required that those classes need to be loaded before this class can be loaded.

Two time stamps will be used in the next subsection, one signaling a time after which a particular class can be loaded (also called Start-Loading Time Stamp), and the second signaling the time by which the class is required to be loaded (Load-By Time Stamp). The Start-Loading Time Stamp of a class that depends on a number of other classes has to be later than the Load-By Time Stamps of all the classes it depends on. These two timestamps together aid in ensuring that the dependencies between classes are met.

---

1) More information about ZLIB can be found at the ZLIB Home Page http://www.gzip.org/zlib/. ZLIB is the underlying compression mechanism used by both gzip and zip.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.3.2 Semantics of Time Stamps in MPEG-J

The Decoding Time Stamp (DTS) and Composition Time Stamp (CTS) defined in the SL header in the Sync Layer will be used for the timely delivery of the MPEG-J Elementary Stream. The semantics of these timestamps for the MPEG-J Elementary Stream is defined in this section.

Start-Loading Time Stamp: This is used to signal the time instant at which the process of loading a class can be started. This time stamp is essential to avoid name space and resource conflicts. This timestamp also ensures that the resources for loading the class would be available at the terminal. In addition, this time stamp allows the terminal to receive classes ahead of the time at which they need to be loaded. This is carried in the SL Header as the Decoding Time stamp (DTS).

Each class is loaded by calling the loadClass(className) method of the class loader, where className is the name of the class. The name of class does not include the .class suffix.

Load-By Time Stamp: This time stamp is used to signal the time instant by which a class should be loaded at the MPEG-J terminal. If the received class implements the MPEGlet interface, it will also be initialized at this instant of time (by executing the init() method). After initialization, the MPEGlet would be run (by executing the run() method) as a separate thread at this time instant. This time stamp is carried in the SL Header as the Composition Time Stamp (CTS).

The above two time stamps define a window in time between which a given class shall be loaded. As described in the previous section this window helps in resolving the class dependencies between classes. When the window between these two timestamps are made large enough the problems due to non-uniform loading times on different client terminals can be avoided. Again, if the channel is lossy, this window can be made large enough to allow re-transmissions, if possible. With this mechanism the order in which the classes need to be loaded can be different from the order in which the classes arrive on the terminal.

10.3.3 Streaming Header

10.3.3.1 Description

Each class or a packaged set constitutes a separate Access Unit.

The payload can be classes (compressed or uncompressed) or instances as serialized objects.

Classes are identified by an ID (unique to the session). This ID can be used to identify classes when it is received multiple times. The ID of a class is also used to identify all its instances in the case of serialized objects. Java class names are used as IDs. Since these are variable length strings, the length of the string is also included in the header. The combination of the Class ID and its length (16 bits) are padded till the next 32 bit boundary. When multiple classes are packaged together, the name of the packaged file is used as the ID. There is a list of required classes, whose Load-By time need to precede the Start-Loading time of a class that requires it. A 13-bit number is used to specify the number of classes that are required before loading/instantiating the class/object data. Each required class is specified by its Class IDs and its length. In the packaged case, the list of required classes specify the classes in the archive that have to be loaded. Those files in the archive that are not listed as required classes need not be loaded by the terminal by the DTS or Load-By time.

10.3.3.2 JavaStreamHeader

10.3.3.2.1 Syntax

```
aligned(32) class JavaStreamHeader {
   bit(2) version;
   bit(1) isClassFlag;
   bit(13) numReqClasses;
   bit(1) isPackaged ;
   bit(3) compressionScheme;
   bit(12) reserved;
   JavaClassID classID;

   JavaClassID reqClassID[numReqClasses];

}
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.3.3.2.2 Semantics

version - Version number. This is currently 00.

isClassFlag – If set to 1, the payload represents a class. If set to zero, the payload does not represent a class, but instead represents content accessible to the MPEGlet as a ClassLoader resource. This content can be a java object or any other data that is useful to the MPEGlet. The MPEGlet may obtain a URL to access the content by calling the `getResource()` method of the ClassLoader with the JavaClassID as the parameter. In addition, if **isClassFlag** is set to 1 but **isPackaged** indicates a package, the Zip archive may contain content that does not represent class data. Such data shall be accessible by calling `getResource()` of the ClassLoader with the element name as the parameter.

numReqClasses - Number of classes that are required before loading this class (or before instantiation, in case of objects).

isPackaged – If set to 0, this indicates a single class file and not a package. 1 indicates that multiple class files are packaged together using Zip.

compressionScheme - To specify the type of compression scheme used (000 for objects, when no compression is used, 001 when the contents are compressed using Zip, 010-111 reserved for future use by ISO).

reserved – Bits reserved by ISO for future use. These bits should be 0xFFF.

classID – Information to identify this class or package. The definition of its type JavaClassID is defined in the next subsection.

reqClassID`[n]` – Information to identify the n required classes.

10.3.3.3 JavaClassID

10.3.3.3.1 Syntax

```
aligned(32) class JavaClassID {
   bit(16) length;
   bit(8 * length) ID;
}
```

10.3.3.3.2 Semantics

length - Number of bytes for the ID.

ID - Variable length string that identifies the class. The string is padded, so that the length of the combination of ID length field and the ID is multiple of 32 bits.

10.4 MPEG-J API List

10.4.1 Packages for MPEG-J from MPEG

Packages are a means to organize the implementation of APIs. The MPEG-J APIs are organized as the following packages:

org.iso.mpeg.mpegj.mpegj

org.iso.mpeg.mpegj.scene

org.iso.mpeg.mpegj.resource

org.iso.mpeg.mpegj.network

org.iso.mpeg.mpegj.decoder

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 97 — Categories of APIs

No	API Category and main classes/interfaces	Explanation
1.	**Scene** Scene Graph	Means by which MPEG-J applications access and manipulate the scene graph
2.	**Resource** ResourceManager CapabilityManager	Centralized facility for managing system resources Access to the static and dynamic capabilities of the terminal.
3.	**Media Decoders** MPDecoder	Access and control to the decoders used to decode the audio-visual objects.
4.	**Network** NetworkManager	Access and control of the network components of the MPEG-4 player.
5.	**Section Filtering and Service Information**	MPEG-2 Stream specific APIs defined in Part 9 of DAVIC 1.4.1 specification. This covers Section Filtering, Service Information, Resource Notification, and MPEG Component APIs.

10.4.2 MPEG-J API (org.iso.mpeg.mpegj)

The MPEG-J Terminal class provides the information about the managers that are implemented in the terminal. Each MPEGlet or application instantiates a new MpegjTerminal once it is loaded. This has methods to gain access to all the managers, viz., SceneManager, ResourceManager, and the NetworkManager.

Although an MPEG-J Terminal is instantiated by each MPEGlet, it should not be interpreted as creating a new terminal for each MPEGlet. A MPEG-J Terminal implementation gives the appropriate managers to the MPEGlet. The terminal, along with the managers, controls the environment (for e.g. the name scope) of the MPEGlet.

The ObjectDescriptor, the ESDescriptor, and the DecoderConfigDescriptor interfaces are also part of the org.iso.mpeg.mpeg.mpegj package. These interfaces provide access and abstraction to the above descriptors. Information about nodes, elementary streams, their types, and the decoder information can be obtained used these APIs.

10.4.3 Scene API

The SceneGraph API provides a mechanism by which MPEG-J applications access and manipulate the scene used for composition by the BIFS player. It is a low-level interface, allowing the MPEG-J application to monitor events in the scene, and modify the scene tree in a programmatic way. Nodes may also be created and manipulated, but only the fields of nodes that have been instanced with DEF are accessible to the MPEG-J application

This API has been designed to serve as the lowest layer of the MPEG-J scene graph manager. A terminal designer would only need to implement this package to have MPEG-J bindings to the native scene. Other class libraries could be specified entirely in Java to allow higher-level access to and control of the scene. Those libraries could be supplied as packages that run above this org.iso.mpeg.mpegj.scene package, allowing their selection to be determined based on a profile or level or could sent to the terminal in the bit stream.

10.4.3.1 Events

Events in the BIFS scene graph are identified by the two interface classes EventIn and EventOut. The EventOutListener class can monitor them.

10.4.3.1.1 EventIn

The EventIn interface class contains an interface class definition for each node type defined in MPEG-4 systems. These definitions enumerate all of the exposedField and eventIn field types in the node, in the order they are defined in this document.

Copyright International Organization for Standardization © ISO/IEC 2005 All rights reserved
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.4.3.1.2 EventOut

Likewise, the EventOut interface class contains an interface class definition for each node type defined in MPEG-4 systems. These definitions enumerate all of the exposedField and eventOut field types in the node, in the order they are defined in this document.

10.4.3.1.3 EventOutListener

The Scene Graph APIs also provide an EventOutListener interface, which can be used by the scene graph manager to identify a field value change when an eventOut is triggered.

10.4.3.2 Field Values

The scene graph APIs provide an interface for tagging objects that can return a field value. Similar to VRML, two general field types are supported. SFField is used for single value fields and MFField is used for multiple value fields. The supported SFField types are extended directly from the FieldValue interface, while the Multiple field types are extended through the MFFieldValue interface.

10.4.3.3 Scene Management

The following interfaces are used to facilitate programmatic control over the MPEG-4 terminal's native scene.

10.4.3.3.1 SceneManager

The SceneManager interface is the interface that allows access to the native scene. It contains methods for adding and removing a SceneListener. In order to access the BIFS scene graph, the SceneManager requires an instance of the scene, which it obtains through notification on a SceneListener instance. This method is the only normative way for an MPEG-J application to obtain a scene instance.

10.4.3.3.2 SceneListener

The SceneListener contains a notify method which can be called by the SceneManager when the BIFS scene has changed. The `notify()` method contains arguments to indicate the nature of the change, and an updated Scene instance. Currently three states can be passed through the scene listener. They indicate that the scene is ready, it has been replaced, or it has been removed.

10.4.3.3.2.1 Scene

The Scene interface acts as a proxy for the BIFS scene. It contains a `getNode()` method, which returns a Node proxy for the desired node in the scene. If the requested node does not exist it throws a BadParameterException, and if the scene is no longer valid it throws an InvalidSceneException.

10.4.3.3.2.2 Node

The Node interface acts as a proxy for a BIFS node in the scene graph. As previously mentioned, only nodes that have been instanced by a DEF identifier are available to the MPEG-J application. Three methods are available in the Node proxy for monitoring output events. The `getEventOut()` method reads the current value of an eventOut or exposedField of this node. There are also methods for adding and removing an EventOutListener. All three of these methods throw a BadParameterException if they fail. The fourth method contained in the Node interface is the sendEventIn method. This is the only method available to the application for modifying the BIFS scene. It updates the value of the eventIn or exposedField of the node. It is a synchronous call that will not return until the field is updated in the scene. The fifth method contained in the Node interface is the `getNodeType()` method. This method returns an integer identifying the type of the node (such as Transform). The node type constants are defined in the NodeType interface. All of the methods contained in the Node interface throw an InvalidNodeException if the node is no longer valid (if it has been replaced or deleted).

10.4.3.3.2.3 NodeValue

The NodeValue interface is used to represent the values of SFNode and MFNode fields. There are three types of NodeValue references:

The `getNode()` method of the Scene interface returns a Node that acts as a proxy for a node in the BIFS scene. This object also implements the NodeValue interface.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The `getEventOut()` method of the Node interface may return a SFNodeFieldValue. Its getSFNodeValue method returns a NodeValue that acts as a proxy for a child node in the BIFS scene. However, unlike the proxy returned by the Scene's getNode method, this proxy does not provide any way to access or modify the child node.

The NewNode interface extends NodeValue to support creation of new nodes. This interface has methods that describe the structure of the new node.

Sending an eventIn to a SFNode field (such as the geometry field of a Shape node) replaces a sub-graph of a BIFS scene. The value sent to the eventIn may be any one of the three types of NodeValue references enumerated above. If the value is a proxy for a node or child node, then the SFNode field value becomes a reference to the existing node (equivalent to making a USE reference to the node). If the object implements the NewNode interface, then a new node is created. The node type and DEF identifier of the new node are determined by calling the `getNodeType()` and `getNodeID()` methods (the node is not given a DEF identifier if the `getNodeID()` method returns zero). If the DEF identifier is already in use within the scene, then a BadParameterException shall be thrown. Each field and exposedField of the node shall be initialized to the value returned by calling `getField()` with the appropriate field defID (or the field's default value if null is returned). This algorithm is applied recursively in the case that the field or exposedField is a SFNode or MFNode. The recursion may form a directed acyclic graph if the same object is returned more than once.

10.4.3.4 Scene Controller Architecture

Scene controllers are called at every frame by the application. Therefore, the scene controller scenario is included inside the rendering loop scenario (see Figure 65, using Unified Modeling Language notation). An application developer can implement application-specific logic by extending the **SceneControllerListener** interface and registering this component with the **Renderer**. Using scene controller pattern, an application can create its specific, run-time, dynamic behavior. This enables more optimized applications with guaranteed behavior on any terminals

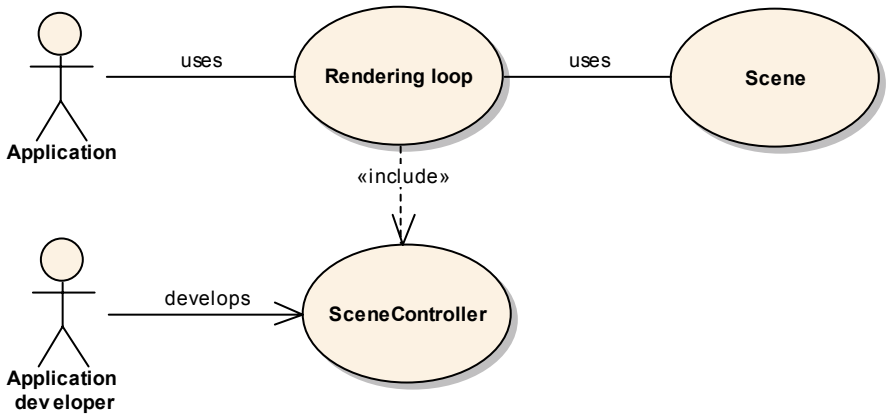


Figure 65 — Scene Controller use-cases.

Note: In Figure 65, Application stands for a terminal's MPEG-4 player.

10.4.3.4.1 SceneController – static view

Figure 66 shows the relationships between the objects used in a rendering loop.

- **SceneControllerManager** interface can be implemented by a **Compositor** object. This interface enables **SceneControllerListener** to be registered with the **Renderer**.
- **SceneControllerListener** interface defines two methods any **SceneControllerListener** (or scene controller for short) must implement.
- **Renderer** is an object that holds the **Scene** and enable access to other media used in the scene
- **Scene** contains a scene graph that is traversed at each rendering frame
- **Canvas** is an object that defines a rectangular area on the screen where painting operations will be displayed

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Note:

***Renderer**, **Canvas** and **Scene** are generic terms for the description of the scene controller pattern. An implementation may use completely different names.*

*In MPEG-4, **Scene** object may be replaced by **Compositor** object because VRML/BIFS scene graph typically retains information (such as for bindable nodes) that might be useful for scene controllers. This is implementation dependent and doesn't change the structure and the behavior of the scene controller pattern.*

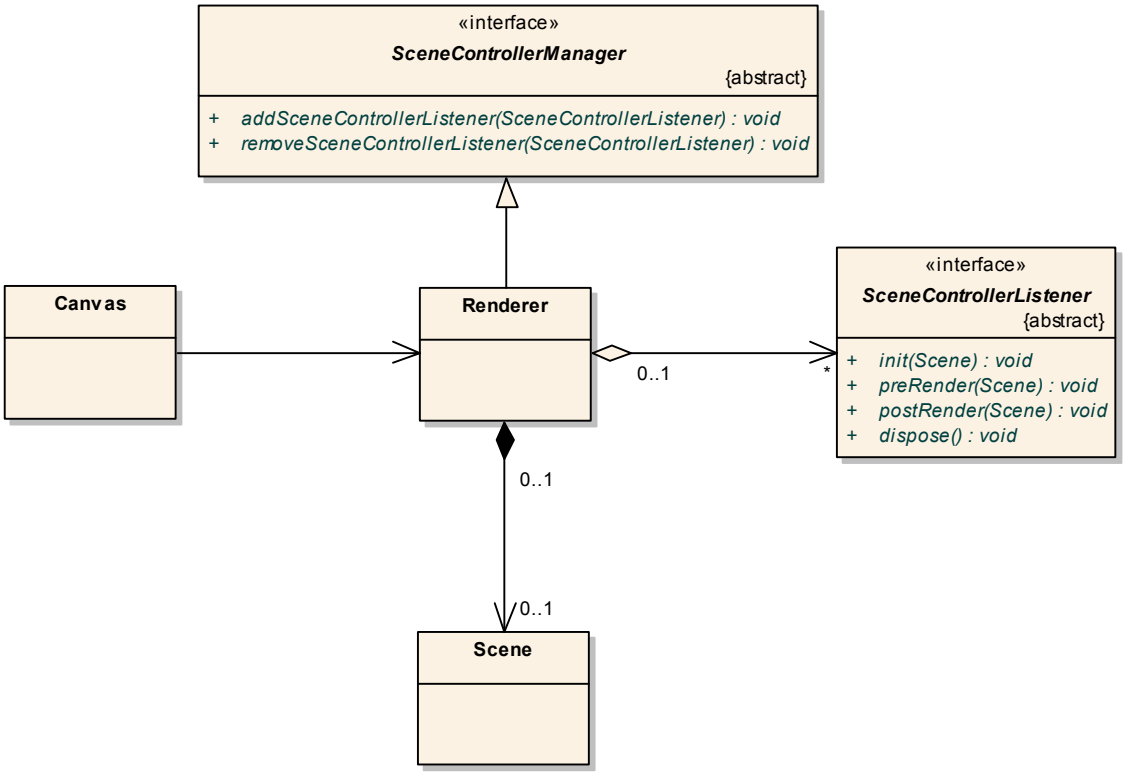


Figure 66 — Class diagram for SceneControllerFigure

10.4.3.4.2 SceneController - dynamic view

Referring to Figure 67, when a **SceneControllerListener** is registered to the compositor, *SceneControllerListener.init()* method is called to make sure its resources are correctly initialized.

At each rendering frame, the *SceneControllerListener.preRender()* method is called. Then, the scene is rendered. Finally, *SceneControllerListener.postRender()* is called.

SceneControllerListener.render() is used to control the objects to be displayed at this frame. *SceneControllerListener.postRender()* might be used for 2D layering, compositing effects, special effects, and so on once the scene is rendered.

SceneControllerListener.dispose() is used to clean up resources allocated with *init()*.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

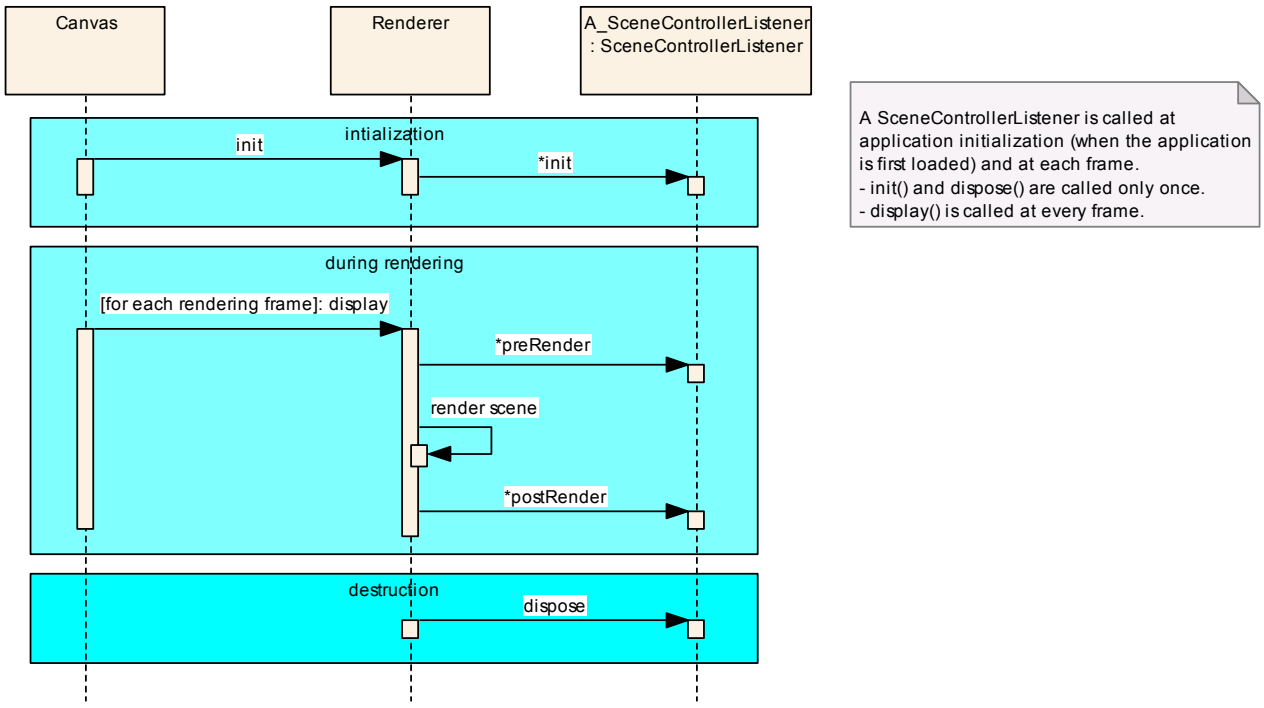


Figure 67 — Scene controller sequence.

10.4.4 Resource API

Program execution may be contingent upon the terminal configuration and its capabilities. An MPEG-J program may need to be aware of its environment, so that it is able to adapt its own execution and the execution of the various components, as they may be configured and running in the MPEG-4 terminal. The APIs in the org.iso.mpeg.mpegj.resource package can be used to monitor the system resources, to listen to exceptional conditions through the event mechanism, and handle such eventualities. The resource package helps the MPEG-4 session to adapt itself to varying terminal resources. The main components of the resource package are the Resource Manager and the event model, capability manager to monitor dynamic and static capabilities of the terminal and the terminal profile manager.

10.4.4.1 Resource Manager

The resource manager API is used for regulation of performance. This provides a centralized facility for managing resources. It is a collection of a number of classes and interfaces summarized as follows.

Interfaces	Classes	Interfaces
	Renderer	
	ResourceManager	
MPDecoderEventGenerator	MPDecoderMediaEvents	MPDecoderMediaListener
MPRendererEventGenerator	MPRendererMediaEvents	MPRendererMediaListener

10.4.4.1.1 Overview of the Event Model

For each decoder the Resource Manager would have an instantiation of a class that implements MPDecoder or a sub-interface. These decoder instantiations generate the different defined events for different conditions in the terminal. The resource manager implementation can handle events if necessary in addition to the event handlers in the application (the order of which is left to the implementation). The MPEG-J application can receive the Event handlers as byte code in the bit stream. The Renderer optionally provides notification of exceptional conditions (during rendering) and notification of frame completion when an application registers with it for this.

Apart from implicitly specifying the above event model the Resource Manager interface also provides access to the capability manager, decoders and their priorities. Given a node in the scene graph, this interface provides access to the decoder associated with that node (through its OD and ESID). It also facilitates setting and getting decoder priorities. It also enables changing a decoder associated with a node.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.4.4.2 Capability Manager

The Terminal Capability API is responsible for providing access to dynamic and static terminal resources. The separation between static and dynamic terminal capabilities has been reflected in the API (the StaticCapability and the DynamicCapability interfaces). Because applications need to be notified when terminal capabilities change, an additional interface named TerminalObserver has been defined. The CapabilityManager class implements all these interfaces. This CapabilityManager handles the terminal capabilities. It is responsible to register/deregister and get/set capabilities. This design solution allows developers to dynamically handle an unlimited number of capabilities without the burden to manage them directly.

In order for the audio-visual session to respond gracefully to these situations, MPEG-J provides mechanisms for an MPEG-J session object to catch dynamic changes to terminal information and modify its behavior. Upon initialization, an MPEG-J session object may subscribe with the CapabilityManager object to receive notification of changes to dynamic terminal information that is important to that particular audio-visual session. If such information does change during an audio-visual session, the CapabilityManager will notify the MPEG-J session of the change through standardized interfaces. Then the MPEG-J session may respond in the manner prescribed by the content creator. For example, consider an MPEG-J scene representing an MPEG-4 video sequence of a newscaster and a background news clip coded as separate video objects. The content creator might desire to freeze the news clip whenever the actual display frame rate is too slow, rather than sacrificing the quality of the newscaster in the foreground. The content creator can specify this behavior programmatically within the media by subscribing to notifications of frame-rate changes from the terminal. Then, when frame rate drops significantly due, for example, to limited CPU capacity, the media object can dynamically adapt and not continually decode the news clip and further degrade the presentation performance.

On some platforms and scenarios, it may be impossible for the terminal to guarantee the constant availability of all of its resources. For example:

A wireless multimedia unit may encounter widely varying communication capacities, including intermittent connections.

A general-purpose computer may experience varying load factors as other processes run on the system.

Audio-visual sessions with content generated at multiple sources may cause resource contention on the terminal.

Due to the large and increasing number of terminal capabilities all the capability that a terminal support is not defined. The Capability class is used to handle terminal capability in a generic way. Each terminal capability is mapped to a subclass of Capability and then managed by the CapabilityManager class. In this way the CapabilityManager class is able to handle a variable number/type of capabilities without the need to modify/extend it hence to modify the applications that use the CapabilityManager class.

Capability values are not mapped to a specific type such as String but they are handled as generic Java Objects. Through the Java Reflection API (java.lang.reflect) every type (possibly added at runtime and not at compile time) can be handled. This facility allows developers to use real types instead of flat types while maintaining the Capability class generic.

10.4.4.3 Terminal Profile Manager

The purpose of the Profile API is to provide a facility that allows applications to find out what is the profile/level supported by the terminal where the application runs. Once an application knows the terminal profile/level it can decide how to behave and what capabilities can operate in the terminal environment. The TerminalProfileManager class allows applications to query the terminal profiles.

10.4.5 Decoder API

The Decoder API facilitate basic control of all the installed decoders in an MPEG session. The decoder associated with a specific node can be queried through the Resource Manager interface. The MPDecoder is an interface that abstracts the most generic decoder.

The MPDecoder APIs allow starting, stopping, pausing, and resuming a decoder. It also facilitates attaching and detaching streams from a decoder using the ESDescriptor. The Descriptor of the currently attached stream can also be obtained.

The decoder attached to a specific node and ESID can be changed with another decoder of the same type. The Resource Manager facilitates getting a list of available decoders of a specific type and also changing one decoder for another, provided they are of the same type.

10.4.6 Net API

The Network APIs intend to allow the control of the network component of the MPEG-4 player. Through these APIs Java applications can interact with network entities. Due to level of abstraction provided by the MPEG-J Network APIs (and, in

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

turn provided by the DMIF interface), the applications can be unaware of the details of the network connections being used (LAN, WAN, Broadcast, local disks, etc.,) to access to a service.

MPEG-J network APIs do not allow full arbitrary usage of the DMIF and Sync Layers to avoid architectural inconsistencies and duplication of tools.

The functionality provided by the current API can be split into two major groups:

Network query. The ability to perform requests to the network module in order to get statistical information about the DMIF resources used by the MPEG-4 player has been recognised as an important feature.

Channels control. A simple channel control mechanism is also provided. Using this feature an MPEG-J application can temporarily disable or enable existing Elementary Stream channels without any negative influence on the rest of the player. This feature fits with one of the general requirements of MPEG-J: the capability to allow graceful degradation under limited or time varying resources.

10.4.7 Section Filter and Service Information APIs

This subclause refers to APIs normatively in the DAVIC 1.4.1 Part 9, specification. These APIs are Section Filtering, Service Information, Resource Notification, and MPEG Component APIs, which are further described in this section. A compliant MPEG-J Terminal apart from implementing the APIs defined in this document shall also implement the APIs referred normatively in this section.

10.4.7.1 The Service Information (SI) API

This API (org.davic.net.dvb.si) allows inter-operable applications to access service information data from MPEG-2 streams. One example of such applications would be electronic program guides. This API is a relatively high level API allowing applications to access information from the SI tables in a clean and efficient way. The specification of this API is defined by ETSI DI/ MTA-01074, entitled Application Programming Interface (API) for DAVIC Service Information.

10.4.7.2 The MPEG-2 Section Filter API

The objective of this API (org.davic.mpeg.sections) is to provide a general mechanism allowing access to data held in MPEG-2 private sections. This provides a mechanism for inter-operable access to data, which is too specialized to be supported by the high level DVB-SI API or which is not actually related to service information. The definition of the MPEG-2 section filter API is in Annex E of DAVIC 1.4 Part 9 specification. The API definition does not specify the lengths of the section filtering patterns. For those methods which do not specify an offset, the length of the section filtering pattern arrays shall be 8 with their mapping on to the section header as described in the last section of Annex E. For those methods, which include an offset, the length of the section filtering pattern arrays shall be 7. The API definition does not specify the efficiency or effectiveness of the section filtering process. If filtering is happening with filters set beyond the 10th byte of the total section, filtering throughputs must be supported as in DAVIC part 10, section 115.3 with the restriction that support for filtered throughputs of more than 2 Mbits/second is not mandatory.

10.4.7.3 The Resource Notification API

The section filter API uses a resource notification API in the org.davic.resources package. This API provides a standard mechanism for applications to register interest in scarce resources and to be notified of changes in those resources or removal of those resources by the environment. The description of this API is in Annex F of DAVIC 1.4 Part 9: 1998 Information Representation.

10.4.7.4 The MPEG Component API

Various MPEG related APIs use an MPEG component API in the org.davic.mpeg.sections package. This API provides a standard way of referring to standard MPEG features. The definition of the MPEG component API is in Annex G of DAVIC 1.4 Part 9 specification.

10.4.8 Detailed API Listing

The normative detailed API listing can be found in the electronic annex:

The HTML files (javadocs) of all the APIs defined in this document along with the necessary images in gif format.

It also contains the integrated API document (javadocs) of all the html files. This contains the API specification of all the APIs defined in this document.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.5 Informative: Starting the Java Virtual Machine

The objective of this section is to describe the issues involved in deciding when a Java Virtual Machine (JVM) is required to be started, in cases where it is started by the terminal. Typically, invoking or starting a Java Virtual Machine can be time consuming.

The JVM needs to be started only when there is an OD with an MPEG-J Elementary Stream. If starting the JVM is delayed till such time, it obviates the need for starting a JVM in an MPEG-J terminal when there is no MPEG-J ES. By changing the time at which the OD corresponding to the MPEG-J ES is received the terminal, the time at which the JVM is started can be controlled. By sending the OD sufficiently ahead it can be ensured that terminals of varying compute resources can still start the JVM by the required time.

It is not necessary to start the MPEG-J session until it is clear that MPEG-J application programs will be received and are to be executed. When an MPEG-J stream is received right in the beginning the JVM is instantiated and the MPEG-J "decoder" is set up along with all the other Elementary Stream decoders that are essential for that MPEG-4 session. The MPEG-4 Player is started only after this is done. This could result in some delay.

If the OD of MPEG-J elementary streams received by the terminal later than the ODs of the other elementary streams, the terminal could start the MPEG-4 session while the MPEG-J decoder will be setup only after the reception of the OD of MPEG-J ES. However, there is no implicit relationship in the time instants between when an MPEG-J ES is received and its corresponding Object Descriptor.

10.6 Informative: Examples of MPEG-J API usage

This section includes code snippets on the usage of the different categories of MPEG-J APIs. The purpose of this section is, however, not to show typical applications. The scope of this section is to enable a user to quickly get started on using MPEG-J APIs.

10.6.1 Scene APIs :

Examples in this subclause show how to use the scene APIs. Given below is an example that shows how an MPEGlet can access, modify, and created nodes in the scene.

```
import org.iso.mpeg.mpegj.resourceManager.*;
import org.iso.mpeg.mpegj.scene.*;
import org.iso.mpeg.mpegj.*;
import org.iso.mpeg.mpegj.net.*;

public class Scene_Example implements MPEGlet
{
    MpegjTerminal mpegjTerminal;
    Scene m_scene;
    MySceneListener m_sceneListener;
    MyTranslate m_translate;
    MyEventOutListener m_eventoutListener;
    MyNodeCreator m_nodeCreater;
    // other declarations

    public void init()
    {
        mpegjTerminal = new MpegjTerminal(MPEGlet);
        // all the initialization goes here for net and resource
         //managers

         //getting a reference to the Scene object from the //MpegjTerminal object
      try {
            m_scene = m_sceneListener.getScene( mpegjTerminal );
        }catch( MPEGJException mpegj_exp ) { }
        catch ( InterruptedException ie ) { }
       }

       public void run() {
        //to access and modify a field in the BIFS scene
    try {
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            m_translate.translate(m_scene);
        }catch( MPEGJException mpegj_exp ) { }

         //to receive notifications of changes to the scene
    try {
            m_eventoutListener.register(m_scene);
        }catch( MPEGJException mpegj_exp ) { }

       // creating and adding a new node (in this case, a Box node) to //the Scene
    try {
            m_modeCreator.createBox(m_scene);
        }catch( MPEGJException mpegj_exp ) { }

    }

    public void stop(){}
    public void destroy(){}
}
```

10.6.1.1 MySceneListener

The Scene object is the sole entry point for accessing and modifying the BIFS scene. This example shows how to get a reference to the Scene object from the MpegjTerminal object.

```
import org.iso.mpeg.mpegj.*;
import org.iso.mpeg.mpegj.scene.*;

/**
    Gets the scene from the MpegjTerminal.
 */
public class MySceneListener implements SceneListener {
    private Scene m_scene;

    /**
        Waits for and returns the scene associated with the given MpegjTerminal.

        @param terminal the MpegjTerminal containing the scene
        @return the scene
        @exception MPEGJException if an MPEG-J failure occurs
        @exception InterruptedException if interrupted while waiting for the scene
     */
    public synchronized Scene getScene(MpegjTerminal terminal)
            throws MPEGJException, InterruptedException {
        SceneManager mgr = terminal.getSceneManager();

        mgr.addSceneListener(this);

        while (m_scene == null) {
            wait();
        }

        return m_scene;
    }
    /**
        Called back by the SceneManager to provide the scene.
     */
    public synchronized void notify(int what, Scene scene) {
        if (what == SceneListener.Message.SCENE_READY) {
            m_scene = scene;
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            notify();
        }
    }
}
```

10.6.1.2 MyTranslate

This example shows how to access and modify a field in the BIFS scene. Firstly, the value of the translation field of a Transform2D node is printed out. Secondly, the example modifies the translation field by sending a new value to its eventIn. Note that a reference to the Transform2D node was obtained by calling the `scene.getNode()` method with a parameter of 1, which must correspond to the DEF ID of the Transform2D node in the scene.

```
import org.iso.mpeg.mpegj.*;
import org.iso.mpeg.mpegj.scene.*;

/**
    Translates the translation field of a DEF 1 Transform2D node.
 */
public class MyTranslate {
    /**
        Translates the translation field of node 1.

        @param scene the scene
        @exception MPEGJException if an MPEG-J failure occurs
     */
    public void translate(Scene scene) throws MPEGJException {
        Node node = scene.getNode(1);

        // Check that the node is a Transform2D.
        if (node.getNodeType() != NodeType.Transform2D) {
            System.err.println("Node 1 is not a Transform2D!");
            return;
        }

        // Get the current translation and print it.
        int outID = EventOut.Transform2D.translation;
        SFVec2fFieldValue translationEventOut =
            (SFVec2fFieldValue) node.getEventOut(outID);
        float[] translation = translationEventOut.getSFVec2fValue();
        System.out.println("Node 1 translation is " +
            translation[0] + "," + translation[1]);

        // Calculate a new translation.
        final float[] newTranslation = {
            translation[0] + 2,
            translation[1] + 2
        };

        // Set the new translation via the eventIn.
        int inID = EventIn.Transform2D.translation;
        SFVec2fFieldValue translationEventIn =
            new SFVec2fFieldValue() {
                public float[] getSFVec2fValue() {
                    return newTranslation;
                }
            };
        node.sendEventIn(inID, translationEventIn);
    }
}
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.6.1.3 MyEventOutListener

This example shows how to receive notifications of changes to the scene. The MyEventOutListener object registers as a listener of the fraction_changed field of a TimeSensor node. When the field value changes, the `notify()` method of the MyEventOutListener object is called with the new value.

```
import org.iso.mpeg.mpegj.*;
import org.iso.mpeg.mpegj.scene.*;

/**
    Listens to the fraction_changed field of a DEF 2 TimeSensor.
 */
public class MyEventOutListener implements EventOutListener {
    /**
        Registers as a listener on the fraction_changed field.

        @param scene the scene
        @exception MPEGJException if an MPEG-J failure occurs
     */
    public void register(Scene scene) throws MPEGJException {
        Node node = scene.getNode(2);

        // Check that the node is a TimeSensor.
        if (node.getNodeType() != NodeType.TimeSensor) {
            System.err.println("Node 2 is not a TimeSensor!");
            return;
        }

        // Add ourselves as a listener.
        node.addEventOutListener(EventOut.TimeSensor.fraction_changed, this);
    }

    /**
        Called back when the fraction_changed field changes.

        @param outID the eventOut identifier
        @param newValue the new value of the field
     */
    public void notify(int outID, FieldValue newValue) {
        float fractionChanged = ((SFFloatFieldValue) newValue).getSFFloatValue();
        System.out.println("fraction_changed=" + fractionChanged);
    }
}
```

10.6.1.4 MyNodeCreator

This example demonstrates how a new node (in this case, a Box node) can be created and added to the scene. This is achieved by setting the value of a SFNode field with an object implementing the NewNode interface. This example uses a utility class called MyNewNode that implements the NewNode interface. The MyNewNode class exposes the attributes of the Box node including its node type, DEF ID and field values (in particular, the size field). The value of the size field is represented using another utility class called MySFVec3f that implements the SFVec3fFieldValue interface. Once the MyNewNode object has been created, it can be added to the scene by sending it as an eventIn to the geometry field of a Shape node.

```
import org.iso.mpeg.mpegj.*;
import org.iso.mpeg.mpegj.scene.*;
import java.util.Hashtable;

/**
    Sets the geometry of a DEF 3 Shape node to a Box node with size 2,3,4.
    Uses two extra utility classes: MySFNode and MySFVec3f.
```

Copyright International Organization for Standardization © ISO/IEC 2005 All rights reserved
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
 */
public class MyNodeCreator {
    public void createBox(Scene scene) throws MPEGJException {
        Node shapeNode = scene.getNode(3);

        // Create the representation of the Box node.
        MyNewNode box = new MyNewNode(NodeType.Box, 0);

        // Add the size field to the representation.
        box.setField(Field.Box.size, new MySFVec3f(2, 3, 4));

        // Send the Box node as an eventIn to the geometry field.
        shapeNode.sendEventIn(EventIn.Shape.geometry, box);
    }
}

/**
    Utility class to represent a new node.
 */
class MyNewNode implements SFNodeFieldValue, NewNode {
    int m_type;
    int m_id;
    Hashtable m_fields = new Hashtable();
    NodeValue m_value;

    MyNewNode(NodeValue value) {
        m_value = value;
    }

    MyNewNode(int type, int id) {
        m_type = type;
        m_id = id;
        m_value = this;
    }

    void setField(int defID, FieldValue value) {
        m_fields.put(new Integer(defID), value);
    }

    public NodeValue getSFNodeValue() {
        return m_value;
    }

    public int getNodeType() {
        return m_type;
    }

    public int getNodeID() {
        return m_id;
    }

    public FieldValue getField(int defID) {
        return (FieldValue) m_fields.get(new Integer(defID));
    }
}

/**
    Utility class to represent an SFVec3f.
 */
class MySFVec3f implements SFVec3fFieldValue {
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    float[] m_value;

    MySFVec3f(float x, float y, float z) {
        m_value = new float[] { x, y, z };
    }

    public float[] getSFVec3fValue() {
        return m_value;
    }
}
```

10.6.2 Resource and Decoder APIs

Examples in this section show how the ResourceManager and Decoder APIs can be used to monitor resources and adapt to time varying resource conditions.

```
import org.iso.mpeg.mpegj.resourceManager.*;
import org.iso.mpeg.mpegj.decoder.*;
import org.iso.mpeg.mpegj.scene.*;
import org.iso.mpeg.mpegj.*;
import org.iso.mpeg.mpegj.net.*;

public class RM_Example implements MPEGlet
{
    MpegjTerminal mpegjTerminal;
    private ResourceManager resourceManager;
    RendererEventHandler renderer_EH;
    DecoderEventHandler decoder_EH;
    // other declarations

    Node m_node;
    Renderer m_renderer;
    java.util.Vector m_decoders;

    public void RM_Example() {
        m_decoders = new java.util.Vector( 3, 3);
    }

    public void init()
    {

        // intialize the mpegjterminal
        mpegjTerminal = new MpegjTerminal(MPEGlet);

     // other initializations go here

        // get resource manager from the mpegjterminal
        try
        {
            resourceManager = mpegjTerminal.getResourceManager();
        }catch(MPEGJException ex){ }

        // create event handlers for renderer and decoders
        renderer_EH = new EventHandler();
        decoder_EH = new EventHandler();

        // getting the renderer from the Resource Manager
        try {
            m_renderer = resourceManager.getRenderer();
        } catch( RendererNotFoundException rnfe) { }
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        // to add event listener to the Renderer
        if( m_renderer != null )
            m_renderer.addMPRendererMediaListener(renderer_EH);

        // to get the required decoder from the resource manager
        try {

                    MPdecoderImp dec1 =
                                        resourceManager.getDecoder(m_node);

            // registering the decoder as listener to the events
            dec1.addMPDecoderMediaListener(decoder_EH);

        } catch( DecoderNotFoundException  dnfe) { }
          catch (BadNodeException bne) { }

        // to change a decoder associated with a node
        // get a decoder from the available decoders list
        MPdecoder decoder = resourceManager.getAvailableDecoder(
                                                            decoder_type );

        try {
                resourceManager.changeDecoder( node, decoder)
            }catch ( DecoderNotFoundException dnfe ) { }
        catch( BadNodeException bne) { }

        // to stop a decoder and restarting again
        decoder.stop();
        decoder.start();


         // to retrieve all capabilities (static, dynamic, profile)  of // the terminal
        try {
            CapabilityManager cm =
                                resourceManager.getCapabilityManager();
        }catch (CapabilityManagerNotFoundException cmnfe ) { }
         // other code goes here
    }
       public void stop(){}
    public void destroy(){}
}
```

10.6.2.1 Listener class for Decoder Events :

```
import org.iso.mpeg.mpegj.resourceManager.*;
import org.iso.mpeg.mpegj.decoder.*;

public class DecoderEventHandler implements MPDecoderMediaListener{
    public void mPDecoderMediaHandler(MPDecoderMediaEvents event) {
        MPDecoderImp dec = ((MPDecoderImp)event.getSource());

int condition = event.getCondition();
System.out.println("Event in Decoder with condition "+ condition );
    // we can stop the decoder and restart it again
        dec.stop();
        // can change the decoder if we want
        // restart it again
        dec.start();
```


Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

```
    }
    public DecoderEventHandler( ) {
        super();
    }
}
```

10.6.2.2 Listener class for Renderer Events

```
import org.iso.mpeg.mpegj.resourceManager.*;
 public class RendererEventHandler implements MPRendererMediaListener{
    public void mPCompositeMediaHandler(MPRendererMediaEvents event){
        System.out.println("Renderer Event with condition "+event.getCondition() );
        // other code goes here
    }
    public RendererEventHandler( ) {
        super();
    }
}
```

10.6.3 Net APIs

This section illlustrates how to use the Net APIs through a simple example which enables and disables channels.

```
import org.iso.mpeg.mpegj.resourceManager.*;
import org.iso.mpeg.mpegj.scene.*;
import org.iso.mpeg.mpegj.*;
import org.iso.mpeg.mpegj.net.*;

public class Net_Example implements MPEGlet
{
    MpegjTerminal mpegjTerminal;
    private NetworkManager netManager;
    ChannelController cc;
    public void Net_Example() {
    }
    public void init()
    {
        mpegjTerminal = new MpegjTerminal(MPEGlet);
        try
        {
                    netManager = mpegjTerminal.getNetworkManager();
        } catch(NetworkManagerNotFoundException ex){ }
        catch(MPEGJException ex){ }

              // to get the channel controller used to enable/ disable //the channels
        cc = netManager.getChannelController();
        // to enable a channel
        cc.enable( serviceSessionID, channelID);
        // to disable a channel
        cc.disable( serviceSessionID, channelID);

    }
    public void stop(){}
    public void destroy(){}
}
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.6.4 Section Filtering APIs

This shows a simple example of how the section information can be extracted from the MPEG-2 Transort Stream.

```
import org.iso.mpeg.mpegj.*;
import java.lang.Boolean;

public class SI_SF_Example implements MPEGlet
{
    MpegjTerminal mpegjTerminal;
    SimpleSectionFilter ssFilter;
    SectionFilterListener sfListener;
    int milliSecs;

  // Class public methods
  public SI_SF_Example() {

    // intialize the mpegjterminal
    mpegjTerminal = new MpegjTerminal(this);
    // other initializations go here

    ssFilter = new SimpleSectionFilter();
    sfListener = new SectionFilterListener();

    // Specify an object to be notified of events relating to this SectionFilter object.
    ssFilter.addSectionFilterListener( sfListener);

    //Set the time-out for this section filter
    ssFilter.setTimeOut( milliSecs);

    // create mask and value parameters
    byte[] posValue = new byte[12];
    byte[] posMask  = new byte[12];
    for ( int i=0; i<12; i++ )
    {   posValue[i] = 0;
        posMask[i]  = 0;
    }
    posMask[0] =  (byte)0xFF; // only check first byte
    posValue[0] = (byte)0;    // table_id PAT

    //sets the SectionFilter object as filtering only for sections matching a specific PID
and
    //table_id, and where contents of the section match the specified filter pattern.
    ssFilter.startFilter
                ( 0         // index, the number of this section filter
                , 100       // id,    uniquely identifying this filter action
                , ssFilter  // the listener to receive the events indicating a new section
has arrived
                , 0         // PID,   in the case of the PAT 0
                , posMask   // mask,  which bits to check
                , posValue  // value, the value checked bits should have
                , null // neg masking not done, always call this function in
SommitSectionFilter, other startfilter methods are incorrect.
                , null
                );

    Section m_section = ssFilter.getSection()
    try {
        byte[] m_data = m_section.getData();
    } catch (NoDataAvailableException ndae) { }
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    // sections matching this SectionFilter object will stop.
    ssFilter.stopFiltering();
  }
  public void init() {}
  public void stop(){}
  public void destroy(){}
}
```

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex A
(normative)

Curve-based animators

A.1 Overview

An animated object follows an *animation path* over a period of time. The path is defined in the local coordinate system of the animated object. At frame *t* (or at time *t*), the object is at position $\mathbf{v}(t) \in \Re^3$ and has orientation represented by a unit quaternion $\mathbf{q}(t) \in \Re^4$. A *keyframe animation* is specified by pairs $(t_i, \mathbf{v}_i, \mathbf{q}_i)$, where t_i is called a *keyframe*, $\mathbf{v}_i$ is the key value i.e. the position the object must be at time t_i and $\mathbf{q}_i$ the object's orientation at time t_i . An animation is completely specified by

$$\begin{cases} \mathbf{v}(t) &= \mathbf{C}(f(t)) = \mathbf{C} \circ f(t) \\ \mathbf{q}(t) &= \mathbf{Q}(f(t)) = \mathbf{Q} \circ f(t) \end{cases}$$

Eq. 2 – Animation model.

where $\mathbf{C}(u)$ is the animation path with parameter u , $\mathbf{Q}(u)$ the orientation path, and $f(t)$ is the *timeline*. The animation path is often a succession of curves of different characteristics that join at key values. Let $\mathbf{C}_i(t)$ be the characteristic of curve segment *i* and $\mathbf{Q}_i(t)$ its orientation.

In the simplest case, like with VRML and BIFS interpolators nodes (PositionInterpolator, ScalarInterpolator, and so on), the content creator specifies $(t_i, \mathbf{v}_i, \mathbf{q}_i)$. The animation is piecewise linear with

$$\begin{cases} f(t) &= Id(t) = t \\ \mathbf{C}(t) &= \mathbf{C}_i(t) = \mathbf{v}_i + \dfrac{t - t_i}{t_{i+1} - t_i}(\mathbf{v}_{i+1} - \mathbf{v}_i) \quad \text{and } i = \lfloor t * (n-1) \rfloor \\ \mathbf{Q}(u) &= \mathbf{q}_{i-1} \dfrac{\sin(1-u)\Omega}{\sin\Omega} + \mathbf{q}_i \dfrac{\sin u\Omega}{\sin\Omega} \quad \text{and } \mathbf{q}_{i-1}\mathbf{q}_i = \cos\Omega \end{cases}$$

Eq. 3 – Piecewise linear animation model in VRML and BIFS.

with n the number of keyframes. The equation for $\mathbf{Q}(u)$ is known as spherical linear interpolator or *slerp*.

Often, it is needed to have more control over the timeline to create more realistic effects or even special effects. The animation framework proposes different types of timelines and path characteristics. Even if this specification was developed independently, we aligned it for convenience with similar concepts found in SMIL and SVG recommendations. However, this specification extends animation concepts in 1/2/3D with more flexibility for specifying animation paths and timelines.

Animation support in BIFS is borrowed from VRML 2.0 specification. It can be done in two ways:

1. using piecewise linear interpolators
2. programmatically using VRMLScript (in Script node) or Java language (via EAI, MPEG-J, or JSI in Script node)

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

While the second way is the most flexible, it requires usage of external interfaces and interpreted languages that imply a relative performance slow down. The first way is built in the browser and allows faster performance. The model is as follows: a timer (a TimeSensor node) generates clock ticks over a normalized timeline; a clock tick is in [0, 1] where 1 corresponds to the duration of the animation. Each clock tick is sent to an interpolator node (PositionInterpolator, ColorInterpolator, and so on) that generates a value that can be sent to any field of the same type, hence producing animation. An interpolator node specifies piecewise linear segments against time (Figure A.1 —).

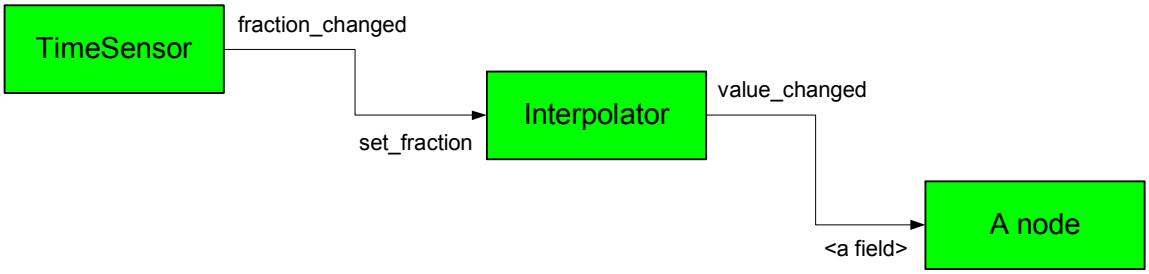


Figure A.1 — Animation in VRML/BIFS. A timer sends events to an interpolator that generates values that can be sent to anode.

The overview on NURBS curves and surfaces (subclause ISO/IEC 14496-16 subclause 4.3.2) reveals the key concepts used in this specification. These curves are used in path and velocity curve definitions as well as for orientation interpolation. Here, a review of orientation is made followed by the Animator nodes.

A.2 Animating object orientation

VRML and BIFS proposed the OrientationInteprolator node that interpolates $(axis, angle) = (\mathbf{n}, \theta)$ pairs between key frames, where $\mathbf{n} \in \Re^3$ is a unit vector and $\theta \in [-\pi, \pi]$ the angle around the axis. *Between two frames, the shortest path on the unit sphere is computed*, which can be translated by applying spherical linear interpolation in quaternion space.

First let's review quaternion mathematics and then a technique for C^2 interpolation in quaternion space is exposed.

The quaternion (a,b,c,d) is a point on the unit sphere S^3 in 4-space and is short-hand for $a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$ with

$$i^2 = j^2 = k^2 = ijk = -1.$$

A quaternion (a,b,c,d) is often expressed as a scalar component and a vector component $(s,\mathbf{n})$ with $s = a$ and $\mathbf{n} = (b,c,d)$. Using previous notation, the multiplication of two quaternions is

$$(s_1,\mathbf{n}_1)(s_2,\mathbf{n}_2) = (s_1 * s_2 - \mathbf{n}_1 \cdot \mathbf{n}_2, s_1\mathbf{n}_2 + s_2\mathbf{n}_1 + \mathbf{n}_1 \times \mathbf{n}_2)$$

A unit quaternion satisfies $a^2 + b^2 + c^2 + d^2 = 1$.

A point $\mathbf{p} \in \Re^3$ is rotated by a unit quaternion $(s,\mathbf{n})$ to the point $(s,-\mathbf{n})(0,\mathbf{p})(s,\mathbf{n})$, which is a quaternion with a scalar part of 0, so it can be interpreted as a 3-space point.

In particular the unit quaternion $(\cos\frac{\theta}{2}, \mathbf{n}\sin\frac{\theta}{2})$ represents the orientation of an object where the object in its canonical orientation has been rotated by θ radians about the axis $\mathbf{n}$.

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Mathematically, an orientation can be represented by either of two antipodal quaternions (by flipping $\mathbf{n}$). For motion control, only one quaternion is appropriate: the quaternion with smaller angular gap on S^3 to the previous quaternion i.e. if $q_1 q_2 < 0$, then use $q_2' = -q_2$. This makes the orientation path always perform the shortest rotation between the orientation key frames.

Spherical linear interpolation (also called *slerping* technique) is equivalent to linear interpolation between two key orientations and, as for linear interpolation, produces jerky, sharply changing motion across the keys. This is particularly important for camera flythrough where sharp changes in angular acceleration have to be avoided.

What is needed is a higher order method with C^2 continuity. Many methods have been proposed in the literature but leads to no closed form algebraic solution and, when based on slerping, generate nonrational curves, may become expensive to calculate, and may not provide C^2 continuity.

The technique proposed by Johnstone and Williams produces very good results in practice and obeys derivative continuity requirements. It uses an invertible rational mapping M between S^3 and $\Re^4$.

The transformation M^{-1} from $S^3 \rightarrow \Re^4$ is

$$\text{let } s' = sqrt(2 * (1 - s))$$
$$\begin{cases} x = a / s' \\ y = b / s' \\ z = c / s' \\ w = (1 - s) / s' \end{cases}$$

and the transformation M from $\Re^4 \rightarrow S^3$ is

$$\text{let } d = x^2 + y^2 + z^2 + w^2$$
$$\begin{cases} s = \left(x^2 + y^2 + z^2 - w^2\right) / d \\ a = 2xw/d \\ b = 2yw/d \\ c = 2zw/d \end{cases}$$

The method is as follows:

1. Apply M^{-1} to key quaternions (s, a, b, c) to obtain their resulting values in $\Re^4$ $x, y, z, w)$
2. Interpolate the resulting 4-vectors using any spline (in this specification we use natural cubic B-splines).
3. Convert the desired point to S^3 using M.

However, if your animation passes close to the pole (1, 0, 0, 0), numerical instability will happen in M^{-1}. Johnstone define "too close to the pole" if the angle between the control quaternions and (1, 0, 0, 0) is smaller than 30 degrees. His solution is to multiply all control quaternions by a quaternion $\mathbf{q}_f$ that is not within 30 degrees of any other control quaternion and use $\mathbf{q}_f$ to rotate all quaternions to a "safe" region. After interpolation, the resulting quaternion is multiplied by $\mathbf{q}_f^{-1}$. This can be done in a preprocessing step i.e. before the animation starts.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex B
(normative)

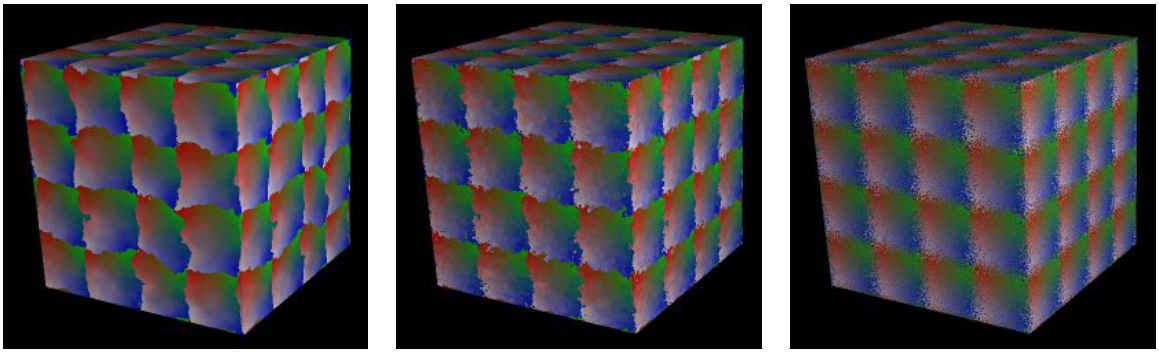
Procedural textures algorithms

B.1 Overview

The processes underlying the creation of the textures include the generation of a fractal 'plasma' field, subdivision of the texture into cells, spatial distortion of the texture, selection of colors to apply to the texture, and control of how the colors vary within a cell. The five cell types supported are shown in Figure 31.

B.2 Fractal Plasma Calculation

The plasma is a two-dimensional array with the same dimensions as the final texture ($width \times height$). It contains a single value per element ($P(i,j) \in [-0.5,0.5], \forall i \in [0, w-1], \forall j \in [0, h-1]$) that varies over the plasma in a manner reminiscent of hills and valleys. The generation of the plasma is controlled by a *seed* for a pseudo-random number generator (subclause B.4) and by a *roughness* parameter that controls how widely spaced the peaks and troughs will be (Figure B.1 —).



(a) (b) (c)

Figure B.1 — The effects of the roughness parameter: (a) 3, (b) 5, (c) 8.

The plasma array (P) is calculated as follows:

```
plasma(width, height, roughness, seed)
{
   srandom(seed);
   L = width * height;
   P[0, 0] = 0;
   w = width;
   h = height;
   rmult = 1;
   while (w > h)
   {
      step = w / 2;
      p1 = 0;
      while (p1 < width)
      {
         v = random() - 0.5;
         if (step < roughness)
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        {
           rmult *= 0.5;
           v = v * mult + (P[p1, 0] + P[(p1 + w) % width, 0]) / 2;
        }
        P[p1 + step, 0] = v;
        p1 += w;
     }
     w = w / 2;
  }
  while (h > w)
  {
     step = h / 2;
     p1 = 0;
     while (p1 < height)
     {
        v = random() - 0.5;
        if (step < roughness)
        {
           rmult *= 0.5;
           v = v * rmult + (P[0, p1] + I[0, (p1 + h) % height]) / 2;
        }
        P[0, p1 + step] = v;
        p1 += h;
     }
     h = h / 2;
  }
  // here, w==h
  while (w > 1)
  {
     step = w / 2;
     if (step < roughness)
        rmult *= 0.5;
     for (y = 0; y < height; y += w)
     {
        for (x = 0; x < width; x += w)
        {
           x1 = x;
           x2 = (x + w) % width;
           x3 = x + step;
           y1 = y;
           y2 = (y + w) % height;
           y3 = y + step;

           v1 = random() - 0.5;
           v2 = random() - 0.5;
           v3 = random() - 0.5;

           if (step < roughness)
           {
              v1 = v1 * rmult + (P[x1, y1] + P[x2, y1]) / 2;
              v2 = v2 * rmult + (P[x1, y1] + P[x1, y2]) / 2;
              v3 = v3 * rmult +
                   (P[x1, y1] + P[x2, y1] + P[x1, y2] + P[x2, y2]) / 4;
           }

           P[x3, y1] = v1;
           P[x1, y3] = v2;
           P[x3, y3] = v3;
        }
     }
     w = w / 2;
  }

  offset & scale  so all values lie between -0.5 and +0.5;
}
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

B.3 Warping

Warping is achieved using a user specified 2D curve and is used to produce large-scale variations in the texture, such as a gradual shift from bottom to top, or ripple effects. The curve itself is either a collection of straight-line segments, or a smooth Hermite spline. An example of the effects of warping is shown in Figure B.2 —.

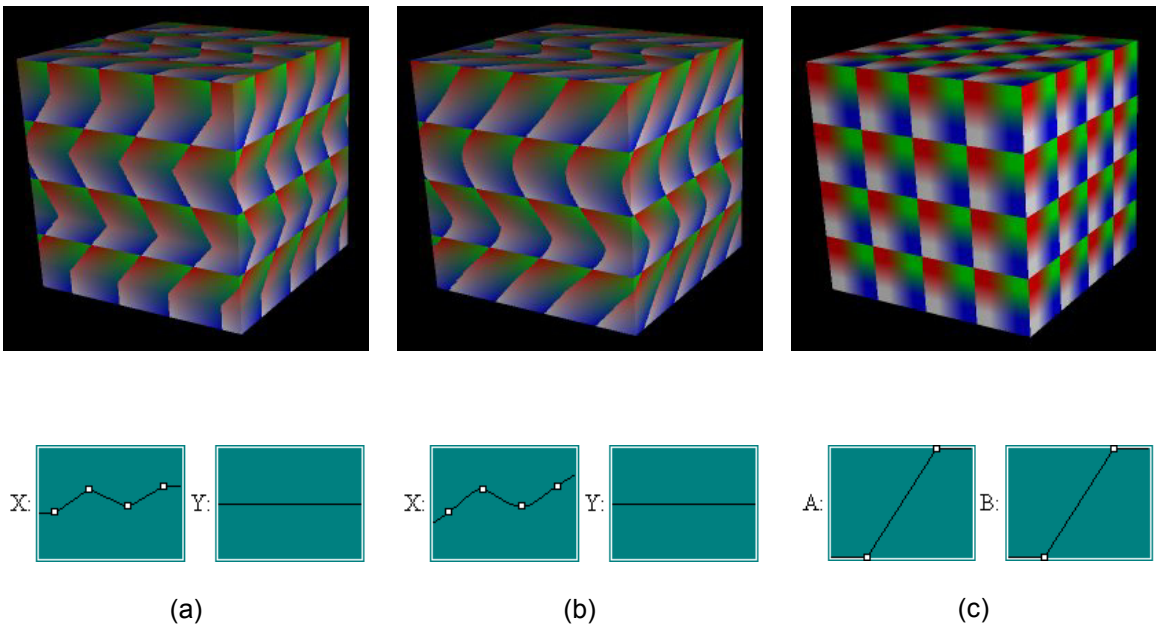


Figure B.2 — The effect of warping: (a) linear x wrap, (b) smooth x warp, (c) linear a & b warp.

```
remap(v, WarpMap[], smooth)
{
n = WarpMap number of elements;

   if (n == 0)
      return 0.5;
   if (n == 1)
      return WarpMap[0].y;

   if (smooth)    // Hermite spline interpolation
   {
      MinDx = 0.001;

      if (f < WarpMap[0].x) // Extrapolate before first knot
      {
         dy = WarpMap[1].y - WarpMap[0].y;
         dx = WarpMap[1].y - WarpMap[0].x;
         if (dx < MinDx) dx = MinDx;
         f = WarpMap[0].y - (WarpMap[0].x - f) * dy / dx;
         return f;
      }
      if (f > WarpMap[n-1].x) // Extrapolate after last knot
      {
         dy = WarpMap[n - 1].y - WarpMap[n - 2].y;
         dx = WarpMap[n - 1].x - WarpMap[n - 2].x;
         if (dx < MinDx) dx = MinDx;
         f = WarpMap[n - 1].y + (f - WarpMap[n - 1].x) * dy / dx;
         return f;
      }
      for (i = 0; i < n-1; i++) // Find knot interval
      {
         if (f >= WarpMap[i].x && f <= WarpMap[i + 1].x && WarpMap[i].x != WarpMap[i +
1].x)
```

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      {
         dx1 = (WarpMap.x[i + 1] - WarpMap.x[i]);

         f = (f - WarpMap.x[i]) / dx1;

         y1 = WarpMap.y[i];
         y2 = WarpMap.y[i + 1];
         if (i > 1)
         {
            y0 = WarpMap.y[i - 1];
            dx0 = WarpMap.x[i] - WarpMap.x[i - 1];
         }
         else
         {
            y0 = y1 - (y2 - y1);
            dx0 = dx1;
         }

         if (i < n - 2)
         {
            y3 = WarpMap.y[i + 2];
            dx2 = WarpMap.x[i + 2] - WarpMap.x[i + 1];
         }
         else
         {
            y3 = y2 + (y2 - y1);
            dx2 = dx1;
         }

         if (dx0 < MinDx) dx0 = MinDx;
         if (dx2 < MinDx) dx2 = MinDx;

         m0 = (y1 - y0) * dx1 / dx0;
         m1 = (y2 - y1) * dx1 / dx1;
         m2 = (y3 - y2) * dx1 / dx2;

         m0 = (m0 + m1) * 0.5;
         m2 = (m2 + m1) * 0.5;

         f2 = f * f; // f^2
         f3 = f * f2; // f^3

         // Create a Hermite spline

         y = y1 * (2 * f3 – 3 * f2 + 1);
         y += y2 * (-2 * f3 + 3 * f2);
         y += m0 * (f3 – 2 * f2 + f);
         y += m2 * (f3 - f2);
         return y;
      }
   }
}
else // Linear interpolation
{
   if (f < WarpMap[0].x)// Extrapolate before first knot
      return WarpMap[0].y;
   if (f > WarpMap[n - 1].x) // Extrapolate after knot
      return WarpMap[n-1].y;
   for (i = 0; i < n - 1; i++)  // Find knot interval
   {
      if (f >= WarpMap[i].x && f <= WarpMap[i + 1].x &&
          WarpMap[i].x != WarpMap[i + 1].x)
      {
         f = (f - WarpMap[i].x) / (WarpMap[i + 1].x - WarpMap[i].x);
         return WarpMap[i].y + f * (WarpMap[i + 1].y - WarpMap[i].y);
      }
   }
   return WarpMap[n-1].y;
```


© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   }
}
```

B.4 Pseudo-random Number Generator

The pseudo-random number generator used for the plasma generation is based on the fast 32-bit linear congruential generator.

```
static unsigned int seed = 826374438;

srandom(s) // seed the pseudo-random number generator
{
   ci = (unsigned int)(s * 0xFFFF);

   // Run through three cycles to remove obvious visual artifacts
   // that occur when input is too closely related to output
   ci = ci * 1664525 + 1013904223;
   ci = ci * 1664525 + 1013904223;
   ci = ci * 1664525 + 1013904223;

union { unsigned int Int; float Float; } IEEEFloatMaker;

   IEEEFloatMaker.Int = (ci & 0x7FFFFF) | 0x3F800000;

   seed = IEEEFloatMaker.Float - 1.0;
}

random() // return random value in [-0.5, 0.5)
{
   seed = seed * 1664525 + 1013904223;

union { unsigned int Int; float Float; } IEEEFloatMaker;

   // Or in 23 bits of mantissa, now [1.0, 2.0)
   IEEEFloatMaker.Int = (seed & 0x7FFFFF) | 0x3F800000;

   // Return [-0.5, 0.5)
   return(IEEEFloatMaker.Float - 1.5);
}

nonrandom(a, b) // return a repeatable, coordinate based, "random" value
{
   srandom(a);
   srandom(b + random());
   return(random());
}
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

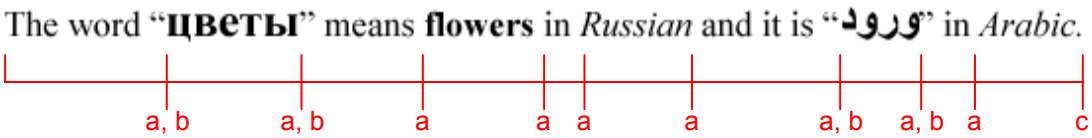
Annex C
(informative)

Text Processing in BIFS

In order to allow for support of all languages, full Unicode character set and rich formatting capabilities at a reasonable implementation cost, MPEG-4 employs distributed system design architecture to separate the functional implementation of the language script support services between the MPEG-4 encoder and decoder. An encoder will be responsible for support of the application specific rich text formatting, Unicode language scripts processing and separation of the input character strings into the 'runs' of text.

> NOTE: A 'run' of text is normally a maximum of one line in length and consists of a text string formatted in a single font at a particular size, in a single direction, in a particular script and a particular language system.

A MPEG-4 encoder should parse and tag different runs of text and should encode them in clusters that represent a single sentence or a paragraph, e.g. using **Layout** node, while each run of text should be encoded separately using **Text** nodes with the corresponding **FontStyle** or **XFontStyle** node. An example of complex text and its separation in the multiple text runs is presented below:

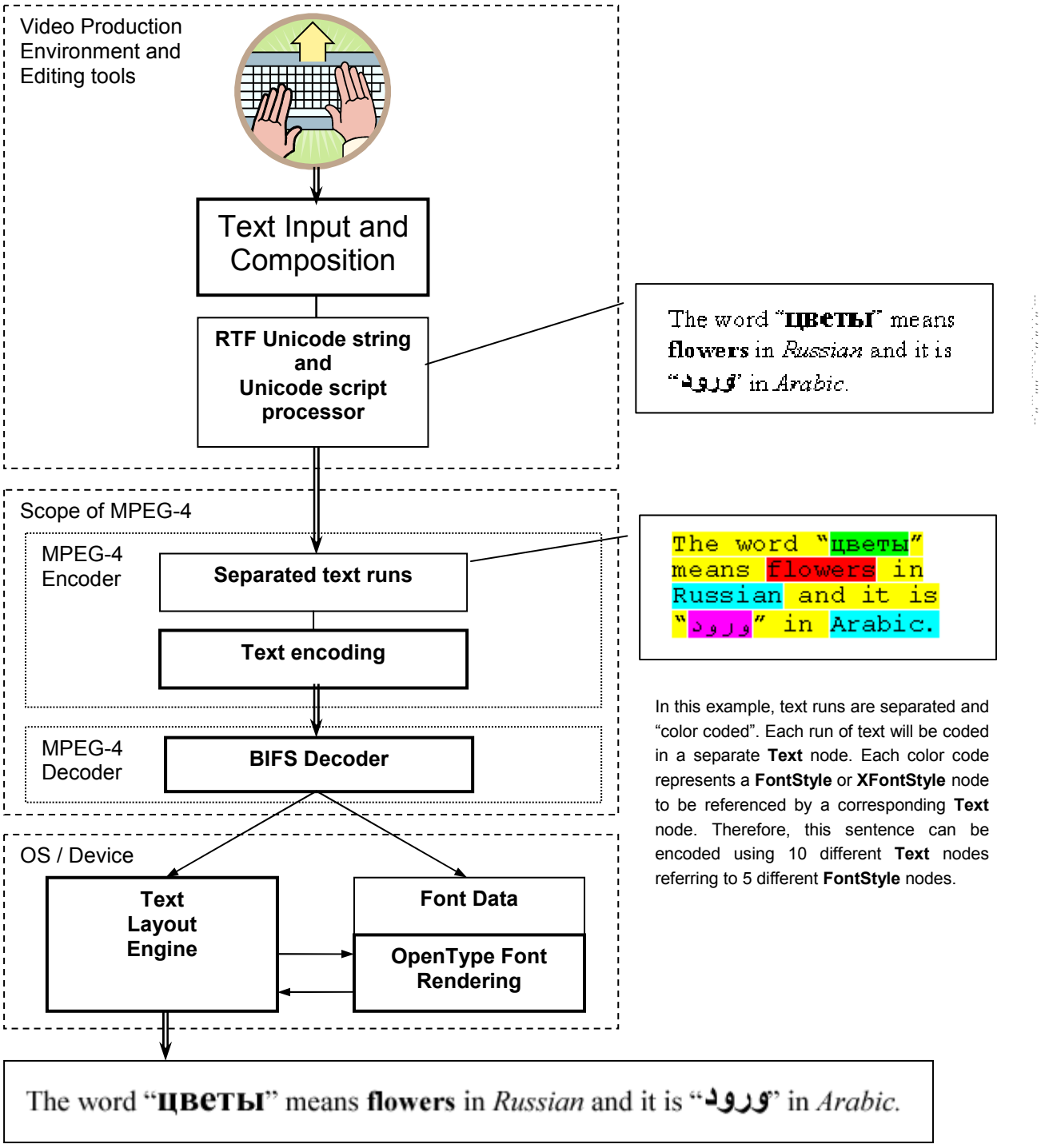


In the presented example there are three types of run boundary, indicated by the small red letters:

a) Change of font (bold to plain, plain to italic, italic to plain).
b) Change of script, language system and direction. Any one of these changes is sufficient to require the start of a new run. In many multilingual texts, a change of script might also occasion a change in font, but for this example it is presumed that this is a multiscript font with both Latin, Cyrillic and Arabic support.
c) End of line (the end of the line may also be a change of font, script, language system and direction).

An MPEG-4 terminal should be responsible for the implementation of the text layout services based on individually encoded text runs. These services should be supported by the font rendering subsystem, including advanced font rendering capabilities such as glyph substitution and positioning. Final layout of the text will be done based on the fields of the **FontStyle**, **XFontStyle**, **Text** and **Layout** nodes and the rules imposed by a specific language script (in case when a text run is presented in a language that imposes additional requirements).
The illustration of the proposed solution is presented on the following figure.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved



Composition of the text presented in this example can be implemented in a single **Layout** node with ten **Text** nodes defined as children referring to five different **FontStyle** nodes. Original composition point of the text line should be defined either directly or by the parent node, the text will be positioned using font rendering information taking into account font metrics, requested text size and style. Individual text runs encoded in **Text** nodes, will be processed using font, direction, spacing and language rules defined by the corresponding **FontStyle** or **XFontStyle** node and the placement of the text run and final composition of the text line will be done based on the direction and justification rules defined in the **Layout** node.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex D
(informative)

Patent statements

The International Organization for Standardization and the International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this part of ISO/IEC 14496 may involve the use of patents.

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured the ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patents right are registered with ISO and IEC. Information may be obtained from the companies listed below.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 14496 may be the subject of patent rights other than those identified in this annex. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Organisation
AGFA Monotype
Microsoft Corporation

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Bibliography

[1] Jérôme Daniel, "Spatial Sound Encoding Including Near Field Effect: Introducing Distance Coding Filters and a Viable, New Ambisonic Format", AES 23rd International Conference, Copenhagen, 2003

[2] Michael A. Gerzon, "Ambisonics in Multichannel Broadcasting and Video", J. Audio Eng. Soc. Vol. 33, 1983 October 8-12

© ISO/IEC 2005 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ICS 35.040

Price based on 502 pages

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale