INTERNATIONAL STANDARD

ISO/IEC 14496-1

Second edition
2001-10-01

Information technology — Coding of audio-visual objects —

Part 1: Systems

Technologies de l'information — Codage des objets audiovisuels —

Partie 1: Systèmes

Reference number
ISO/IEC 14496-1:2001(E)



© ISO/IEC 2001

Hulu, LLC - Ex. 1024, Page 000001
IPR2025-00082 (Hulu, LLC v. Piranha Media Distribution, LLC)

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 2001

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.ch
Web www.iso.ch

Printed in Switzerland

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000002

Contents

Page

0 Introduction viii
0.1 Overview viii
0.2 Architecture viii
0.3 Terminal Model: Systems Decoder Model x
0.4 Multiplexing of Streams: The Delivery Layer x
0.5 Synchronization of Streams: The Sync Layer x
0.6 The Compression Layer xi
0.7 Application Engine xii
1 Scope 1
2 Normative references 1
3 Additional reference 2
4 Terms and definitions 2
5 Abbreviations and Symbols 7
6 Conventions 8
7 Systems Decoder Model 8
7.1 Introduction 8
7.2 Concepts of the systems decoder model 9
7.3 Timing Model Specification 10
7.4 Buffer Model Specification 12
8 Object Description Framework 14
8.1 Introduction 14
8.2 Common data structures 15
8.3 Intellectual Property Management and Protection (IPMP) 17
8.4 Object Content Information (OCI) 19
8.5 Object Descriptor Stream 21
8.6 Object Descriptor Components 24
8.7 Rules for Usage of the Object Description Framework 46
8.8 Usage of the IPMP System interface 55
9 Scene Description 58
9.1 Introduction 58
9.2 Concepts 60
9.3 BIFS Syntax 74
9.4 Node Semantics 133
10 Synchronization of Elementary Streams 226
10.1 Introduction 226
10.2 Sync Layer 227

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000003

10.3 DMIF Application Interface **236**
11 MPEG-J **236**
11.1 Introduction **236**
11.2 Architecture **237**
11.3 MPEG-J Session **239**
11.4 Delivery of MPEG-J Data **240**
11.5 MPEG-J API List **243**
12 Multiplexing of Elementary Streams **249**
12.1 Introduction **249**
12.2 FlexMux Tool **249**
13 File Format **255**
13.1 Introduction **255**
13.2 File organization **260**
13.3 Extensibility **284**
14 Syntactic Description Language **285**
14.1 Introduction **285**
14.2 Elementary Data Types **285**
14.3 Composite Data Types **288**
14.4 Arithmetic and Logical Expressions **292**
14.5 Non-Parsable Variables **292**
14.6 Syntactic Flow Control **292**
14.7 Built-In Operators **294**
14.8 Scoping Rules **294**
15 Profiles **294**
15.1 Introduction **294**
15.2 OD Profile Definitions **295**
15.3 Scene Graph Profile Definitions **295**
15.4 Graphics Profile Definitions **299**
15.5 MPEG-J Profile Definitions **301**
15.6 MPEG-J Profiles Tools **301**
15.7 MPEG-J Profiles **301**
15.8 MPEG-J Profiles@Levels **302**
Annex A (informative) **Bibliography** **303**
Annex B (informative) **Time Base Reconstruction** **304**
B.1 Time Base Reconstruction **304**
B.2 Temporal aliasing and audio resampling **305**
B.3 Reconstruction of a Synchronised Audio-visual Scene: A Walkthrough **305**
Annex C (normative) **View Dependent Object Scalability** **307**
C.1 Introduction **307**
C.2 Bitstream Syntax **307**

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000004

C.3 Bitstream Semantics **308**
Annex D (informative) **Registration procedure** **310**
D.1 Procedure for the request of a Registration ID (RID) **310**
D.2 Responsibilities of the Registration Authority **310**
D.3 Contact information for the Registration Authority **310**
D.4 Responsibilities of Parties Requesting a RID **310**
D.5 Appeal Procedure for Denied Applications **311**
D.6 Registration Application Form **311**
Annex E (informative) **The QoS Management Model for ISO/IEC 14496 Content** **313**
Annex F (informative) **Conversion Between Time and Date Conventions** **314**
Annex G (normative) **Adaptive Arithmetic Decoder for BIFS-Anim** **316**
Annex H (normative) **Node coding tables** **318**
H.1 Node Tables **318**
H.2 Node Definition Type Tables **341**
H.3 Node Tables for Extended Nodes **348**
H.4 Node Definition Type Tables for extended node types **355**
Annex I (informative) **MPEG-4 Audio TTS application with Facial Animation** **357**
Annex J (informative) **Graphical representation of object descriptor and sync layer syntax** **358**
J.1 Length encoding of descriptors and commands **358**
J.2 Object Descriptor Stream and OD commands **358**
J.3 IPMP stream **359**
J.4 OCI stream **359**
J.5 Object descriptor and its components **359**
J.6 OCI Descriptors **362**
J.7 Sync layer configuration and syntax **365**
Annex K (informative) **Patent statements** **366**
K.1 Patent Statements for Version 1 ..... **366**
K.2 Patent Statements for Version 2 ..... **367**
Annex L (informative) **Elementary Stream Interface** **369**
Annex M (Informative) **Definition of bodySceneGraph nodes** **371**
M.1 Introduction **371**
M.2 Detailed Semantics **371**
M.3 Overview **371**
M.4 The Nodes **371**
Annex N (Informative) **Implementation of MaterialKey node** **380**
Annex O (Informative) **Example implementation of spatial audio processing (perceptual approach)** **382**
O.1 Example algorithm implementation **382**
O.2 Elementary spectral corrector **383**
O.3 Input Filter **384**
O.4 Direct path **384**

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000005

O.5 Directional early reflections **385**
O.6 Diffuse late reverberation **385**
O.7 Setting the delays **386**
O.8 Scalability **387**
Annex P (informative) **Upstream walkthrough** **388**
P.1 Introduction **388**
P.2 Configuration **388**
P.3 Content access procedure with DAI **389**
P.4 Example **389**
Annex Q (Informative) **Layout of Media Data** **393**
Annex R (Informative) **Random Access** **394**
Annex S (Informative) **Starting the Java Virtual Machine** **395**
Annex T (Informative) **Examples of MPEG-J API usage** **396**
T.1 Scene APIs : **396**
T.2 Resource and Decoder APIs **400**
T.3 Net APIs **402**
T.4 Section Filtering APIs **403**
Annex U (Normative) **MPEG-J APIs Listing (HTML)** **405**
Annex V (Normative) **MPEG-J APIs Listing** **406**
V.1 package org.iso.mpeg.mpegj **406**
V.2 package org.iso.mpeg.mpegj.resource **413**
V.3 package org.iso.mpeg.mpegj.decoder **442**
V.4 package org.iso.mpeg.mpegj.net **454**
V.5 package org.iso.mpeg.mpegj.scene **461**

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000006

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 14496 may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

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

This second edition cancels and replaces the first edition (ISO/IEC 14496-1:1999), which has been technically revised.

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

— *Part 1: Systems*

— *Part 2: Visual*

— *Part 3: Audio*

— *Part 4: Conformance testing*

— *Part 5: Reference software*

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

— *Part 7: Optimized software for MPEG-4 visual tools*

Annexes C, G, H, U and V form a normative part of this part of ISO/IEC 14496. Annexes A, B, D, E, F and I to T are for information only.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000007

0 Introduction

0.1 Overview

ISO/IEC 14496 specifies a system for the communication of interactive audio-visual scenes. This specification includes the following elements:

1. the coded representation of natural or synthetic, two-dimensional (2D) or three-dimensional (3D) objects that can be manifested audibly and/or visually (audio-visual objects) (specified in part 1,2 and 3 of ISO/IEC 14496);

2. the coded representation of the spatio-temporal positioning of audio-visual objects as well as their behavior in response to interaction (scene description, specified in this part of ISO/IEC 14496);

3. the coded representation of information related to the management of data streams (synchronization, identification, description and association of stream content, specified in this part of ISO/IEC 14496); and

4. a generic interface to the data stream delivery layer functionality (specified in part 6 of ISO/IEC 14496).

5. an application engine for programmatic control of the player: format, delivery of downloadable Java byte code as well as its execution lifecycle and behavior through APIs (specified in this part of ISO/IEC 14496); and

6. a file format to contain the media information of an ISO/IEC 14496 presentation in a flexible, extensible format to facilitate interchange, management, editing, and presentation of the media.

The overal operation of a system communicating audio-visual scenes can be paraphrased as follows:

At the sending terminal, the audio-visual scene information is compressed, supplemented with synchronization information and passed to a delivery layer that multiplexes it into one or more coded binary streams that are transmitted or stored. At the receiving terminal, these streams are demultiplexed and decompressed. The audio-visual objects are composed according to the scene description and synchronization information and presented to the end user. The end user may have the option to interact with this presentation. Interaction information can be processed locally or transmitted back to the sending terminal. ISO/IEC 14496 defines the syntax and semantics of the bitstreams that convey such scene information, as well as the details of their decoding processes.

This part of ISO/IEC 14496 specifies the following tools:

- a terminal model for time and buffer management;
- a coded representation of interactive audio-visual scene description information (Binary Format for Scenes – BIFS);
- a coded representation of metadata for the identification, description and logical dependencies of the elementary streams (object descriptors and other descriptors);
- a coded representation of descriptive audio-visual content information (object content information – OCI);
- an interface to intellectual property management and protection (IPMP) systems;
- a coded representation of synchronization information (sync layer – SL); and
- a multiplexed representation of individual elementary streams in a single stream (FlexMux).
- an application engine (MPEG-Java - MPEG-J).

These various elements are described functionally in this subclause and specified in the normative clauses that follow.

0.2 Architecture

The information representation specified in ISO/IEC 14496-1 describes the means to create an interactive audio-visual scene in terms of coded audio-visual information and associated scene description information. The entity

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000008

that composes and sends, or receives and presents such a coded representation of an interactive audio-visual scene is generically referred to as an "audio-visual terminal" or just "terminal". This terminal may correspond to a standalone application or be part of an application system.

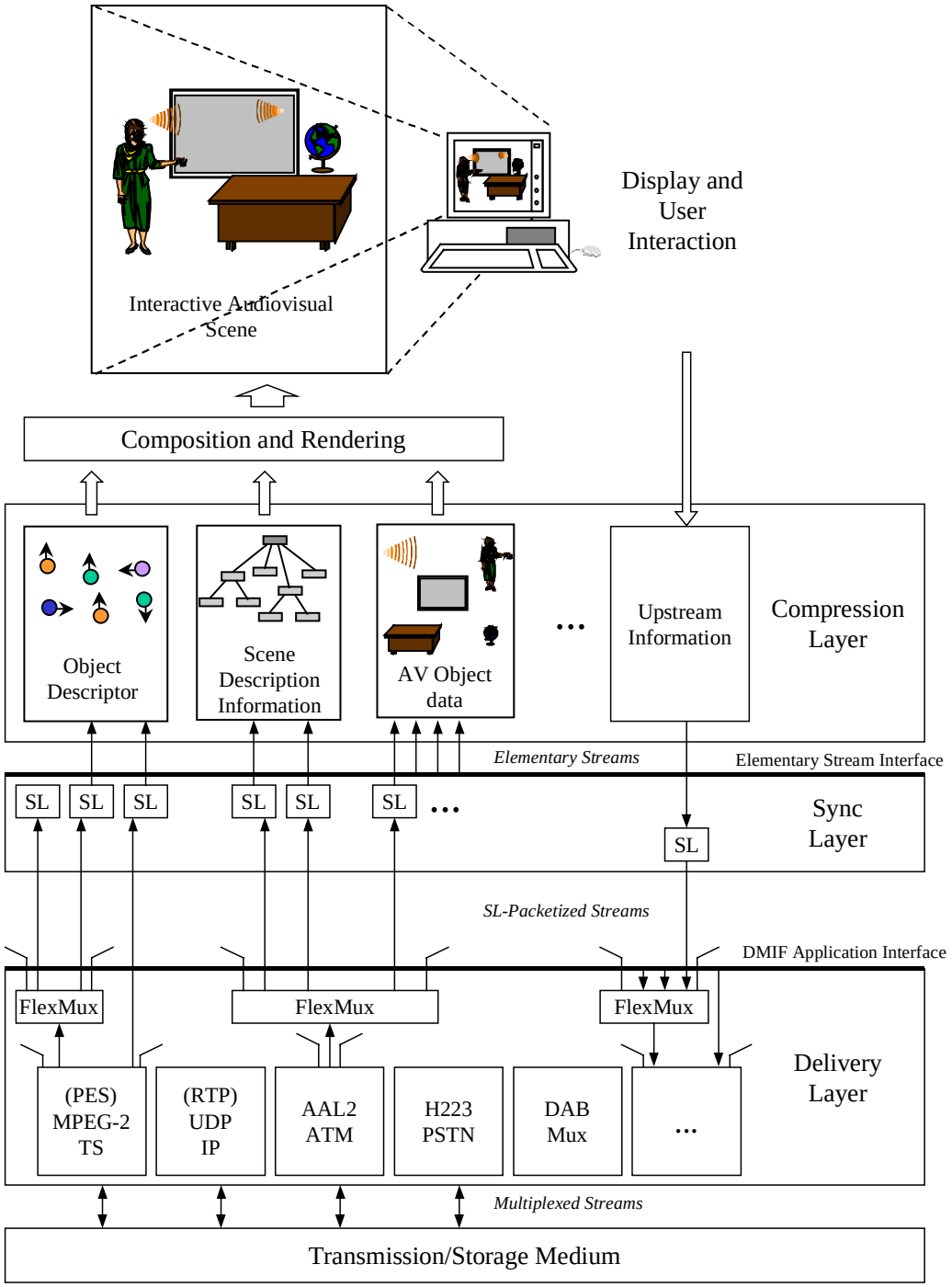


Figure 1 - The ISO/IEC 14496 terminal architecture

The basic operations performed by such a receiver terminal are as follows. Information that allows access to content complying with ISO/IEC 14496 is provided as initial session set up information to the terminal. Part 6 of ISO/IEC 14496 defines the procedures for establishing such session contexts as well as the interface to the delivery layer that generically abstracts the storage or transport medium. The initial set up information allows, in a recursive manner, to locate one or more elementary streams that are part of the coded content representation. Some of these elementary streams may be grouped together using the multiplexing tool described in ISO/IEC 14496-1.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000009

Elementary streams contain the coded representation of either audio or visual data or scene description information. Elementary streams may as well themselves convey information to identify streams, to describe logical dependencies between streams, or to describe information related to the content of the streams. Each elementary stream contains only one type of data.

Elementary streams are decoded using their respective stream-specific decoders. The audio-visual objects are composed according to the scene description information and presented by the terminal's presentation device(s). All these processes are synchronized according to the systems decoder model (SDM) using the synchronization information provided at the synchronization layer.

These basic operations are depicted in Figure 1, and are described in more detail below.

0.3 Terminal Model: Systems Decoder Model

The systems decoder model provides an abstract view of the behavior of a terminal complying with ISO/IEC 14496-1. Its purpose is to enable a sending terminal to predict how the receiving terminal will behave in terms of buffer management and synchronization when reconstructing the audio-visual information that comprises the presentation. The systems decoder model includes a systems timing model and a systems buffer model which are described briefly in the following subclauses.

0.3.1 Timing Model

The timing model defines the mechanisms through which a receiving terminal establishes a notion of time that enables it to process time-dependent events. This model also allows the receiving terminal to establish mechanisms to maintain synchronization both across and within particular audio-visual objects as well as with user interaction events. In order to facilitate these functions at the receiving terminal, the timing model requires that the transmitted data streams contain implicit or explicit timing information. Two sets of timing information are defined in ISO/IEC 14496-1: clock references and time stamps. The former convey the sending terminal's time base to the receiving terminal, while the latter convey a notion of relative time for specific events such as the desired decoding or composition time for portions of the encoded audio-visual information.

0.3.2 Buffer Model

The buffer model enables the sending terminal to monitor and control the buffer resources that are needed to decode each elementary stream in a presentation. The required buffer resources are conveyed to the receiving terminal by means of descriptors at the beginning of the presentation. The terminal can then decide whether or not it is capable of handling this particular presentation. The buffer model allows the sending terminal to specify when information may be removed from these buffers and enables it to schedule data transmission so that the appropriate buffers at the receiving terminal do not overflow or underflow.

0.4 Multiplexing of Streams: The Delivery Layer

The term delivery layer is used as a generic abstraction of any existing transport protocol stack that may be used to transmit and/or store content complying with ISO/IEC 14496. The functionality of this layer is not within the scope of ISO/IEC 14496-1, and only the interface to this layer is considered. This interface is the DMIF Application Interface (DAI) specified in ISO/IEC 14496-6. The DAI defines not only an interface for the delivery of streaming data, but also for signaling information required for session and channel set up as well as tear down. A wide variety of delivery mechanisms exist below this interface, with some of them indicated in Figure 1. These mechanisms serve for transmission as well as storage of streaming data, i.e., a file is considered to be a particular instance of a delivery layer. For applications where the desired transport facility does not fully address the needs of a service according to the specifications in ISO/IEC 14496, a simple multiplexing tool (FlexMux) with low delay and low overhead is defined in ISO/IEC 14496-1.

0.5 Synchronization of Streams: The Sync Layer

Elementary streams are the basic abstraction for any streaming data source. Elementary streams are conveyed as sync layer-packetized (SL-packetized) streams at the DMIF Application Interface. This packetized representation additionally provides timing and synchronization information, as well as fragmentation and random access information. The sync layer (SL) extracts this timing information to enable synchronized decoding and, subsequently, composition of the elementary stream data.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000010

0.6 The Compression Layer

The compression layer receives data in its encoded format and performs the necessary operations to decode this data. The decoded information is then used by the terminal's composition, rendering and presentation subsystems.

0.6.1 Object Description Framework

The purpose of the object description framework 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. Object content information and the interface to intellectual property management and protection systems are also part of this framework.

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 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 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).

0.6.1.1 Intellectual Property Management and Protection

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 are carried as part of an object descriptor stream. IPMP elementary streams carry time variant IPMP information that can be associated to multiple object descriptors.

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.

0.6.1.2 Object Content Information

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. An OCI stream is organized in a sequence of small, synchronized entities called events that contain a set of OCI descriptors. OCI streams can be associated to multiple object descriptors.

0.6.2 Scene Description Streams

Scene description addresses the organization of audio-visual objects in a scene, in terms of both spatial and temporal attributes. This information allows the composition and rendering of individual audio-visual objects after the respective decoders have reconstructed the streaming data for them. For visual data, ISO/IEC 14496-1 does not mandate particular composition algorithms. Hence, visual composition is implementation dependent. For audio data, the composition process is defined in a normative manner in 9.2.2.13 and ISO/IEC 14496-3.

The scene description is represented using a parametric approach (BIFS - Binary Format for Scenes). The description consists of an encoded hierarchy (tree) of nodes with attributes and other information (including event sources and targets). Leaf nodes in this tree correspond to elementary audio-visual data, whereas intermediate nodes group this material to form audio-visual objects, and perform grouping, transformation, and other such

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000011

operations on audio-visual objects (scene description nodes). The scene description can evolve over time by using scene description updates.

In order to facilitate active user involvement with the presented audio-visual information, ISO/IEC 14496-1 provides support for user and object interactions. Interactivity mechanisms are integrated with the scene description information, in the form of linked event sources and targets (routes) as well as sensors (special nodes that can trigger events based on specific conditions). These event sources and targets are part of scene description nodes, and thus allow close coupling of dynamic and interactive behavior with the specific scene at hand. ISO/IEC 14496-1, however, does not specify a particular user interface or a mechanism that maps user actions (e.g., keyboard key presses or mouse movements) to such events.

Such an interactive environment may not need an upstream channel, but ISO/IEC 14496 also provides means for client-server interactive sessions with the ability to set up upstream elementary streams and associate them to specific downstream elementary streams.

0.6.3 Audio-visual Streams

The coded representations of audio and visual information are described in ISO/IEC 14496-3 and ISO/IEC 14496-2, respectively. The reconstructed audio-visual data are made available to the composition process for potential use during the scene rendering.

0.6.4 Upchannel Streams

Downchannel elementary streams may require upchannel information to be transmitted from the receiving terminal to the sending terminal (e.g., to allow for client-server interactivity). Figure 1 indicates the flowpath for an elementary stream from the receiving terminal to the sending terminal. The content of upchannel streams is specified in the same part of the specification that defines the content of the downstream data. For example, upchannel control streams for video downchannel elementary streams are defined in ISO/IEC 14496-2.

0.7 Application Engine

The MPEG-J is a programmatic system (as opposed to a conventional parametric system) which specifies API for interoperation of MPEG-4 media players with Java code. By combining MPEG-4 media and safe executable code, content creators may embed complex control and data processing mechanisms with their media data to intelligently manage the operation of the audio-visual session. The parametric MPEG-4 System forms the Presentation Engine while the MPEG-J subsystem controlling the Presentation Engine forms the Application Engine.

The Java application is delivered as a separate elementary stream to the MPEG-4 terminal. There it will be directed to the MPEG-J run time environment, from where the MPEG-J program will have access to the various components and required data of the MPEG-4 player to control it.

In addition to the basic packages of the language (java.lang, java.io, java.util) a few categories of APIs have been defined for different scopes. For Scene graph API the objective is to provide access to the scene graph: to inspect the graph, to alter nodes and their fields, and to add and remove nodes within the graph. The Resource API is used for regulation of performance: it provides a centralized facility for managing resources. This is used when the program execution is contingent upon the terminal configuration and its capabilities, both static (that do not change during execution) and dynamic. Decoder API allows the control of the decoders that are present in the terminal. The Net API provides a way to interact with the network, being compliant to the MPEG-4 DMIF Application Interface. Complex applications and enhanced interactivity are possible with these basic packages. The architecture of MPEG-J will be presented in more detail in clause 11.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000012

Information technology — Coding of audio-visual objects —

Part 1: Systems

1 Scope

This part of ISO/IEC 14496 specifies system level functionalities for the communication of interactive audio-visual scenes. More specifically:

1. system level description of the coded representation of natural or synthetic, two-dimensional (2D) or three-dimensional (3D) objects that can be manifested audibly and/or visually (audio-visual objects);

2. the coded representation of the spatio-temporal positioning of audio-visual objects as well as their behavior in response to interaction (scene description); and

3. the coded representation of information related to the management of data streams (synchronization, identification, description and association of stream content).

4. a system level description of an application engine (format, delivery, lifecycle, and behavior of downloadable Java byte code applications); and

5. a system level interchange and storage format of interactive audio-visual scenes.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 14496. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO/IEC 14496 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

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 10646-1:2000, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane*

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 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000013

ISO/IEC 13818-7:1997, *Information technology — Generic coding of moving pictures and associated audio information — Part 7: Advanced Audio Coding (AAC)*

ISO/IEC 14496-2:1999, *Information technology — Coding of audio-visual objects — Part 2: Visual*

ISO/IEC 14772-1:1998, *Information technology — Computer graphics and image processing — The Virtual Reality Modeling Language — Part 1: Functional specification and UTF-8 encoding*

ISO/IEC 14772-1:1998/Amd.1, *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:—[1)], *Information technology — ECMAScript language specification*

ITU-T Rec. H.262 (2000) | ISO/IEC 13818-2:2000, *Information technology — Generic coding of moving pictures and associated audio information: Video*

ITU-T Rec. T.81 (1992) | ISO/IEC 10918-1:1994, *Information technology — Digital compression and coding of continuous-tone still images: Requirements and guidelines*

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, Volume1: 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 part of ISO/IEC 14496, the following terms and definitions apply.

[1)] To be published. (Revision of ISO/IEC 16262:1998)

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000014

4.1 Access Unit (AU)
An individually accessible portion of data within an *elementary stream*. An access unit is the smallest data entity to which timing information can be attributed

4.2 Alpha Map
The representation of the transparency parameters associated with a texture map.

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

4.4 Audio-visual Object
A representation of a natural or synthetic object that has an audio and/or visual manifestation. 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)
A 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)
A coded representation of a parametric scene description format.

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

4.8 Byte Aligned
A 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
A contiguous set of samples stored for one stream.

4.10 Clock Reference
A special time stamp that conveys a reading of a time base.

4.11 Composition
The 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)
A random access memory that contains composition units.

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

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

4.15 Compression Layer
The 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. It incorporates the decoders.

4.16 Container Atom
An atom whose sole purpose is to contain and group a set of related atoms.

4.17 Decoder
An entity that translates between the coded representation of an elementary stream and its decoded representation.

4.18 Decoding buffer (DB)
A buffer at the input of a decoder that contains access units.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000015

4.19 Decoder configuration
The 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)
An indication of the nominal decoding time of an access unit.

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

4.22 Descriptor
A 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)
An interface specified in ISO/IEC 14496-6. 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.

4.24 Elementary Stream (ES)
A 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
A 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)
A conceptual interface modeling the exchange of elementary stream data and associated control information between the compression layer and the sync layer.

4.27 FlexMux Channel (FMC)
A label to differentiate between data belonging to different constituent streams within one FlexMux Stream. A sequence of data in one FlexMux channel within a FlexMux stream corresponds to one single SL-packetized stream.

4.28 FlexMux Packet
The smallest data entity managed by the FlexMux tool. It consists of a header and a payload.

4.29 FlexMux Stream
A sequence of FlexMux Packets with data from one or more SL-packetized streams that are each identified by their own FlexMux channel.

4.30 FlexMux tool
A tool that allows the interleaving of data from multiple data streams.

4.31 Graphics Profile
A profile that specifies the permissible set of graphical elements of the BIFS tool that may be used in a scene description stream. Note that BIFS comprises both graphical and scene description elements.

4.32 Hint Track
A special track which contains instructions for packaging one or more tracks into a TransMux. It does not contain media data (an elementary stream).

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

4.34 Inter
A mode for coding parameters that uses previously coded parameters to construct a prediction.

4.35 Intra
A mode for coding parameters that does not make reference to previously coded parameters to perform the encoding.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000016

4.36 Initial Object Descriptor
A special object descriptor that allows the receiving terminal to gain initial access to portions of content encoded according to ISO/IEC 14496. It conveys profile and level information to describe the complexity of the content.

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

4.38 Intellectual Property Management and Protection (IPMP) System
A generic term for mechanisms and tools to manage and protect intellectual property. Only the interface to such systems is normatively defined.

4.39 Movie Atom
A container atom whose sub-atoms define the meta-data for a presentation ('moov').

4.40 Movie Data Atom
A container atom which can hold the actual media data for a presentation ('mdat').

4.41 MP4 File
The name of the file format described in this specification.

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

4.43 Object Content Information (OCI)
Additional information about content conveyed through one or more elementary streams. It is either aggregated to individual elementary stream descriptors or is itself conveyed as an elementary stream.

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

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

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

4.47 Object Descriptor Stream
An elementary stream that conveys object descriptors encapsulated in object descriptor commands.

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

4.49 Parametric Audio Decoder
A 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.

4.50 Quality of Service (QoS)
The performance that an elementary stream requests from the delivery channel through which it is transported. QoS is characterized by a set of parameters (e.g., bit rate, delay jitter, bit error rate, etc.).

4.51 Random Access
The process of beginning to read and decode a coded representation at an arbitrary point within the elementary stream.

4.52 Reference Point
A location in the data or control flow of a system that has some defined characteristics.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000017

4.53 Rendering
The 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).

4.54 Rendering Area
The portion of the display device's screen into which the scene description and its constituent audio-visual objects are to be rendered.

4.55 Sample
An access unit for an elementary stream. In hint tracks, a sample defines the formation of one or more TransMux packets.

4.56 Sample Table
A packed directory for the timing and physical layout of the samples in a track.

4.57 Scene Description
Information that describes the spatio-temporal positioning of audio-visual objects as well as their behavior resulting from object and user interactions. The scene description makes reference to elementary streams with audio-visual data by means of pointers to object descriptors.

4.58 Scene Description Stream
An elementary stream that conveys scene description information.

4.59 Scene Graph Elements
The 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 clause 15.

4.60 Scene Graph Profile
A profile that defines the permissible set of scene graph elements of the BIFS tool that may be used in a scene description stream. Note that BIFS comprises both graphical and scene description elements.

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

4.62 Structured Audio
A method of describing synthetic sound effects and music as defined by ISO/IEC 14496-3.

4.63 Sync Layer (SL)
A 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. The sync layer syntax is configurable and can be configured to be empty.

4.64 Sync Layer Configuration
A configuration of the sync layer syntax for a particular elementary stream using information contained in its elementary stream descriptor.

4.65 Sync Layer Packet (SL-Packet)
The smallest data entity managed by the sync layer consisting of a configurable header and a payload. The payload may consist of one complete access unit or a partial access unit.

4.66 Syntactic Description Language (SDL)
A language defined by ISO/IEC 14496-1 that allows the description of a bitstream's syntax.

4.67 Systems Decoder Model (SDM)
A model that provides an abstract view of the behavior of a terminal compliant to ISO/IEC 14496. It consists of the buffer model and the timing model.

4.68 System Time Base (STB)
The time base of the terminal. Its resolution is implementation-dependent. All operations in the terminal are performed according to this time base.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000018

4.69 Terminal
A system that sends, or receives and presents the coded representation of an interactive audio-visual scene as defined by ISO/IEC 14496-1. It can be a standalone system, or part of an application system complying with ISO/IEC 14496.

4.70 Time Base
The notion of a clock; it is equivalent to a counter that is periodically incremented.

4.71 Timing Model
A 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.

4.72 Time Stamp
An indication of a particular time instant relative to a time base.

4.73 Track
A collection of related samples in an MP4 file. For media data, a track corresponds to an elementary stream. For hint tracks, a track corresponds to a TransMuxchannel

5 Abbreviations and Symbols

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	FlexMux 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

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000019

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 clause 14.

7 Systems Decoder Model

7.1 Introduction

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

The systems decoder model specifies:

1. the interface for accessing demultiplexed data streams (DMIF Application Interface),

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

3. the behavior of elementary stream decoders,

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

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

These elements are depicted in Figure 2. Each elementary stream is attached to one single decoding buffer. More than one elementary stream may be connected to a single decoder (e.g., in a decoder of a scaleable audio-visual object).

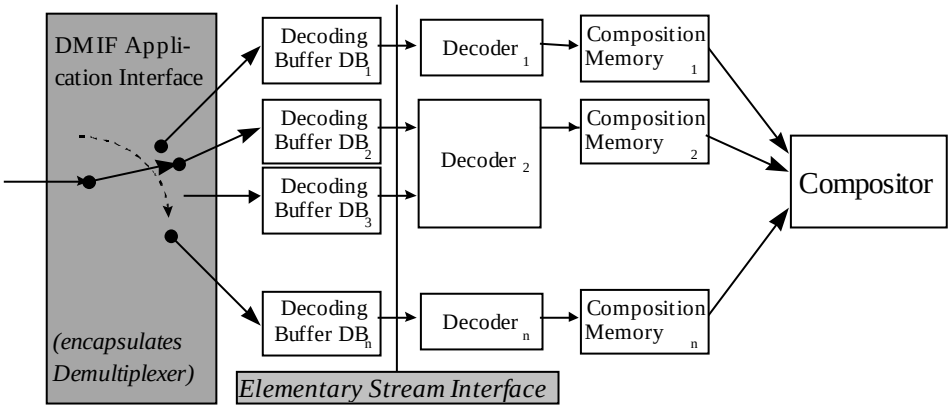


Figure 2 - Systems Decoder Model

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000020

7.2 Concepts of the systems decoder model

This subclause defines the concepts necessary for the specification of the timing and buffering model. The sequence of definitions corresponds to a walk from the left to the right side of the SDM illustration in Figure 2.

7.2.1 DMIF Application Interface (DAI)

For the purposes of the systems decoder model, the DMIF Application Interface encapsulates the demultiplexer and provides access to streaming data that is consumed by the decoding buffers. The streaming data received through the DAI consists of SL-packetized streams. The required properties of the DAI are described in 10.3. The DAI semantics are fully specified in ISO/IEC 14496-6.

7.2.2 SL-Packetized Stream (SPS)

An SL-packetized stream consists of a sequence of packets, according to the syntax and semantics specified in 10.2, that encapsulate a single elementary stream. The packets contain elementary stream data partitioned in access units as well as side information, e.g., for timing and access unit labeling. SPS data payload enters the decoding buffers, i.e., the side information is removed at the input to the decoding buffers.

7.2.3 Access Units (AU)

Elementary stream data is partitioned into access units. The delineation of an access unit is completely determined by the entity that generates the elementary stream (e.g., the compression layer). An access unit is the smallest data entity to which timing information can be attributed. Two access units from the same elementary stream shall never refer to the same decoding or composition time. Any further partitioning of the data in an elementary stream is not visible for the purposes of the systems decoder model. Access units are conveyed by SL-packetized streams and are received by the decoding buffers. The decoders consume access units with the necessary side information (e.g., time stamps) from the decoding buffers.

NOTE — An ISO/IEC 14496-1 compliant terminal implementation is not required to process each incoming access unit as a whole. It is furthermore possible to split an access unit into several fragments for transmission as specified in clause 10. This allows the sending terminal to dispatch partial AUs immediately as they are generated during the encoding process. Such partial AUs may have significance for improved error resilience.

7.2.4 Decoding Buffer (DB)

The decoding buffer is a buffer at the input of an elementary stream decoder in the receiving terminal that receives and stores access units. The systems buffer model enables the sending terminal to monitor the decoding buffer resources that are used during a presentation.

7.2.5 Elementary Streams (ES)

Streaming data received at the output of a decoding buffer, independent of its content, is considered as an elementary stream for the purpose of ISO/IEC 14496. The elementary streams are produced and consumed by the compression layer entities (encoders and decoders, respectively). ISO/IEC 14496 assumes that the integrity of an elementary stream is preserved from end to end.

7.2.6 Elementary Stream Interface (ESI)

The elementary stream interface is a concept that models the exchange of elementary stream data and associated control information between the compression layer and the sync layer. It is explained further in Annex L.

7.2.7 Decoder

For the purposes of this model, the decoder extracts access units from the decoding buffer at precisely defined points in time and places composition units, the results of the decoding processes, in the composition memory. A decoder may be attached to several decoding buffers.

7.2.8 Composition Units (CU)

Decoders consume access units and produce composition units. An access unit corresponds to an integer number of composition units. Composition units reside in composition memory.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000021

7.2.9 Composition Memory (CM)

The composition memory is a random access memory that contains composition units. The size of this memory is not normatively specified.

7.2.10 Compositor

The compositor takes composition units out of the composition memory and either consumes them (e.g. composes and presents them, in the case of audio-visual data) or skips them. The compositor is not specified in ISO/IEC 14496-1, as the details of this operation are not relevant within the context of the systems decoder model. Subclause 7.3.5 defines which composition units are available to the compositor at any instant of time.

7.3 Timing Model Specification

The timing model relies on clock references and time stamps to synchronize audio-visual data conveyed by one or more elementary streams. The concept of a clock with its associated clock references is used to convey the notion of time to a receiving terminal. Time stamps are used to indicate the precise time instants at which the receiving terminal consumes the access units in the decoding buffers or may access the composition units resident in the composition memory. The time stamps are therefore associated with access units and composition units. The semantics of the timing model are defined in the subsequent clauses. The syntax for conveying timing information is specified in 10.2.

NOTE — This timing model is designed for rate-controlled (“push”) applications.

7.3.1 System Time Base (STB)

The system time base (STB) defines the terminal’s notion of time. The resolution of the STB is implementation dependent. All actions of the terminal are scheduled according to this time base for the purpose of this timing model.

NOTE — This does not imply that all terminals compliant with ISO/IEC 14496 operate on one single STB.

7.3.2 Object Time Base (OTB)

The object time base (OTB) defines the notion of time for a given data stream. The resolution of this OTB can be selected as required by the application or as defined by a profile. All time stamps that the sending terminal inserts in a coded data stream refer to this time base. The OTB of a data stream is known at the receiving terminal either by means of object clock reference information inserted in the stream or by an indication that its time base is slaved to a time base conveyed with another stream, as specified in 10.2.3.

NOTE 1 — Elementary streams may be created for the sole purpose of conveying time base information.

NOTE 2 — The receiving terminal’s system time base need not be locked to any of the available object time bases.

7.3.3 Object Clock Reference (OCR)

A special kind of time stamps, object clock references (OCR), are used to convey the OTB to the elementary stream decoder. The value of the OCR corresponds to the value of the OTB at the time the sending terminal generates the object clock reference time stamp. OCR time stamps are placed in the SL packet header as described in 10.2.4. The receiving terminal shall evaluate the OCR when its last bit is extracted at the input of the decoding buffer.

7.3.4 Decoding Time Stamp (DTS)

Each access unit has an associated nominal decoding time, the time at which it must be available in the decoding buffer for decoding. The AU is not guaranteed to be available in the decoding buffer either before or after this time. Decoding is assumed to occur instantaneously when the instant of time indicated by the DTS is reached.

This point in time can be implicitly specified if the (constant) temporal distance between successive access units is indicated in the setup of the elementary stream (see 10.2.3). Otherwise a decoding time stamp (DTS) whose syntax is defined in 10.2.4 conveys this point in time.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000022

A decoding time stamp shall only be conveyed for an access unit that carries a composition time stamp as well, and only if the DTS and CTS values are different. Presence of both time stamps in an AU may indicate a reversal between coding order and composition order.

7.3.5 Composition Time Stamp (CTS)

Each composition unit has an associated nominal composition time, the time at which it must be available in the composition memory for composition. The CU is not guaranteed to be available in the composition memory *for composition* before this time. Since the SDM assumes an instantaneous decoding process, the CU is available to the *decoder*, at that instant in time corresponding to the DTS of the corresponding AU, for further use (e.g. in prediction processes).

This instant in time is implicitly known, if the (constant) temporal distance between successive composition units is indicated in the setup of the elementary stream. Otherwise a composition time stamp (CTS) whose syntax is defined in 10.2.4 conveys this instant in time.

The current CU is instantaneously accessible by the compositor anytime between its composition time and the composition time of the subsequent CU. If a subsequent CU does not exist, the current CU becomes unavailable at the end of the lifetime of its elementary stream (i.e., when its elementary stream descriptor is removed).

7.3.6 Occurrence and Precision of Timing Information in Elementary Streams

The frequency at which DTS, CTS and OCR values are to be inserted in the bitstream as well as the precision, jitter and drift are application and profile dependent. Some usage considerations can be found in 10.2.7.

7.3.7 Time Stamps for Dependent Elementary Streams

An audio-visual object may refer to multiple elementary streams that constitute a scaleable content representation (see 8.7.1.5). Such a set of elementary streams shall adhere to a single object time base. Temporally co-located access units for such elementary streams are then identified by identical DTS or CTS values.

EXAMPLE

The example in Figure 3 illustrates the arrival of two access units at the Systems Decoder. Due to the constant delay assumption of the model (see 7.4.2 below), the arrival times correspond to the instants in time when the sending terminal has sent the respective AUs. The sending terminal must select this instant in time so that the Decoding Buffer at the receiving terminal never overflows or underflows. At the receiving terminal, an AU is instantaneously decoded, at that instant in time corresponding to its DTS, and the resulting CU(s) are placed in the composition memory and remain there until the subsequent CU(s) arrive or the associated object descriptor is removed.

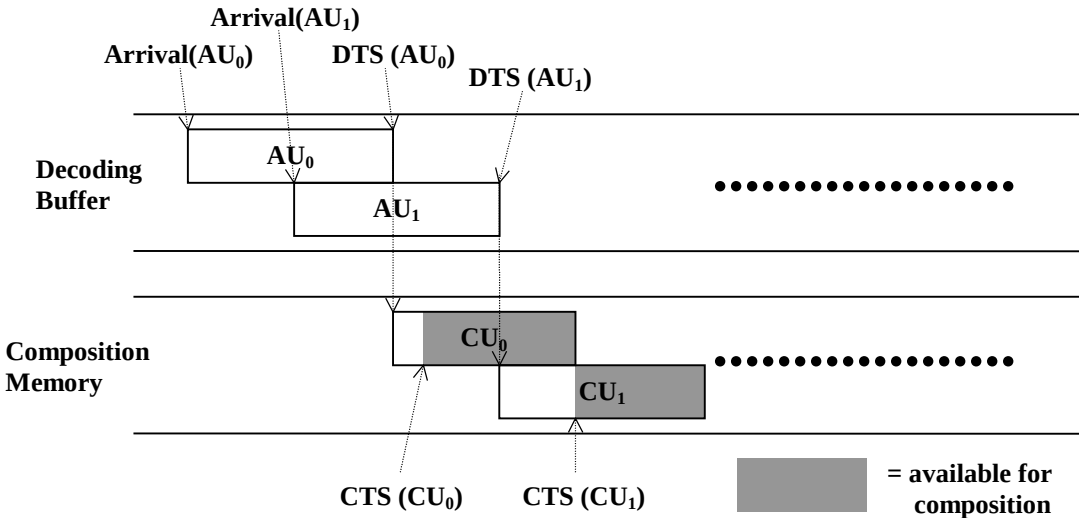


Figure 3 - Composition unit availability

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000023

7.4 Buffer Model Specification

7.4.1 Elementary Decoder Model

Figure 4 indicates one branch of the systems decoder model (Figure 2). This simplified model is used to specify the buffer model. It treats each elementary stream separately and therefore, associates a composition memory with only one decoder. The legend following Figure 4 elaborates on the symbols used in this figure.

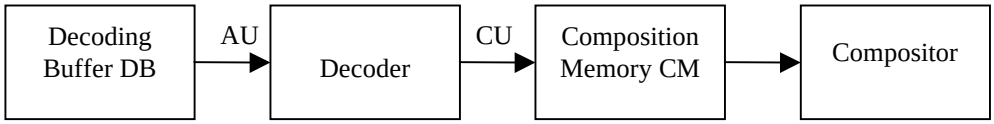


Figure 4 - Flow diagram for the systems decoder model

Legend:

DB Decoding buffer for the elementary stream.
CM Composition memory for the elementary stream.
AU The current access unit input to the decoder.
CU The current composition unit input to the composition memory. CU results from decoding AU. There may be several composition units resulting from decoding one access unit.

7.4.2 Assumptions

7.4.2.1 Constant end-to-end delay

Data transmitted in real time have a timing model in which the end-to-end delay from the encoder input at the sending terminal, to the decoder output at the receiving terminal, is constant. This delay is equal to the sum of the delay due to the encoding process, subsequent buffering, multiplexing at the sending terminal, the delay due to the delivery layers and the delay due to the demultiplexing, decoder buffering and decoding processes at the receiving terminal.

Note that the receiving terminal is free to add a temporal offset (delay) to the absolute values of all time stamps if it can cope with the additional buffering needed. However, the temporal difference between two time stamps (that determines the temporal distance between the associated AUs or CUs) has to be preserved for real-time performance.

NOTE — Two elementary streams that adhere to different time bases may be synchronized tightly in case of constant end-to-end delay as assumed by this model. If an application cannot implement this model assumption, such tight synchronization may not be achievable. Tolerances for the constant end-to-end delay assumption need to be defined through the profile and level mechanism.

7.4.2.2 Demultiplexer

The end-to-end delay between multiplexer output, at the sending terminal, and demultiplexer input, at the receiving terminal, is constant.

7.4.2.3 Decoding Buffer

The needed decoding buffer size is known by the sending terminal and conveyed to the receiving terminal as specified in 8.6.6.

The size of the decoding buffer is measured in bytes.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000024

The decoding buffer is filled at the rate given by the maximum bit rate for this elementary stream while data is available and with a zero rate otherwise. The maximum bit rate is conveyed by the sending terminal as a part of the decoder configuration information during the set up phase for each elementary stream (see 8.6.6).

Information is received from the DAI in the form of SL packets. The SL packet headers are removed at the input to the decoding buffers.

7.4.2.4 Decoder

The decoding processes are assumed to be instantaneous for the purposes of the systems decoder model.

7.4.2.5 Composition Memory

The mapping of an AU to one or more CUs (by the decoder) is known implicitly at both the sending and the receiving terminals.

7.4.2.6 Compositor

The composition processes are assumed to be instantaneous for the purposes of the systems decoder model.

7.4.3 Managing Buffers: A Walkthrough

In this example, we assume that the model is used in a “push” scenario. In applications where non-real time content is to be delivered, flow control by suitable signaling may be established to request access units at the time they are needed at the receiving terminal. The mechanisms for doing so are application-dependent, and are not specified in ISO/IEC 14496.

The behaviors of the various elements in the SDM are modeled as follows:

- The sending terminal signals the required decoding buffer resources to the receiving terminal before starting the delivery. This is done as specified in 8.6.6 either explicitly, by requesting the decoding buffer sizes for individual elementary streams, or implicitly, by indicating a profile (see clause 15). The decoding buffer size is measured in bytes.

- The sending terminal models the behavior of the decoding buffers by making the following assumptions :

- Each decoding buffer is filled at the maximum bitrate specified for its associated elementary stream as long as data is available.

- At the instant of time corresponding to its DTS, an AU is instantaneously decoded and removed from the decoding buffer.

- At the instant of time corresponding to its DTS, a known amount of CUs corresponding to the just decoded AU are put in the composition memory.

The current CU is available to the compositor between instants of time corresponding to the CTS of the current CU and the CTS of the subsequent CU. If a subsequent CU does not exist, the current CU becomes unavailable at the end of lifetime of its data stream.

Using these assumptions on the buffer model, the sending terminal may freely use the space in the decoding buffers. For example, it may deliver data for several AUs of a stream, for non real time usage, to the receiving terminal, and pre-store them in the DB long before they have to be decoded (assuming sufficient space is available). Subsequently, the full delivery bandwidth may be used to transfer data of a real time stream just in time. The composition memory may be used, for example, as a reordering buffer. In the case of visual decoding, it may contain the decoded P-frames needed by a video decoder for the decoding of intermediate B-frames, before the arrival of the CTS of the latest P-frame.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000025

8 Object Description Framework

8.1 Introduction

The scene description (specified in clause 9) and the elementary streams that convey streaming data are the basic building blocks of the architecture of ISO/IEC 14496-1. Elementary streams carry data for audio or visual objects as well as for the scene description itself. The object description framework provides the link between elementary streams and the scene description. The scene description declares the spatio-temporal relationship of audio-visual objects, while the object description framework specifies the elementary stream resources that provide the time-varying data for the scene. This indirection facilitates independent changes to the scene structure, the properties of the elementary streams (e.g. its encoding) and their delivery.

The object description framework consists of a set of descriptors that allows to identify, describe and properly associate elementary streams to each other and to audio-visual objects used in the scene description. Numeric identifiers, called ObjectDescriptorIDs, associate object descriptors to appropriate nodes in the scene description. Object descriptors are themselves conveyed in elementary streams to allow time stamped changes to the available set of object descriptors to be made.

Each object descriptor is itself a collection of descriptors that describe one or more elementary streams that are associated to a single node and that usually relate to a single audio or visual object. This allows to indicate a scaleable content representation as well as multiple alternative streams that convey the same content, e.g., in multiple qualities or different languages.

An elementary stream descriptor within an object descriptor identifies a single elementary stream with a numeric identifier, called ES_ID. Each elementary stream descriptor contains the information necessary to initiate and configure the decoding process for the elementary stream, as well as intellectual property identification. Optionally, additional information may be associated to a single elementary stream, most notably quality of service requirements for its transmission or a language indication. Both, object descriptors and elementary stream descriptors may use URLs to point to remote object descriptors or a remote elementary stream source, respectively.

The object description framework provides the hooks to implement intellectual property management and protection (IPMP) systems. IPMP information is conveyed both through IPMP descriptors as part of the object descriptor stream and through IPMP streams that carry time variant IPMP information. The structure of IPMP descriptors and IPMP streams is specified in this clause while their internal syntax and semantics and, hence, the operation of the IPMP system is outside the scope of ISO/IEC 14496.

Object content information allows the association of metadata with a whole presentation or with individual object descriptors or with elementary stream descriptors. A set of OCI descriptors is defined that either form an integral part of an object descriptor or elementary stream descriptor or are conveyed by means of a proper OCI stream that allows the conveyance of time variant object content information.

Access to ISO/IEC 14496 content is gained through an initial object descriptor that needs to be made available through means not defined in ISO/IEC 14496. The initial object descriptor in the simplest case points to the scene description stream and the corresponding object descriptor stream. The access scenario is outlined in 8.7.3.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000026

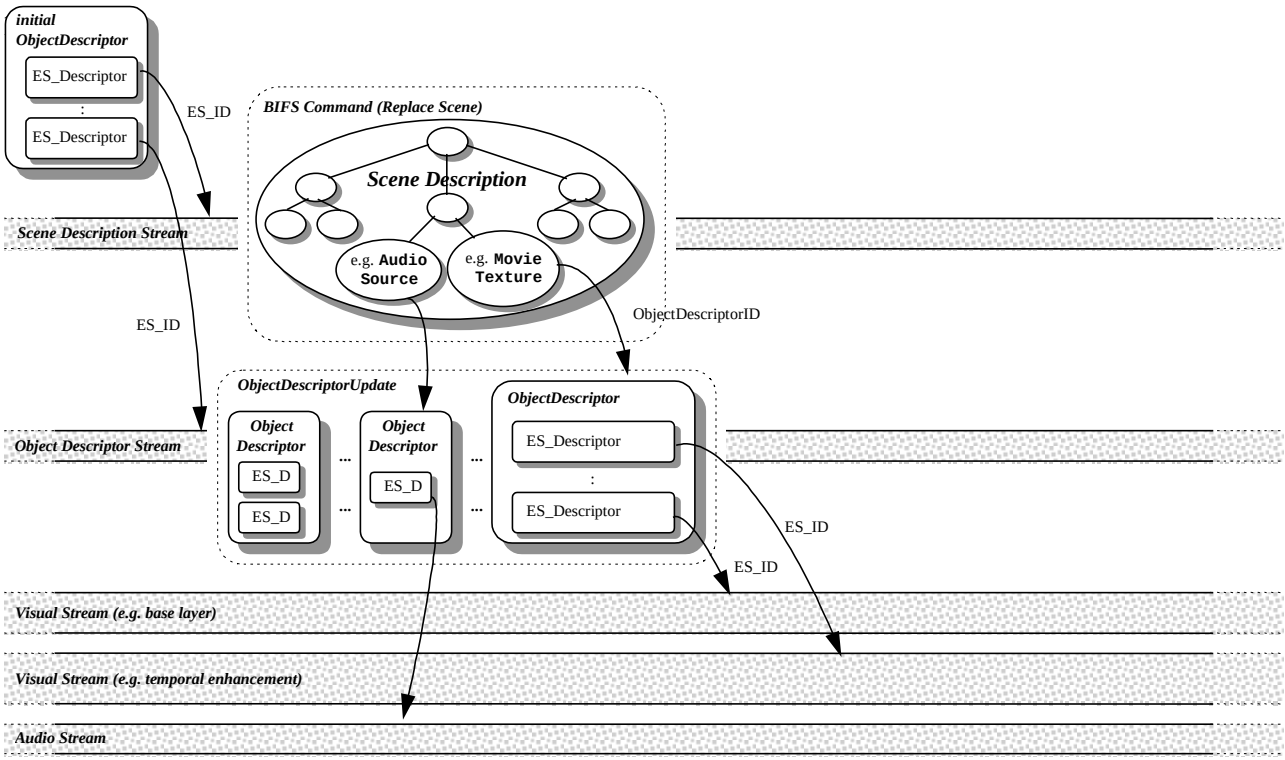


Figure 5 - Object descriptors linking scene description to elementary streams

The remainder of this clause is structured in the following way:

- Subclause 8.2 specifies the data structures on which the object descriptor framework is based.
- Subclause 8.3 specifies the concepts of the IPMP elements in the object description framework.
- Subclause 8.4 specifies the object content information elements in the object description framework.
- Subclause 8.5 specifies the object descriptor stream and the syntax and semantics of the command set that allows the update or removal of object descriptor components.
- Subclause 8.6 specifies the syntax and semantics of the object descriptor and its component descriptors.
- Subclause 8.7 specifies rules for object descriptor usage as well as the procedure to access content through object descriptors.
- Subclause 8.8 specifies the usage of the IPMP system interface.

8.2 Common data structures

8.2.1 Overview

The commands and descriptors defined in this subclause constitute self-describing classes, identified by unique class tags. Each class encodes explicitly its size in bytes. This facilitates future compatible extensions of the commands and descriptors. A class may be expanded with additional syntax elements that are ignored by an OD decoder that expects an earlier revision of a class. In addition, anywhere in a syntax where a set of tagged classes is expected it is permissible to intersperse expandable classes with unknown class tag values. These classes shall be skipped, using the encoded size information.

The remainder of this clause defines the syntax and semantics of the command and descriptor classes. Some commands and descriptors contain themselves a set of component descriptors. They are said to *aggregate a set of component descriptors.*

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000027

Table 1 - List of Class Tags for Descriptors

Tag value	Tag name
0x00	Forbidden
0x01	ObjectDescrTag
0x02	InitialObjectDescrTag
0x03	ES_DescrTag
0x04	DecoderConfigDescrTag
0x05	DecSpecificInfoTag
0x06	SLConfigDescrTag
0x07	ContentIdentDescrTag
0x08	SupplContentIdentDescrTag
0x09	IPI_DescrPointerTag
0x0A	IPMP_DescrPointerTag
0x0B	IPMP_DescrTag
0x0C	QoS_DescrTag
0x0D	RegistrationDescrTag
0x0E	ES_ID_IncTag
0x0F	ES_ID_RefTag
0x10	MP4_IOD_Tag
0x11	MP4_OD_Tag
0x12	IPL_DescrPointerRefTag
0x13	ExtendedProfileLevelDescrTag
0x14	profileLevelIndicationIndexDescrTag
0x15-0x3F	Reserved for ISO use
0x40	ContentClassificationDescrTag
0x41	KeyWordDescrTag
0x42	RatingDescrTag
0x43	LanguageDescrTag
0x44	ShortTextualDescrTag
0x45	ExpandedTextualDescrTag
0x46	ContentCreatorNameDescrTag
0x47	ContentCreationDateDescrTag
0x48	OCICreatorNameDescrTag
0x49	OCICreationDateDescrTag
0x4A	SmpteCameraPositionDescrTag
0x4B-0x5F	Reserved for ISO use (OCI extensions)
0x60-0xBF	Reserved for ISO use
0xC0-0xFE	User private
0xFF	Forbidden

8.2.2 BaseDescriptor

8.2.2.1 Syntax

```
abstract aligned(8) expandable(2^28-1) class BaseDescriptor : bit(8) tag=0 {
   // empty. To be filled by classes extending this class.
}
```

8.2.2.2 Semantics

This class is an abstract base class that is extended by the descriptor classes specified in 8.6. Each descriptor constitutes a self-describing class, identified by a unique class tag. This abstract base class establishes a common name space for the class tags of these descriptors. The values of the class tags are defined in Table 1. As an expandable class the size of each class instance in bytes is encoded and accessible through the instance variable sizeOfInstance (see 14.3.3).

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000028

A class that allows the aggregation of classes of type BaseDescriptor may actually aggregate any of the classes that extend BaseDescriptor.

NOTE — User private descriptors may have an internal structure, for example to identify the country or manufacturer that uses a specific descriptor. The tags and semantics for such user private descriptors may be managed by a registration authority if required.

The following additional symbolic names are introduced:

ExtDescrTagStartRange = 0x80
ExtDescrTagEndRange = 0xFE
OCIDescrTagStartRange = 0x40
OCIDescrTagEndRange = 0x5F

8.2.3 BaseCommand

8.2.3.1 Syntax

```
abstract aligned(8) expandable(2^28-1) class BaseCommand : bit(8) tag=0 {
   // empty. To be filled by classes extending this class.
}
```

8.2.3.2 Semantics

This class is an abstract base class that is extended by the command classes specified in 8.5.5. Each command constitutes a self-describing class, identified by a unique class tag. This abstract base class establishes a common name space for the class tags of these commands. The values of the class tags are defined in Table 2. As an expandable class the size of each class instance in bytes is encoded and accessible through the instance variable sizeOfInstance (see 14.3.3).

Table 2 - List of Class Tags for Commands

Tag value	Tag name
0x00	forbidden
0x01	ObjectDescrUpdateTag
0x02	ObjectDescrRemoveTag
0x03	ES_DescrUpdateTag
0x04	ES_DescrRemoveTag
0x05	IPMP_DescrUpdateTag
0x06	IPMP_DescrRemoveTag
0x07	ES_DescrRemoveRefTag
0x08-0xBF	Reserved for ISO (command tags)
0xC0-0xFE	User private
0xFF	forbidden

A class that allows the aggregation of classes of type BaseCommand may actually aggregate any of the classes that extend BaseCommand.

NOTE — User private commands may have an internal structure, for example to identify the country or manufacturer that uses a specific command. The tags and semantics for such user private command may be managed by a registration authority if required.

8.3 Intellectual Property Management and Protection (IPMP)

8.3.1 Overview

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. An IPMP System is a non-normative component that provides intellectual property management and protection functions for the terminal.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000029

The IPMP interface consists of IPMP elementary streams and IPMP descriptors. The normative structure of IPMP elementary streams is specified in this subclause. IPMP descriptors are carried as part of an object descriptor stream and are specified in 8.6.14. The IPMP interface allows applications (or derivative application standards) to build specialized IPMP Systems. Alternatively, an application may choose not to use an IPMP System, thereby offering no management and protection features. 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. The detailed semantics and decoding process of the IPMP System are not in the scope of ISO/IEC 14496. The usage of the IPMP System Interface, however, is explained in 8.8.

8.3.2 IPMP Streams

8.3.2.1 Structure of the IPMP Stream

The 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(s).

8.3.2.2 Access Unit Definition

An IPMP access unit consists of one or more IPMP messages, as defined in 8.3.2.5. All IPMP messages that are to be processed at the same instant in time shall constitute a single access unit. Access units in IPMP streams shall be labeled and time-stamped by suitable means. This shall be done via the related flags and the composition time stamps, respectively, in the SL packet header (see 10.2.4). The composition time indicates the point in time at which an IPMP access unit becomes valid, i.e., when the embedded IPMP messages shall be evaluated. Decoding and composition time for an IPMP access unit shall always have the same value.

An access unit does not necessarily convey or update the complete set of IPMP messages that are currently required. In that case it just modifies the persistent state of the IPMP system. However, if an access unit conveys the complete set of IPMP messages 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'.

NOTE — An SL packet with `randomAccessPointFlag=1` but with no IPMP messages in it indicates that at the current time instant no IPMP messages are required for operation.

8.3.2.3 Time Base for IPMP Streams

The time base associated to an IPMP 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 10.2.4) for this stream or by indicating the elementary stream from which this IPMP stream inherits the time base (see 10.2.3). All time stamps in the SL-packetized IPMP stream refer to this time base.

An IPMP stream shall adhere to the same time base as the one or more content elementary streams to which it is associated (see 8.8). Consequently, an IPMP stream may not be associated to multiple content elementary streams that themselves adhere to different time bases.

8.3.2.4 IPMP Decoder Configuration

8.3.2.4.1 Syntax

```
class IPMPDecoderConfiguration extends DecoderSpecificInfo : bit(8) tag=DecSpecificInfoTag {
   // IPMP system specific configuration information
}
```

8.3.2.4.2 Semantics

An IPMP system may require information to initialize its operation. This information shall be conveyed by extending the `decoderSpecificInfo` class as specified in 8.6.7. If utilized, `IPMPDecoderConfiguration` shall be conveyed in the `ES_Descriptor` declaring the IPMP stream.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000030

8.3.2.5 IPMP message syntax and semantics

8.3.2.5.1 Syntax

```
abstract aligned(8) expandable(2^28-1) class IPMP_Message
{
   bit(16)  IPMPS_Type;
   if (IPMPS_Type == 0) {
      bit(8) URLString[sizeOfInstance-2];
   } else {
      bit(8) IPMP_data[sizeOfInstance-2];
   }
}
```

8.3.2.5.2 Semantics

The `IPMP_Message` conveys control information for an IPMP System.

`IPMPS_Type` - the type of the IPMP System. A zero value does not correspond to an IPMP System, but shall indicate the presence of a URL. A non-zero value shall indicate a specific IPMP System Type. The values 0x0001-0x2000 are reserved for future ISO use. A Registration Authority, as designated by ISO, shall assign a unique valid value for this field for each specific IPMP System Type. The `IPMPS_Type` is used, for example, for distinguishing between IPMP systems from different companies.

`URLString[]` - contains a UTF-8 (ISO/IEC 10646-1) encoded URL that shall point to the location of a remote `IPMP_Message`. The `IPMPS_Type` of this `IPMP_Message` can be 0 or not. If 0, another URL is referenced. This process continues until an `IPMP_Message` with a non-zero `IPMPS_Type` is accessed.

`IPMP_data` - opaque data to control the IPMP System.

8.4 Object Content Information (OCI)

8.4.1 Overview

Audio-visual objects that are associated with elementary stream data through an object descriptor may have additional object content information attached to them. For this purpose, a set of OCI descriptors is defined in 8.6.18. OCI descriptors may directly be included as part of an object descriptor or `ES_Descriptor` as defined in 8.6.

In order to accommodate time variant OCI that is separable from the object descriptor stream, OCI descriptors may as well be conveyed in an OCI stream. An OCI stream is referred to through an ES_Descriptor, with the `streamType` field set to OCI_Stream. How OCI streams may be aggregated to object descriptors is defined in 8.7.1.3. The structure of the OCI stream is defined in this subclause.

8.4.2 OCI Streams

8.4.2.1 Structure of the OCI Stream

The OCI stream is an elementary stream that conveys time-varying object content information, termed OCI events. Each OCI event consists of a number of OCI descriptors.

8.4.2.2 Access Unit Definition

An OCI access unit consists of one or more OCI_Events, as described in 8.4.2.5. Access units in OCI elementary streams shall be labelled and time stamped by suitable means. This shall be done by means of the related flags and the composition time stamp, respectively, in the SL packet header (see 10.2.4). The composition time indicates the point in time when an OCI access unit becomes valid, i.e., when the embedded OCI events shall be added to the list of events. Decoding and composition time for an OCI access unit shall always have the same value.

An access unit may or may not convey or update the complete set of OCI events that are currently valid. In the latter case, it just modifies the persistent state of the OCI decoder. However, if an access unit conveys the complete set of OCI events valid 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'.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000031

NOTE — An SL packet with `randomAccessPointFlag=1` but with no OCI events in it indicates that at the current time instant no valid OCI events exist.

8.4.2.3 Time Base for OCI Streams

The time base associated with an OCI stream shall be indicated by suitable means. This shall be done by the use of object clock reference time stamps in the SL packet headers (see 10.2.4) for this stream or by indicating the elementary stream from which this OCI stream inherits the time base (see 10.2.3). All time stamps in the SL-packetized OCI stream refer to this time base.

8.4.2.4 OCI Decoder Configuration

8.4.2.4.1 Syntax

```
class OCIDecoderConfiguration extends DecoderSpecificInfo  : bit(8) tag=DecSpecificInfoTag {
   const bit(8) versionLabel = 0x01;
}
```

8.4.2.4.2 Semantics

This information is needed to initialize operation of the OCI decoder. It shall be conveyed by extending the `decoderSpecificInfo` class as specified in 8.6.7. `OCIDecoderConfiguration` shall be conveyed in the `ES_Descriptor` declaring the OCI stream.

`versionLabel` – indicates the version of OCI specification used on the corresponding OCI data stream. Only the value 0x01 is allowed; all the other values are reserved.

8.4.2.5 OCI_Events syntax and semantics

8.4.2.5.1 Syntax

```
abstract aligned(8) expandable(2^28-1) class OCI_Event {
   bit(15) eventID;
   bit(1)  absoluteTimeFlag;
   bit(32) startingTime;
   bit(32) duration;
   OCI_Descriptor OCI_Descr[1 .. 255];
}
```

8.4.2.5.2 Semantics

`eventID` – contains the identification number of the described event that is unique within the scope of this OCI stream.

`absoluteTimeFlag` – indicates the time base for `startingTime` as described below.

`startingTime` – indicates the starting time of the event in hours, minutes, seconds and hundredth of seconds. The format is 8 digits, the first 6 digits expressing hours, minutes and seconds with 4 bits each in binary coded decimal and the last two expressing hundredth of seconds in hexadecimal using 8 bits.

EXAMPLE — 02:36:45:89 is coded as “0x023645” concatenated with “0b0101.1001” (89 in binary), resulting to “0x02364559”.

If `absoluteTimeFlag` is set to zero, `startingTime` is relative to the object time base of the corresponding object. In that case it is the responsibility of the application to ensure that this object time base is conveyed such that `startingTime` can be identified unambiguously (see 10.2.7). If `absoluteTimeFlag` is set to one, `startingTime` is expressed as an absolute value, refering to wall clock time.

`duration` – contains the duration of the corresponding object in hours, minutes, seconds and hundredth of seconds. The format is 8 digits, the first 6 digits expressing hours, minutes and seconds with 4 bits each in binary coded decimal and the last two expressing hundredth of seconds in hexadecimal using 8 bits.

`OCI_Descr[]` – an array of one up to 255 `OCI_Descriptor` classes as specified in 8.6.18.2.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000032

8.5 Object Descriptor Stream

8.5.1 Structure of the Object Descriptor Stream

Similar to the scene description, object descriptors are transported in a dedicated elementary stream, termed object descriptor stream. Within such a stream, it is possible to dynamically convey, update and remove complete object descriptors, or their component descriptors, the ES_Descriptors, and IPMP descriptors. The update mechanism allows, for example, to advertise new elementary streams for an audio-visual object as they become available, or to remove references to streams that are no longer available. Updates are time stamped to indicate the instant in time they take effect.

This subclause specifies the structure of the object descriptor elementary stream including the syntax and semantics of its constituent elements, the object descriptor commands (OD commands).

8.5.2 Access Unit Definition

An OD access unit consists of one or more OD commands, as described in 8.5.5. All OD commands that are to be processed at the same instant in time shall constitute a single access unit. Access units in object descriptor elementary streams shall be labelled and time stamped by suitable means. This shall be done by means of the related flags and the composition time stamp, respectively, in the SL packet header (see 10.2.4). The composition time indicates the point in time when an OD access unit becomes valid, i.e., when the embedded OD commands shall be executed. Decoding and composition time for an OD access unit shall always have the same value.

An access unit may not convey or update the complete set of object descriptors that are currently required. In that case it just modifies the persistent state of the object descriptor decoder. However, if an access unit conveys the complete set of object descriptors 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'.

NOTE — An SL packet with `randomAccessPointFlag=1` but with no OD commands in it indicates that at the current time instant no valid object descriptors exist.

8.5.3 Time Base for Object Descriptor Streams

The time base associated to an object descriptor 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 10.2.4) for this stream or by indicating the elementary stream from which this object descriptor stream inherits the time base (see 10.2.3). All time stamps in the SL-packetized object descriptor stream refer to this time base.

8.5.4 OD Decoder Configuration

The object descriptor decoder does not require additional configuration information.

8.5.5 OD Command Syntax and Semantics

8.5.5.1 Overview

Object descriptors and their components as defined in 8.6 shall always be conveyed as part of one of the OD commands specified in this subclause. The commands describe the action to be taken on the components conveyed with the command, specifically 'update' or 'remove'. Each command affects one or more object descriptors, ES_Descriptors or IPMP descriptors.

8.5.5.2 ObjectDescriptorUpdate

8.5.5.2.1 Syntax

```
class ObjectDescriptorUpdate extends BaseCommand : bit(8) tag=ObjectDescrUpdateTag {
   ObjectDescriptorBase OD[1 .. 255];
}
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000033

8.5.5.2.2 Semantics

The `ObjectDescriptorUpdate` class conveys a list of new or updated object descriptors. If an object descriptor is updated, the streams refered to by the old object descriptor shall be closed and the streams refered to by the new object descriptor may be accessed by the content access procedure (see 8.7.3.6.2).

NOTE - The ES_DescriptorUpdate or ES_DescriptorRemove commands may be used to add or remove individual ES_Descriptors of an existing object descriptor.

`OD[]` – an array of object descriptors as defined in 8.6.3 and 8.6.4. The array shall have any number of one up to 255 elements.

8.5.5.3 ObjectDescriptorRemove

8.5.5.3.1 Syntax

```
class ObjectDescriptorRemove extends BaseCommand : bit(8) tag=ObjectDescrRemoveTag {
   bit(10) objectDescriptorId[(sizeOfInstance*8)/10];
}
```

8.5.5.3.2 Semantics

The `ObjectDescriptorRemove` class renders unavailable a set of object descriptors. The BIFS nodes associated to these object descriptors shall have no reference any more to the elementary streams that have been listed in the removed object descriptors. An objectDescriptorID that does not refer to a valid object descriptor is ignored.

NOTE — It is possible that a scene description node references an OD_ID which does not currently have an associated OD.

`ObjectDescriptorId[]` – an array of `ObjectDescriptorIDs` that indicates the object descriptors that are removed.

8.5.5.4 ES_DescriptorUpdate

8.5.5.4.1 Syntax

```
class ES_DescriptorUpdate extends BaseCommand : bit(8) tag=ES_DescrUpdateTag {
   bit(10) objectDescriptorId;
   ES_Descriptor esDescr[1 .. 255];
}
```

8.5.5.4.2 Semantics

The `ES_DescriptorUpdate` class conveys a list of new ES_Descriptors for the object descriptor labeled `objectDescriptorID`. ES_Descriptors with ES_IDs that have already been received within the same name scope shall be ignored.

To update the characterstics of an elementary stream, it is required that its original ES_Descriptor be removed and the changed ES_Descriptor be conveyed.

When an IPMP stream is added, the affected elementary streams, as defined in 8.8.2, shall be processed under the new IPMP conditions starting at the point in time that this ES_DescriptorUpdate command becomes valid (see 8.5.2).

`ES_DescriptorUpdate` shall not be applied on object descriptors that have set URL_Flag to '1' (see 8.6.3).

An elementary stream identified with a given ES_ID may be attached to more than one object descriptor. All corresponding `ES_Descriptors` refering to this ES_ID that are conveyed through either `ES_DescriptorUpdate` or `ObjectDescriptorUpdate` commands shall have identical content.

`objectDescriptorID` - identifies the object descriptor for which `ES_Descriptors` are updated. If the objectDescriptorID does not refer to any valid object descriptor, then this command is ignored.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000034

esDescr[] – an array of ES_Descriptors as defined in 8.6.5. The array shall have any number of one up to 255 elements.

8.5.5.5 ES_DescriptorRemove

8.5.5.5.1 Syntax

```
class ES_DescriptorRemove extends BaseCommand : bit(8) tag=ES_DescrRemoveTag {
   bit(10) objectDescriptorId;
   aligned (8) bit(16) ES_ID[1..255];
}
```

8.5.5.5.2 Semantics

The ES_DescriptorRemove class removes the reference to an elementary stream from an object descriptor and renders this stream unavailable for nodes referencing this object descriptor.

When an IPMP stream is removed, the affected elementary streams, as defined in 8.8.2, shall be processed under the new IPMP conditions starting at the point in time that this ES_DescriptorRemove command becomes valid (see 8.5.2).

ES_DescriptorRemove shall not be applied on object descriptors that have set URL_Flag to '1' (see 8.6.3).

objectDescriptorID - identifies the object descriptor from which ES_Descriptors are removed. If the objectDescriptorID does not refer to a valid object descriptor in the same scope, then this command is ignored.

ES_ID[] – an array of ES_IDs that labels the ES_Descriptors to be removed from objectDescriptorID. If any of the ES_IDs do not refer to an ES_Descriptor currently referenced by the OD, then those ES_IDs are ignored. The array shall have any number of one up to 255 elements.

8.5.5.6 IPMP_DescriptorUpdate

8.5.5.6.1 Syntax

```
class IPMP_DescriptorUpdate extends BaseCommand : bit(8) tag=IPMP_DescrUpdateTag {
   IPMP_Descriptor ipmpDescr[1..255];
}
```

8.5.5.6.2 Semantics

The IPMP_DescriptorUpdate class conveys a list of new or updated IPMP_Descriptors. An IPMP_Descriptor identified by an IPMP_DescriptorID that has already been received within the same name scope shall be replaced by the new descriptor.

Updates to an IPMP_Descriptor shall be propagated at the time this IPMP_DescriptorUpdate becomes valid (see 8.5.2) to all IPMP Systems that refer to this IPMP_Descriptor through an IPMP_DescriptorPointer (see 8.6.13). The handling of the descriptors by the IPMP systems is not normative.

IPMP_Descriptors remain valid until they are replaced by another IPMP_DescriptorUpdate command or removed.

ipmpDescr[] – an array of IPMP_Descriptor as specified in 8.6.14.

8.5.5.7 IPMP_DescriptorRemove

8.5.5.7.1 Syntax

```
class IPMP_DescriptorRemove extends BaseCommand : bit(8) tag=IPMP_DescrRemoveTag {
   bit(8) IPMP_DescriptorID[1..255];
}
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000035

8.5.5.7.2 Semantics

The IPMP_DescriptorRemove class conveys a list of IPMP_DescriptorsIDs that identify the IPMP_Descriptors that shall be removed.

The removal of IPMP_Descriptors shall be notified to all IPMP systems at the time this IPMP_DescriptorRemove becomes valid (see 8.5.2). The handling of the descriptors by the IPMP systems is not normative.

IPMP_DescriptorID[] - is a list of IPMP_DescriptorIDs.

8.6 Object Descriptor Components

8.6.1 Overview

Object descriptors contain various additional descriptors as their components, in order to describe individual elementary streams and their properties. They shall always be conveyed as part of one of the OD commands specified in the previous subclause. This subclause defines the syntax and semantics of object descriptors and their component descriptors.

8.6.2 ObjectDescriptorBase

8.6.2.1 Syntax

```
abstract class ObjectDescriptorBase extends BaseDescriptor : bit(8)
tag=[ObjectDescrTag..InitialObjectDescrTag] {
// empty. To be filled by classes extending this class.
}
```

8.6.2.2 Semantics

This is an abstract base class for the different types of object descriptor classes defined subsequently. The term “object descriptor” is used to generically refer to any such derived object descriptor class or instance thereof.

8.6.3 ObjectDescriptor

8.6.3.1 Syntax

```
class ObjectDescriptor extends ObjectDescriptorBase : bit(8) tag=ObjectDescrTag {
   bit(10) ObjectDescriptorID;
   bit(1) URL_Flag;
   const bit(5) reserved=0b1111.1;
   if (URL_Flag) {
      bit(8) URLlength;
      bit(8) URLstring[URLlength];
   } else {
      ES_Descriptor esDescr[1 .. 255];
      OCI_Descriptor ociDescr[0 .. 255];
      IPMP_DescriptorPointer ipmpDescrPtr[0 .. 255];
   }
   ExtensionDescriptor extDescr[0 .. 255];
}
```

8.6.3.2 Semantics

The ObjectDescriptor consists of three different parts.

The first part uniquely labels the object descriptor within its name scope (see 8.7.2.4) by means of an objectDescriptorId. Nodes in the scene description use objectDescriptorID to refer to the related object descriptor. An optional URLstring indicates that the actual object descriptor resides at a remote location.

The second part consists of a list of ES_Descriptors, each providing parameters for a single elementary as well as an optional set of object content information descriptors and pointers to IPMP descriptors for the contents for elementary stream content described in this object descriptor.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000036

The third part is a set of optional descriptors that support the inclusion of future extensions as well as the transport of private data in a backward compatible way.

`objectDescriptorId` – This syntax element uniquely identifies the `ObjectDescriptor` within its name scope. The value 0 is forbidden and the value 1023 is reserved.

`URL_Flag` – a flag that indicates the presence of a `URLstring`.

`URLlength` – the length of the subsequent `URLstring` in bytes.

`URLstring[]` – A string with a UTF-8 (ISO/IEC 10646-1) encoded URL that shall point to another `ObjectDescriptor`. Only the content of this object descriptor shall be returned by the delivery entity upon access to this URL. Within the current name scope, the new object descriptor shall be referenced by the `objectDescriptorId` of the object descriptor carrying the URLstring. On name scopes see 8.7.2.4. Permissible URLs may be constrained by profile and levels as well as by specific delivery layers.

`esDescr[]` – an array of `ES_Descriptors` as defined in 8.6.5. The array shall have any number of one up to 255 elements.

`ociDescr[]` – an array of `OCI_Descriptors`, as defined in 8.6.18.2, that relates to the audio-visual object(s) described by this object descriptor. The array shall have any number of zero up to 255 elements.

`ipmpDescrPtr[]` – an array of `IPMP_DescriptorPointer`, as defined in 8.6.13, that points to the IPMP_Descriptors related to the elementary stream(s) described by this object descriptor. The array shall have any number of zero up to 255 elements.

`extDescr[]` – an array of `ExtensionDescriptors` as defined in 8.6.16. The array shall have any number of zero up to 255 elements.

8.6.4 InitialObjectDescriptor

8.6.4.1 Syntax

```
class InitialObjectDescriptor extends ObjectDescriptorBase : bit(8) tag=InitialObjectDescrTag
{
   bit(10) ObjectDescriptorID;
   bit(1) URL_Flag;
   bit(1) includeInlineProfileLevelFlag;
   const bit(4) reserved=0b1111;
   if (URL_Flag) {
      bit(8) URLlength;
      bit(8) URLstring[URLlength];
   } else {
      bit(8) ODProfileLevelIndication;
      bit(8) sceneProfileLevelIndication;
      bit(8) audioProfileLevelIndication;
      bit(8) visualProfileLevelIndication;
      bit(8) graphicsProfileLevelIndication;
      ES_Descriptor esDescr[1 .. 255];
      OCI_Descriptor ociDescr[0 .. 255];
      IPMP_DescriptorPointer ipmpDescrPtr[0 .. 255];
   }
   ExtensionDescriptor extDescr[0 .. 255];
}
```

8.6.4.2 Semantics

The `InitialObjectDescriptor` is a variation of the `ObjectDescriptor` specified in the previous subclause that allows to signal profile and level information for the content refered by it. It shall be used to gain initial access to ISO/IEC 14496 content (see 8.7.3).

Profile and level information indicated in the `InitialObjectDescriptor` indicates the profile and level supported by at least the first base layer stream (i.e. an elementary stream with a `streamDependenceFlag` set to 0) in each object descriptor depending on this initial object descriptor.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000037

`objectDescriptorId` – This syntax element uniquely identifies the `InitialObjectDescriptor` within its name scope (see 8.7.2.4). The value 0 is forbidden and the value 1023 is reserved.

`URL_Flag` – a flag that indicates the presence of a `URLstring`.

`includeInlineProfileLevelFlag` – a flag that, if set to one, indicates that the subsequent profile indications take into account the resources needed to process any content that might be inlined.

`URLlength` – the length of the subsequent `URLstring` in bytes.

`URLstring[]` – A string with a UTF-8 (ISO/IEC 10646-1) encoded URL that shall point to another `InitialObjectDescriptor`. Only the content of this object descriptor shall be returned by the delivery entity upon access to this URL. Within the current name scope, the new object descriptor shall be referenced by the `objectDescriptorId` of the object descriptor carrying the URLstring. On name scopes see 8.7.2.4. Permissible URLs may be constrained by profile and levels as well as by specific delivery layers.

`ODProfileLevelIndication` – an indication as defined in Table 3 of the object descriptor profile and level required to process the content associated with this `InitialObjectDescriptor`.

Table 3 - ODProfileLevelIndication Values

Value	Profile	Level
0x00	Forbidden	-
0x01-0x7F	reserved for ISO use	-
0x80-0xFD	user private	-
0xFE	no OD profile specified	-
0xFF	no OD capability required	-
NOTE — Usage of the value 0xFE indicates that the content described by this InitialObjectDescriptor does not comply to any OD profile specified in ISO/IEC 14496-1. Usage of the value 0xFF indicates that none of the OD profile capabilities are required for this content.		

`sceneProfileLevelIndication` – an indication as defined in Table 4 of the scene graph profile and level required to process the content associated with this `InitialObjectDescriptor`.

Table 4 - sceneProfileLevelIndication Values

Value	Profile	Level
0x00	Reserved for ISO use	-
0x01	Simple2D profile	L1
0x02-0x7F	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.		

`audioProfileLevelIndication` – an indication as defined in Table 5 of the audio profile and level required to process the content associated with this `InitialObjectDescriptor`.

Table 5 - audioProfileLevelIndication Values

Value	Profile	Level
0x00	Reserved for ISO use	-
0x01	Main Profile	L1
0x02	Main Profile	L2
0x03	Main Profile	L3

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000038

0x04	Main Profile	L4
0x05	Scalable Profile	L1
0x06	Scalable Profile	L2
0x07	Scalable Profile	L3
0x08	Scalable Profile	L4
0x09	Speech Profile	L1
0x0A	Speech Profile	L2
0x0B	Synthesis Profile	L1
0x0C	Synthesis Profile	L2
0x0D	Synthesis Profile	L3
0x0E-0x7F	reserved for ISO use	-
0x80-0xFD	user private	-
0xFE	no audio profile specified	-
0xFF	no audio capability required	-
NOTE — Usage of the value 0xFE indicates that the content described by this InitialObjectDescriptor does not comply to any audio profile specified in ISO/IEC 14496-3. Usage of the value 0xFF indicates that none of the audio profile capabilities are required for this content.		

`visualProfileLevelIndication` – an indication as defined in Table 6 of the visual profile and level required to process the content associated with this `InitialObjectDescriptor`.

Table 6 - visualProfileLevelIndication Values

Value	**Profile**	**Level**
0x00	Reserved for ISO use	-
0x01	Simple	L3
0x02	Simple	L2
0x03	Simple	L1
0x04	Simple Scalable	L2
0x05	Simple Scalable	L1
0x06	Core	L2
0x07	Core	L1
0x08	Main	L4
0x09	Main	L3
0x0A	Main	L2
0x0B	N-Bit	L2
0x0C	Hybrid	L2
0x0D	Hybrid	L1
0x0E	Basic Animated Texture	L2
0x0F	Basic Animated Texture	L1
0x10	Scalable Texture	L3
0x11	Scalable Texture	L2
0x12	Scalable Texture	L1
0x13	Simple Face Animation	L2
0x14	Simple Face Animation	L1
0x15-0x7F	reserved for ISO use	-
0x80-0xFD	user private	-
0xFE	no visual profile specified	-
0xFF	no visual capability required	-
NOTE — Usage of the value 0xFE indicates that the content described by this InitialObjectDescriptor does not comply to any visual profile specified in ISO/IEC 14496-2. Usage of the value 0xFF indicates that none of the visual profile capabilities are required for this content.		

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000039

`graphicsProfileLevelIndication` – an indication as defined in Table 7 of the graphics profile and level required to process the content associated with this `InitialObjectDescriptor`.

Table 7 - graphicsProfileLevelIndication Values

Value	Profile	Level
0x00	Reserved for ISO use	
0x01	Simple2D profile	L1
0x02-0x7F	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.		

`esDescr[]` – an array of `ES_Descriptors` as defined in 8.6.5. The array shall have any number of one up to 255 elements.

`ociDescr[]` – an array of OCI_Descriptors as defined in 8.6.18.2 that relates to the set of audio-visual objects that are described by this initial object descriptor. The array shall have any number of zero up to 255 elements.

`ipmpDescrPtr[]` – an array of `IPMP_DescriptorPointer`, as defined in 8.6.13, that points to the IPMP_Descriptors related to the elementary stream(s) described by this object descriptor. The array shall have any number of zero up to 255 elements.

`extDescr[]` – an array of `ExtensionDescriptors` as defined in 8.6.16. The array shall have any number of zero up to 255 elements.

8.6.5 ES_Descriptor

8.6.5.1 Syntax

```
class ES_Descriptor extends BaseDescriptor : bit(8) tag=ES_DescrTag {
   bit(16) ES_ID;
   bit(1) streamDependenceFlag;
   bit(1) URL_Flag;
   bit(1) OCRstreamFlag;
   bit(5) streamPriority;
   if (streamDependenceFlag)
      bit(16) dependsOn_ES_ID;
   if (URL_Flag) {
      bit(8) URLlength;
      bit(8) URLstring[URLlength];
   }
   if (OCRstreamFlag)
      bit(16) OCR_ES_Id;
   DecoderConfigDescriptor decConfigDescr;
   SLConfigDescriptor slConfigDescr;
   IPI_DescrPointer ipiPtr[0 .. 1];
   IP_IdentificationDataSet ipIDS[0 .. 255];
   IPMP_DescriptorPointer ipmpDescrPtr[0 .. 255];
   LanguageDescriptor langDescr[0 .. 255];
   QoS_Descriptor qosDescr[0 .. 1];
   RegistrationDescriptor regDescr[0 .. 1];
   ExtensionDescriptor extDescr[0 .. 255];
}
```

8.6.5.2 Semantics

The `ES_Descriptor` conveys all information related to a particular elementary stream and has three major parts.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000040

The first part consists of the ES_ID which is a unique reference to the elementary stream within its name scope (see 8.7.2.4), a mechanism to describe dependencies of elementary streams within the scope of the parent object descriptor and an optional URL string. Dependencies and usage of URLs are specified in 8.7.

The second part consists of the component descriptors which convey the parameters and requirements of the elementary stream.

The third part is a set of optional extension descriptors that support the inclusion of future extensions as well as the transport of private data in a backward compatible way.

ES_ID – This syntax element provides a unique label for each elementary stream within its name scope. The values 0 and 0xFFFF are reserved.

streamDependenceFlag – If set to one indicates that a dependsOn_ES_ID will follow.

URL_Flag – if set to 1 indicates that a URLstring will follow.

OCRstreamFlag – indicates that an OCR_ES_ID syntax element will follow.

streamPriority – indicates a relative measure for the priority of this elementary stream. An elementary stream with a higher streamPriority is more important than one with a lower streamPriority. The absolute values of streamPriority are not normatively defined.

dependsOn_ES_ID – is the ES_ID of another elementary stream on which this elementary stream depends. The stream with dependsOn_ES_ID shall also be associated to the same object descriptor as the current ES_Descriptor.

URLlength – the length of the subsequent URLstring in bytes.

URLstring[] – contains a UTF-8 (ISO/IEC 10646-1) encoded URL that shall point to the location of an SL-packetized stream by name. The parameters of the SL-packetized stream that is retrieved from the URL are fully specified in this ES_Descriptor. See also 8.7.3.3. Permissible URLs may be constrained by profile and levels as well as by specific delivery layers.

OCR_ES_ID – indicates the ES_ID of the elementary stream within the name scope (see 8.7.2.4) from which the time base for this elementary stream is derived. Circular references between elementary streams are not permitted.

decConfigDescr – is a DecoderConfigDescriptor as specified in 8.6.6.

slConfigDescr – is an SLConfigDescriptor as specified in 8.6.8.

ipiPtr[] – an array of zero or one IPI_DescrPointer as specified in 8.6.12.

ipIDS[] – an array of zero or more IP_IdentificationDataSet as specified in 8.6.9.

Each ES_Descriptor shall have either one IPI_DescrPointer or zero up to 255 IP_IdentificationDataSet elements. This allows to unambiguously associate an IP Identification to each elementary stream.

ipmpDescrPtr[] – an array of IPMP_DescriptorPointer, as defined in 8.6.13, that points to the IPMP_Descriptors related to the elementary stream described by this ES_Descriptor. The array shall have any number of zero up to 255 elements.

langDescr[] – an array of zero or one LanguageDescriptor structures as specified in 8.6.18.6. It indicates the language attributed to this elementary stream.

NOTE — Multichannel audio streams may be treated as one elementary stream with one ES_Descriptor by ISO/IEC 14496. In that case different languages present in different channels of the multichannel stream are not identifyable with a LanguageDescriptor.

qosDescr[] – an array of zero or one QoS_Descriptor as specified in 8.6.15.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000041

`extDescr[]` – an array of `ExtensionDescriptor` structures as specified in 8.6.16.

8.6.6 DecoderConfigDescriptor

8.6.6.1 Syntax

```
class DecoderConfigDescriptor extends BaseDescriptor : bit(8) tag=DecoderConfigDescrTag {
   bit(8) objectTypeIndication;
   bit(6) streamType;
   bit(1) upStream;
   const bit(1) reserved=1;
   bit(24) bufferSizeDB;
   bit(32) maxBitrate;
   bit(32) avgBitrate;
   DecoderSpecificInfo decSpecificInfo[0 .. 1];
   profileLevelIndicationIndexDescriptor profileLevelIndicationIndexDescr [0..255];
}
```

8.6.6.2 Semantics

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

`ObjectTypeIndication` – an indication of the object or scene description type that needs to be supported by the decoder for this elementary stream as per Table 8. For `streamType` values other than audioStream and visualStream, the `objectTypeIndication` shall be set to 0xFF, indicating that no object type is specified.

Table 8 - objectTypeIndication Values

Value	`ObjectTypeIndication` Description
0x00	Forbidden
0x01	Systems ISO/IEC 14496-1 [a]
0x02	Systems ISO/IEC 14496-1 [b]
0x03-0x1F	reserved for ISO use
0x20	Visual ISO/IEC 14496-2 [c]
0x21-0x3F	reserved for ISO use
0x40	Audio ISO/IEC 14496-3 [d]
0x41-0x5F	reserved for ISO use
0x60	Visual ISO/IEC 13818-2 Simple Profile
0x61	Visual ISO/IEC 13818-2 Main Profile
0x62	Visual ISO/IEC 13818-2 SNR Profile
0x63	Visual ISO/IEC 13818-2 Spatial Profile
0x64	Visual ISO/IEC 13818-2 High Profile
0x65	Visual ISO/IEC 13818-2 422 Profile
0x66	Audio ISO/IEC 13818-7 Main Profile
0x67	Audio ISO/IEC 13818-7 LowComplexity Profile
0x68	Audio ISO/IEC 13818-7 Scaleable Sampling Rate Profile
0x69	Audio ISO/IEC 13818-3
0x6A	Visual ISO/IEC 11172-2
0x6B	Audio ISO/IEC 11172-3
0x6C	Visual ISO/IEC 10918-1
0x6D - 0xBF	reserved for ISO use
0xC0 - 0xFE	user private
0xFF	no object type specified

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000042

[a] This object type shall be used for all streamTypes defined in ISO/IEC 14496-1 except IPMP streams.

[b] Includes associated Amendment(s) and Corrigendum(a).

[c] Includes associated Amendment(s) and Corrigendum(a). The actual object types are defined in ISO/IEC 14496-2 and are conveyed in the DecoderSpecificInfo as specified in ISO/IEC 14496-2, Annex K.

[d] Includes associated Amendment(s) and Corrigendum(a). The actual object types are defined in ISO/IEC 14496-3 and are conveyed in the DecoderSpecificInfo as specified in ISO/IEC 14496-3 subpart 1 subclause 6.2.1.

`streamType` – conveys the type of this elementary stream as per Table 9.

Table 9 - streamType Values

`streamType` value	Stream type description
0x00	Forbidden
0x01	ObjectDescriptorStream (see 8.5)
0x02	ClockReferenceStream (see 10.2.5)
0x03	SceneDescriptionStream (see 9.2.1)
0x04	VisualStream
0x05	AudioStream
0x06	MPEG7Stream
0x07	IPMPStream (see 8.3.2)
0x08	ObjectContentInfoStream (see 8.4.2)
0x09	MPEGJStream
0x0A - 0x1F	reserved for ISO use
0x20 - 0x3F	user private

`upStream` – indicates that this stream is used for upstream information.

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

`maxBitrate` – is the maximum bitrate in bits per second of this elementary stream in any time window of one second duration.

`avgBitrate` – is the average bitrate in bits per second of this elementary stream. For streams with variable bitrate this value shall be set to zero.

`decSpecificInfo[]` – an array of zero or one decoder specific information classes as specified in 8.6.7.

`ProfileLevelIndicationIndexDescr [0..255]` – an array of unique identifiers for a set of profile and level indications as carried in the `ExtensionProfileLevelDescr` defined in clause 8.6.19.

8.6.7 DecoderSpecificInfo

8.6.7.1 Syntax

```
abstract class DecoderSpecificInfo extends BaseDescriptor : bit(8) tag=DecSpecificInfoTag
{
   // empty. To be filled by classes extending this class.
}
```

8.6.7.2 Semantics

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

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000043

For values of `DecoderConfigDescriptor.objectTypeIndication` that refer to streams complying with ISO/IEC 14496-2 the syntax and semantics of decoder specific information are defined in Annex K of that part.

For values of `DecoderConfigDescriptor.objectTypeIndication` that refer to streams complying with ISO/IEC 14496-3 the syntax and semantics of decoder specific information are defined in section 1, clause 1.6 of that part.

For values of `DecoderConfigDescriptor.objectTypeIndication` that refer to scene description streams the semantics of decoder specific information is defined in 9.2.1.2.

For values of `DecoderConfigDescriptor.objectTypeIndication` that refer to streams complying with ISO/IEC 13818-7 the decoder specific information consists of the ADIF -header if it is present (or none if it is not present) and an access unit is a „raw_data_block()“ as defined in ISO/IEC 13818-7.

For values of `DecoderConfigDescriptor.objectTypeIndication` that refer to streams complying with ISO/IEC 13818-3 the decoder specific information is empty since all necessary data is in the bitstream frames itself. The access units in this case are the „frame()“ bitstream element as is defined in ISO/IEC 11172-3.

For values of `DecoderConfigDescriptor.objectTypeIndication` that refer to streams complying with ISO/IEC 10918-1, the decoder specific information is:

```
class JPEG_DecoderConfig extends DecoderSpecificInfo : bit(8) tag=DecSpecificInfoTag {
   int(16) headerLength;
   int(16) Xdensity;
   int(16) Ydensity;
   int(8) numComponents;
}
```

with

`headerLength` –indicates the number of bytes to skip from the beginning of the stream to find the first pixel of the image.

`Xdensity` and `Ydensity` – specify the pixel aspect ratio.

`numComponents` – indicates whether the image has Y component only or is Y, Cr, Cb. It shall be equal to 1 or 3.

8.6.8 SLConfigDescriptor

This descriptor defines the configuration of the sync layer header for this elementary stream. The specification of this descriptor is provided together with the specification of the sync layer in 10.2.3.

8.6.9 IP_IdentificationDataSet

8.6.9.1 Syntax

```
abstract class IP_IdentificationDataSet extends BaseDescriptor
     : bit(8) tag=ContentIdentDescrTag..SupplContentIdentDescrTag
{
   // empty. To be filled by classes extending this class.
}
```

8.6.9.2 Semantics

This class is an abstract base class that is extended by the descriptor classes that implement IP identification. A descriptor that allows to aggregate classes of type IP_IdentificationDataSet may actually aggregate any of the classes that extend IP_IdentificationDataSet.

8.6.10 ContentIdentificationDescriptor

8.6.10.1 Syntax

```
class ContentIdentificationDescriptor extends IP_IdentificationDataSet
     : bit(8) tag=ContentIdentDescrTag
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000044

```
{
   const bit(2) compatibility=0;
   bit(1)    contentTypeFlag;
   bit(1)    contentIdentifierFlag;
   bit(1)    protectedContent;
   bit(3)    reserved = 0b111;
   if (contentTypeFlag)
      bit(8)  contentType;
   if (contentIdentifierFlag) {
      bit(8)  contentIdentifierType;
      bit(8)  contentIdentifier[sizeOfInstance-2-contentTypeFlag];
   }
}
```

8.6.10.2 Semantics

The content identification descriptor is used to identify content. All types of elementary streams carrying content can be identified using this mechanism. The content types include audio, visual and scene description data. Multiple content identification descriptors may be associated to one elementary stream. These descriptors shall never be detached from the ES_Descriptor.

`compatibility` – must be set to 0.

`contentTypeFlag` – flag to indicate if a definition of the type of content is available.

`contentIdentifierFlag` – flag to indicate presence of creation ID.

`protectedContent` - if set to one indicates that the elementary streams that refer to this IP_IdentificationDataSet are protected by a method outside the scope of ISO/IEC 14496. The behavior of the terminal compliant with the ISO/IEC 14496 specifications when processing such streams is undefined.

`contentType` – defines the type of content using one of the values specified in Table 10.

Table 10 - contentType Values

0	Audio-visual
1	Book
2	Serial
3	Text
4	Item or Contribution (e.g. article in book or serial)
5	Sheet music
6	Sound recording or music video
7	Still Picture
8	Musical Work
9-254	Reserved for ISO use
255	Others

`contentIdentifierType` – defines a type of content identifier using one of the values specified in Table 11.

Table 11 - contentIdentifierType Values

0	ISAN	International Standard Audio-Visual Number
1	ISBN	International Standard Book Number
2	ISSN	International Standard Serial Number
3	SICI	Serial Item and Contribution Identifier
4	BICI	Book Item and Component Identifier
5	ISMN	International Standard Music Number
6	ISRC	International Standard Recording Code
7	ISWC-T	International Standard Work Code (Tunes)
8	ISWC-L	International Standard Work Code (Literature)

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000045

9	SPIFF	Still Picture ID
10	DOI	Digital Object Identifier
11-255	Reserved for ISO use	

`contentIdentifier` – international code identifying the content according to the preceding `contentIdentifierType`.

8.6.11 SupplementaryContentIdentificationDescriptor

8.6.11.1 Syntax

```
class SupplementaryContentIdentificationDescriptor extends
   IP_IdentificationDataSet : bit(8) tag= SupplContentIdentDescrTag
{
   bit(24) languageCode;
   bit(8)  supplContentIdentifierTitleLength;
   bit(8)  supplContentIdentifierTitle[supplContentIdentifierTitleLength];
   bit(8)  supplContentIdentifierValueLength;
   bit(8)  supplContentIdentifierValue[supplContentIdentifierValueLength];
}
```

8.6.11.2 Semantics

The supplementary content identification descriptor is used to provide extensible identifiers for content that are qualified by a language code. Multiple supplementary content identification descriptors may be associated to one elementary stream. These descriptors shall never be detached from the ES_Descriptor.

`language code` – This 24 bits field contains the ISO 639-2:1998 bibliographic three character language code of the language of the following text fields.

`supplementaryContentIdentifierTitleLength` – indicates the length of the subsequent `supplementaryContentIdentifierTitle` in bytes.

`supplementaryContentIdentifierTitle` – identifies the title of a supplementary content identifier that may be used when a numeric content identifier (see 8.6.10) is not available.

`supplementaryContentIdentifierValueLength` – indicates the length of the subsequent `supplementaryContentIdentifierValue` in bytes.

`supplementaryContentIdentifierValue` – identifies the value of a supplementary content identifer associated to the preceding `supplementaryContentIdentifierTitle`.

8.6.12 IPI_DescrPointer

8.6.12.1 Syntax

```
class IPI_DescrPointer extends BaseDescriptor : bit(8) tag=IPI_DescrPointerTag {
   bit(16) IPI_ES_Id;
}
```

8.6.12.2 Semantics

The `IPI_DescrPointer` class contains a reference to the elementary stream that includes the `IP_IdentificationDataSets` that are valid for this stream. This indirect reference mechanism allows to convey such descriptors only in one elementary stream while making references to it from any `ES_Descriptor` that shares the same information.

`ES_Descriptors` for elementary streams that are intended to be accessible regardless of the availability of a referred stream shall explicitly include their `IP_IdentificationDataSets` instead of using an `IPI_DescrPointer`.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000046

IPI_ES_Id – the ES_ID of the elementary stream whose ES_Descriptor contains the IP Information valid for this elementary stream. If the ES_Descriptor for IPI_ES_Id is not available, the IPI status of this elementary stream is undefined.

8.6.13 IPMP_DescriptorPointer

8.6.13.1 Syntax

```
class IPMP_DescriptorPointer extends BaseDescriptor : bit(8) tag=IPMP_DescrPointerTag {
   bit(8) IPMP_DescriptorID;
}
```

8.6.13.2 Semantics

IPMP_DescriptorID - ID of the referenced IPMP_Descriptor (see 8.6.14).

Presence of this descriptor in an object descriptor indicates that all streams referred to by embedded ES_Descriptors are subject to protection and management by the IPMP System specified in the referenced IPMP_Descriptor.

Presence of this descriptor in an ES_Descriptor indicates that the stream associated with this descriptor is subject to intellectual property management and protection by the IPMP System specified in the referenced IPMP_Descriptor.

8.6.14 IPMP Descriptor

8.6.14.1 Syntax

```
class IPMP_Descriptor() extends BaseDescriptor :  bit(8) IPMP_DescrTag {
   bit(8) IPMP_DescriptorID;
   unsigned int(16)   IPMPS_Type;
   if (IPMPS_Type == 0) {
      bit(8) URLString[sizeOfInstance-3];
   } else {
      bit(8) IPMP_data[sizeOfInstance-3];
   }
}
```

8.6.14.2 Semantics

The IPMP_Descriptor conveys IPMP information to an IPMP System. IPMP_Descriptors are conveyed in object descriptor streams via IPMP_DescriptorUpdates as specified in 8.5.5.6. They are not directly included in object descriptors or ES_Descriptors. IPMP_Descriptors are referenced by object descriptors or ES_Descriptors using IPMP_DescriptorPointers (see 8.6.13). An IPMP_Descriptor may be referenced by multiple object descriptors or ES_Descriptors.

IPMP_DescriptorID - a unique ID for this IPMP descriptor within its name scope (see 8.7.2.4).

IPMPS_Type - the type of the IPMP System. A zero value does not correspond to an IPMP System but is used to indicate the presence of a URL. A Registration Authority designated by ISO shall assign valid values for this field.

URLString[] - contains a UTF-8 (ISO/IEC 10646-1) encoded URL that shall point to the location of a remote IPMP_Descriptor. The IPMPS_Type of this IPMP_Descriptor can be 0 or not. If 0, another URL is referenced. This process continues until an IPMP_Descriptor with a non-zero IPMPS_Type is accessed.

IPMP_data - opaque data to control the IPMP System.

8.6.14.3 Implementation of a Registration Authority (RA)

ISO/IEC JTC 1/SC 29 shall issue a call for nominations from Member Bodies of ISO or National Committees of IEC in order to identify suitable organizations that will serve as the Registration Authority for the IPMPS_Type as defined in this clause. The selected organization shall serve as the Registration Authority. The so-named

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000047

Registration Authority shall execute its duties in compliance with Annex H of the JTC 1 Directives. The registered IPMPS_Type is hereafter referred to as the Registered Identifier (RID).

Upon selection of the Registration Authority, JTC 1 shall require the creation of a Registration Management Group (RMG) that will review appeals filed by organizations whose request for an RID to be used in conjunction with ISO/IEC 14496 has been denied by the Registration Authority.

Annex D provides information on the procedure for registering a unique IPMPS_Type value.

8.6.15 QoS_Descriptor

8.6.15.1 Syntax

```
class QoS_Descriptor extends BaseDescriptor : bit(8) tag=QoS_DescrTag {
   bit(8) predefined;
   if (predefined==0) {
      QoS_Qualifier qualifiers[];
   }
}
```

8.6.15.2 Semantics

The QoS_descriptor conveys the requirements that the ES has on the transport channel and a description of the traffic that this ES will generate. A set of predefined values is to be determined; customized values can be used by setting the `predefined` field to 0.

`predefined` – a value different from zero indicates a predefined QoS profile according to Table 12.

Table 12 - Predefined QoS Profiles

predefined value	**description**
0x00	Custom
0x01 - 0xff	Reserved

`qualifier` – an array of one or more `QoS_Qualifiers`.

8.6.15.3 QoS_Qualifier

8.6.15.3.1 Syntax

```
abstract aligned(8) expandable(2^28-1) class QoS_Qualifier : bit(8) tag=0x01..0xff {
   // empty. To be filled by classes extending this class.
}

class QoS_Qualifier_MAX_DELAY extends QoS_Qualifier : bit(8) tag=0x01 {
   unsigned int(32) MAX_DELAY;
}

class QoS_Qualifier_PREF_MAX_DELAY extends QoS_Qualifier : bit(8) tag=0x02 {
   unsigned int(32) PREF_MAX_DELAY;
}

class QoS_Qualifier_LOSS_PROB extends QoS_Qualifier : bit(8) tag=0x03 {
   double(32) LOSS_PROB;
}

class QoS_Qualifier_MAX_GAP_LOSS extends QoS_Qualifier : bit(8) tag=0x04 {
   unsigned int(32) MAX_GAP_LOSS;
}

class QoS_Qualifier_MAX_AU_SIZE extends QoS_Qualifier : bit(8) tag=0x41 {
   unsigned int(32) MAX_AU_SIZE;
}

class QoS_Qualifier_AVG_AU_SIZE extends QoS_Qualifier : bit(8) tag=0x42 {
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000048

```
   unsigned int(32) AVG_AU_SIZE;
}

class QoS_Qualifier_MAX_AU_RATE extends QoS_Qualifier : bit(8) tag=0x43 {
   unsigned int(32) MAX_AU_RATE;
}
```

8.6.15.3.2 Semantics

QoS qualifiers are defined as derived classes from the abstract `QoS_Qualifier` class. They are identified by means of their class tag. Unused tag values up to and including 0x7F are reserved for ISO use. Tag values from 0x80 up to and including 0xFE are user private. Tag values 0x00 and 0xFF are forbidden.

`MAX_DELAY` – Maximum end to end delay for the stream in microseconds.

`PREF_MAX_DELAY` – Preferred end to end delay for the stream in microseconds.

`LOSS_PROB` – Allowable loss probability of any single AU as a fractional value between 0.0 and 1.0.

`MAX_GAP_LOSS` – Maximum allowable number of consecutively lost AUs.

`MAX_AU_SIZE` – Maximum size of an AU in bytes.

`AVG_AU_SIZE` – Average size of an AU in bytes.

`MAX_AU_RATE` – Maximum arrival rate of AUs in AUs/second.

8.6.16 ExtensionDescriptor

8.6.16.1 Syntax

```
abstract class ExtensionDescriptor extends BaseDescriptor
: bit(8) tag = ExtDescrTagStartRange .. ExtDescrTagEndRange {
   // empty. To be filled by classes extending this class.
}
```

8.6.16.2 Semantics

This class is an abstract base class that may be extended for defining additional descriptors in future. The available range of class tag values allow ISO defined extensions as well as private extensions. A descriptor that allows to aggregate ExtensionDescriptor classes may actually aggregate any of the classes that extend ExtensionDescriptor. Extension descriptors may be ignored by a terminal that conforms to ISO/IEC 14496-1.

8.6.17 RegistrationDescriptor

The registration descriptor provides a method to uniquely and unambiguously identify formats of private data streams.

8.6.17.1 Syntax

```
class RegistrationDescriptor extends BaseDescriptor  : bit(8) tag=RegistrationDescrTag {
   bit(32) formatIdentifier;
   bit(8) additionalIdentificationInfo[sizeOfInstance-4];
}
```

8.6.17.2 Semantics

`formatIdentifier` – is a value obtained from a Registration Authority as designated by ISO.

`additionalIdentificationInfo` – The meaning of `additionalIdentificationInfo`, if any, is defined by the assignee of that `formatIdentifier`, and once defined, shall not change.

The registration descriptor is provided in order to enable users of ISO/IEC 14496-1 to unambiguously carry elementary streams with data whose format is not recognized by ISO/IEC 14496-1. This provision will permit

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000049

ISO/IEC 14496-1 to carry all types of data streams while providing for a method of unambiguous identification of the characteristics of the underlying private data streams.

In the following subclause and Annex D, the benefits and responsibilities of all parties to the registration of private data format are outlined.

8.6.17.2.1 Implementation of a Registration Authority (RA)

ISO/IEC JTC 1/SC 29 shall issue a call for nominations from Member Bodies of ISO or National Committees of IEC in order to identify suitable organizations that will serve as the Registration Authority for the formatIdentifier as defined in this subclause. The selected organization shall serve as the Registration Authority. The so-named Registration Authority shall execute its duties in compliance with Annex H of the JTC 1 Directives. The registered private data formatIdentifier is hereafter referred to as the Registered Identifier (RID).

Upon selection of the Registration Authority, JTC 1 shall require the creation of a Registration Management Group (RMG) which will review appeals filed by organizations whose request for an RID to be used in conjunction with ISO/IEC 14496-1 has been denied by the Registration Authority.

Annex D provides information on the procedure for registering a unique format identifier.

8.6.18 Object Content Information Descriptors

8.6.18.1 Overview

This subclause defines the descriptors that constitute the object content information. These descriptors may either be included in an `OCI_Event` in an OCI stream or be part of an object descriptor or `ES_Descriptor` as defined in 8.6.

8.6.18.2 OCI_Descriptor Class

8.6.18.2.1 Syntax

```
abstract class OCI_Descriptor extends BaseDescriptor
   : bit(8) tag= OCIDescrTagStartRange .. OCIDescrTagEndRange
{
   // empty. To be filled by classes extending this class.
}
```

8.6.18.2.2 Semantics

This class is an abstract base class that is extended by the classes specified in the subsequent clauses. A descriptor or an OCI_Event that allows to aggregate classes of type OCI_Descriptor may actually aggregate any of the classes that extend OCI_Descriptor.

8.6.18.3 Content classification descriptor

8.6.18.3.1 Syntax

```
class ContentClassificationDescriptor extends OCI_Descriptor
              : bit(8) tag= ContentClassificationDescrTag {
   bit(32) classificationEntity;
   bit(16) classificationTable;
   bit(8) contentClassificationData[sizeOfInstance-6];
}
```

8.6.18.3.2 Semantics

The content classification descriptor provides one or more classifications of the event information. The `classificationEntity` field indicates the organization that classifies the content. The possible values have to be registered with a registration authority to be identified.

`classificationEntity` – indicates the content classification entity. The values of this field are to be defined by a registration authority to be identified.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000050

`classificationTable` – indicates which classification table is being used for the corresponding classification. The classification is defined by the corresponding classification entity. 0x00 is a reserved value.

`contentClassificationData[]` – this array contains a classification data set using a non-default classification table.

8.6.18.4 Key Word Descriptor

8.6.18.4.1 Syntax

```
class KeyWordDescriptor extends OCI_Descriptor : bit(8) tag=KeyWordDescrTag {
   int i;
   bit(24) languageCode;
   bit(1) isUTF8_string;
   aligned(8) unsigned int(8) keyWordCount;
   for (i=0; i<keyWordCount; i++) {
      unsigned int(8) keyWordLength[[i]];
      if (isUTF8_string) then {
         bit(8) keyWord[[i]][keyWordLength[i]];
      } else {
         bit(16) keyWord[[i]][keyWordLength[i]];
      }
   }
}
```

8.6.18.4.2 Semantics

The key word descriptor allows the OCI creator/provider to indicate a set of key words that characterize the content. The choice of the key words is completely free but each time the key word descriptor appears, all the key words given are for the language indicated in `languageCode`. This means that, for a certain event, the key word descriptor must appear as many times as the number of languages for which key words are to be provided.

`languageCode` – contains the ISO 639-2:1998 bibliographic three character language code of the language of the following text fields.

`isUTF8_string` – indicates that the subsequent string is encoded with one byte per character (UTF-8). Else it is two byte per character.

`keyWordCount` – indicates the number of key words to be provided.

`keyWordLength` – specifies the length in characters of each key word.

`keyWord[]` – a Unicode (ISO/IEC 10646-1) encoded string that specifies the key word.

8.6.18.5 Rating Descriptor

8.6.18.5.1 Syntax

```
class RatingDescriptor extends OCI_Descriptor : bit(8) tag=RatingDescrTag {
   bit(32) ratingEntity;
   bit(16) ratingCriteria;
   bit(8)  ratingInfo[sizeOfInstance-6];
}
```

8.6.18.5.2 Semantics

This descriptor gives one or more ratings, originating from corresponding rating entities, valid for a specified country. The `ratingEntity` field indicates the organization which is rating the content. The possible values have to be registered with a registration authority to be identified. This registration authority shall make the semantics of the rating descriptor publicly available.

`ratingEntity` – indicates the rating entity. The values of this field are to be defined by a registration authority to be identified.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000051

`ratingCriteria` – indicates which rating criteria are being used for the corresponding rating entity. The value 0x00 is reserved.

`ratingInfo[]` – this array contains the rating information.

8.6.18.6 Language Descriptor

8.6.18.6.1 Syntax

```
class LanguageDescriptor extends OCI_Descriptor : bit(8) tag=LanguageDescrTag {
   bit(24) languageCode;
}
```

8.6.18.6.2 Semantics

This descriptor identifies the language of the corresponding audio/speech or text object that is being described.

`languageCode` – contains the ISO 639-2:1998 bibliographic three character language code of the corresponding audio/speech or text object that is being described.

8.6.18.7 Short Textual Descriptor

8.6.18.7.1 Syntax

```
class ShortTextualDescriptor extends OCI_Descriptor : bit(8) tag=ShortTextualDescrTag {
   bit(24) languageCode;
   bit(1) isUTF8_string;
   aligned(8) unsigned int(8) nameLength;
   if (isUTF8_string) then {
      bit(8) eventName[nameLength];
      unsigned int(8) textLength;
      bit(8) eventText[textLength];
   } else {
      bit(16) eventName[nameLength];
      unsigned int(8) textLength;
      bit(16) eventText[textLength];
   }
}
```

8.6.18.7.2 Semantics

The short textual descriptor provides the name of the event and a short description of the event in text form.

`languageCode` – contains the ISO 639-2:1998 bibliographic three character language code of the language of the following text fields.

`isUTF8_string` – indicates that the subsequent string is encoded with one byte per character (UTF-8). Else it is two byte per character.

`nameLength` – specifies the length in characters of the event name.

`eventName[]`– a Unicode (ISO/IEC 10646-1) encoded string that specifies the event name.

`textLength` – specifies the length in characters of the following text describing the event.

`eventText[]` – a Unicode (ISO/IEC 10646-1) encoded string that specifies the text description for the event.

8.6.18.8 Expanded Textual Descriptor

8.6.18.8.1 Syntax

```
class ExpandedTextualDescriptor extends OCI_Descriptor : bit(8) tag=ExpandedTextualDescrTag {
   int i;
   bit(24) languageCode;
   bit(1) isUTF8_string;
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000052

```
   aligned(8) unsigned int(8) itemCount;
   for (i=0; i<itemCount; i++){
      unsigned int(8) itemDescriptionLength[[i]];
      if (isUTF8_string) then {
         bit(8) itemDescription[[i]][itemDescriptionLength[i];
      } else {
         bit(16) itemDescription[[i]][itemDescriptionLength[i]];
      }
      unsigned int(8) itemLength[[i]];
      if (isUTF8_string) then {
         bit(8) itemText[[i]][itemLength[i]];
      } else {
         bit(16) itemText[[i]][itemLength[i]];
      }
   }
   unsigned int(8) textLength;
   int nonItemTextLength=0;
   while( textLength == 255 ) {
      nonItemTextLength += textLength;
      bit(8) textLength;
   }
   nonItemTextLength += textLength;
   if (isUTF8_string) then {
      bit(8) nonItemText[nonItemTextLength];
   } else {
      bit(16) nonItemText[nonItemTextLength];
   }
}
```

8.6.18.8.2 Semantics

The expanded textual descriptor provides a detailed description of an event, which may be used in addition to, or independently from, the short event descriptor. In addition to direct text, structured information in terms of pairs of description and text may be provided. An example application for this structure is to give a cast list, where for example the item description field might be “Producer” and the item field would give the name of the producer.

`languageCode` - contains the ISO 639-2:1998 bibliographic three character language code of the language of the following text fields.

`isUTF8_string` – indicates that the subsequent string is encoded with one byte per character (UTF-8). Else it is two byte per character.

`itemCount` – specifies the number of items to follow (itemised text).

`itemDescriptionLength` – specifies the length in characters of the item description.

`itemDescription[]` – a Unicode (ISO/IEC 10646-1) encoded string that specifies the item description.

`itemLength` – specifies the length in characters of the item text.

`itemText[]` – a Unicode (ISO/IEC 10646-1) encoded string that specifies the item text.

`textLength` – specifies the length in characters of the non itemised expanded text. The value 255 is used as an escape code, and it is followed by another `textLength` field that contains the length in bytes above 255. For lengths greater than 511 a third field is used, and so on.

`nonItemText[]` – a Unicode (ISO/IEC 10646-1) encoded string that specifies the non itemised expanded text.

8.6.18.9 Content Creator Name Descriptor

8.6.18.9.1 Syntax

```
class ContentCreatorNameDescriptor extends OCI_Descriptor
             : bit(8) tag= ContentCreatorNameDescrTag {
   int i;
   unsigned int(8) contentCreatorCount;
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000053

```
   for (i=0; i<contentCreatorCount; i++){
      bit(24) languageCode[[i]];
      bit(1) isUTF8_string[[i]];
      aligned(8) unsigned int(8) contentCreatorLength[[i]];
      if (isUTF8_string[[i]]) then {
         bit(8) contentCreatorName[[i]][contentCreatorLength[i]];
      } else {
         bit(16) contentCreatorName[[i]][contentCreatorLength[i]];
      }
   }
}
```

8.6.18.9.2 Semantics

The content creator name descriptor indicates the name(s) of the content creator(s). Each content creator name may be in a different language.

`contentCreatorCount` – indicates the number of content creator names to be provided.

`languageCode` – contains the ISO 639-2:1998 bibliographic three character language code of the language of the following text fields. Note that for languages that only use Latin characters, just one byte per character is needed in Unicode (ISO/IEC 10646-1).

`isUTF8_string` – indicates that the subsequent string is encoded with one byte per character (UTF-8). Else it is two byte per character.

`contentCreatorLength[[i]]` – specifies the length in characters of each content creator name.

`contentCreatorName[[i]][]` – a Unicode (ISO/IEC 10646-1) encoded string that specifies the content creator name.

8.6.18.10 Content Creation Date Descriptor

8.6.18.10.1 Syntax

```
class ContentCreationDateDescriptor extends OCI_Descriptor
             : bit(8) tag= ContentCreationDateDescrTag {
   bit(40) contentCreationDate;
}
```

8.6.18.10.2 Semantics

This descriptor identifies the date of the content creation.

`contentCreationDate` – contains the content creation date of the data corresponding to the event in question, in Universal Time, Co-ordinated (UTC) and Modified Julian Date (MJD) (see Annex F). This field is coded as 16 bits giving the 16 least significant bits of MJD followed by 24 bits coded as 6 digits in 4-bit Binary Coded Decimal (BCD). If the content creation date is undefined all bits of the field are set to 1.

8.6.18.11 OCI Creator Name Descriptor

8.6.18.11.1 Syntax

```
class OCICreatorNameDescriptor extends OCI_Descriptor
             : bit(8) tag=OCICreatorNameDescrTag {
   int i;
   unsigned int(8) OCICreatorCount;
   for (i=0; i<OCICreatorCount; i++) {
      bit(24) languageCode[[i]];
      bit(1) isUTF8_string;
      aligned(8) unsigned int(8) OCICreatorLength[[i]];
      if (isUTF8_string) then {
         bit(8) OCICreatorName[[i]][OCICreatorLength];
       } else {
         bit(16) OCICreatorName[[i]][OCICreatorLength];
      }
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000054

```
   }
}
```

8.6.18.11.2 Semantics

The name of OCI creators descriptor indicates the name(s) of the OCI description creator(s). Each OCI creator name may be in a different language.

`OCICreatorCount` **–** indicates the number of OCI creators.

`languageCode[[i]]` **–** contains the ISO 639-2:1998 bibliographic three character language code of the language of the following text fields.

`isUTF8_string` – indicates that the subsequent string is encoded with one byte per character (UTF-8). Else it is two byte per character.

`OCICreatorLength[[i]]` **–** specifies the length in characters of each OCI creator name.

`OCICreatorName[[i]]` **–** a Unicode (ISO/IEC 10646-1) encoded string that specifies the OCI creator name.

8.6.18.12 OCI Creation Date Descriptor

8.6.18.12.1 Syntax

```
class OCICreationDateDescriptor extends OCI_Descriptor
             : bit(8) tag=OCICreationDateDescrTag {
   bit(40) OCICreationDate;
}
```

8.6.18.12.2 Semantics

This descriptor identifies the creation date of the OCI description.

`OCICreationDate` **-** This 40-bit field contains the OCI creation date for the OCI data corresponding to the event in question, in Co-ordinated Universal Time (UTC) and Modified Julian Date (MJD) (see Annex F). This field is coded as 16 bits giving the 16 least significant bits of MJD followed by 24 bits coded as 6 digits in 4-bit Binary Coded Decimal (BCD). If the OCI creation date is undefined all bits of the field are set to 1.

8.6.18.13 SMPTE Camera Position Descriptor

8.6.18.13.1 Syntax

```
class SmpteCameraPositionDescriptor extends OCI_Descriptor : bit (8)
tag=SmpteCameraPositionDescrTag {
   unsigned int (8) cameraID;
   unsigned int (8) parameterCount;
   for (i=0; i<parameterCount; i++) {
      bit (8) parameterID;
      bit (32) parameter;
   }
}
```

8.6.18.13.2 Semantics

The SMPTE metadata descriptor provides metadata defined by the Proposed SMPTE Standard 315M of “camera positioning information conveyed by ancillary data packets.” The SMPTE 315M defines IDs and data formats for the following parameters:

- camera relative position
- camera pan
- camera tilt
- camera roll

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000055

- origin of world coordinate longitude
- origin of world coordinate latitude
- origin of world coordinate altitude
- vertical angle of view
- focus distance
- lens opening (iris or F-value)
- time address information
- object relative position

`cameraID` - contains the b(0-7) of C-ID of the UDW in Figure 6.

`parameterCount` - specifies the number of parameters and is equal to (the Data Count Word (DC) – 18) / 5.

`parameterID` - contains the b(0-7) of i-th IDn of the UDW.

`parameter` - contains the i-th Parameter n of the UDW (b(0-7) of each word).

8.6.18.13.3 Packet structure defined by SMPTE 315M

Ancillary data packet and space format is defined by ANSI/SMPTE 291M. The SMPTE 315M is one of the registered formats for a specific application of user data space defined by the 291M. The structure of binary-type camera positioning data packets described in the SMPTE 315M is illustrated in Figure 6.

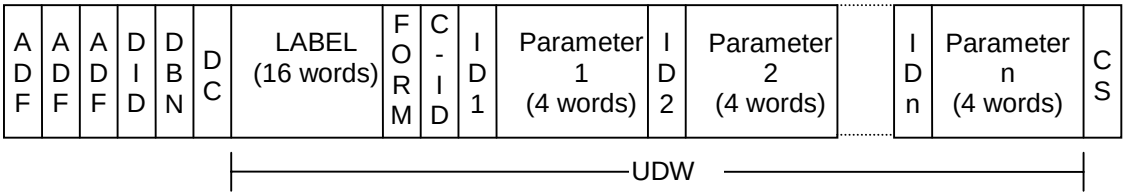


Figure 6 - Binary-type camera positioning data packets (SMPTE 315M)

Ancillary data is defined as 10-bit words. B(0-7), b8 and b9 represent actual data, even parity for b(0-7) and not b8 respectively except ADF.

ADF:	Ancillary Data Flag (000 h, 3ff h, 3ff h)
DID:	Data Identification Word (2f0 h)
DBN:	Data Block Number Word
DC:	Data Count Word
UDW:	User Data Words (up to 255 words)
LABEL:	SMPTE label for metadata of class “camera positioning information” (16 words)
FORM:	Data Type Identification Flag Word (1 word)
C-ID:	Camera Identification Word (1 word)
IDn:	Parameter Identification Word (1 word for each parameter)

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000056

Parameter n: Parameter Data Words (4 words for each parameter)

CS: Checksum Word

The 4 words LABEL(8-11) of LABEL(0-15) shall be set to ‘C’, ‘A’, ‘P’, ‘O’. The Data Type Identification Flag Word (FORM) indicates the data type of the camera identification word (C-ID), parameter identification word (IDn) and parameter data word (Parameter n) contained in the packet. In case of binary-type camera positioning data FORM(0-1) shall be set to 0 h.

8.6.19 Extension Profile Level Descriptor

8.6.19.1 Syntax

```
class ExtensionProfileLevelDescriptor() extends BaseDescriptor :  bit(8)
ExtensionProfileLevelDescrTag {
   bit(8) profileLevelIndicationIndex;
   bit(8) ODProfileLevelIndication;
   bit(8) sceneProfileLevelIndication;
   bit(8) audioProfileLevelIndication;
   bit(8) visualProfileLevelIndication;
   bit(8) graphicsProfileLevelIndication;
   bit(8) MPEGJProfileLevelIndication;
}
```

8.6.19.2 Semantics

The `ExtensionProfileLevelDescriptor` conveys profile and level extension information. This descriptor is used to signal a profile and level indication set and its unique index and can be extended by ISO to signal any future set of profiles and levels.

`profileLevelIndicationIndex` – a unique identifier for the set of profile and level indications described in this descriptor within the name scope defined by the IOD.

`ODProfileLevelIndication` – an indication of the profile and level required to process object descriptor streams associated with the `InitialObjectDescriptor` containing this Extension Profile and Level descriptor.

`sceneProfileLevelIndication` – an indication of the profile and level required to process the scene graph nodes within scene description streams associated with the `InitialObjectDescriptor` containing this Extension Profile and Level descriptor.

`audioProfileLevelIndication` – an indication of the profile and level required to process audio streams associated with the `InitialObjectDescriptor` containing this Extension Profile and Level descriptor.

`visualProfileLevelIndication` – an indication of the profile and level required to process visual streams associated with the `InitialObjectDescriptor` containing this Extension Profile and Level descriptor.

`graphicsProfileLevelIndication` – an indication of the profile and level required to process graphics nodes within scene description streams associated with the `InitialObjectDescriptor` containing this Extension Profile and Level descriptor.

`MPEGJProfileLevelIndication` – an indication as defined in Table 13 of the MPEG-J profile and level required to process the content associated with the InitialObjectDescriptor containing this Extension Profile and Level descriptor.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000057

Table 13 - MPEGJProfileLevelIndication Values

Value	Profile	Level
0x00	Reserved for ISO use	-
0x01	Personal profile	L1
0x02	Main profile	L1
0x03-0x7F	reserved for ISO use	-
0x80-0xFD	user private	-
0xFE	no MPEG-J profile specified	-
0xFF	no MPEG-J 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		

8.6.20 Profile Level Indication Index Descriptor

8.6.20.1 Syntax

```
class ProfileLevelIndicationIndexDescriptor () extends BaseDescriptor
:  bit(8) ProfileLevelIndicationIndexDescrTag {
   bit(8) profileLevelIndicationIndex;
}
```

8.6.20.2 Semantics

`profileLevelIndicationIndex` – a unique identifier for the set of profile and level indications described in this descriptor within the name scope defined by the IOD.

8.7 Rules for Usage of the Object Description Framework

8.7.1 Aggregation of Elementary Stream Descriptors in a Single Object Descriptor

8.7.1.1 Overview

An object descriptor shall aggregate the descriptors for the set of elementary streams that is intended to be associated to a single node of the scene description and that usually relate to a single audio-visual object. The set of streams may convey a scaleable content representation as well as multiple alternative content representations, e.g., multiple qualities or different languages. Additional streams with IPMP and object content information may be attached.

These options are described by the ES_Descriptor syntax elements `streamDependenceFlag`, `dependsOn_ES_ID`, as well as `streamType`. The semantic rules for the aggregation of elementary stream descriptors within one object descriptor (OD) are specified in this subclause.

8.7.1.2 Aggregation of Elementary Streams with the same streamType

An OD may aggregate multiple ES_Descriptors with the same `streamType` of either visualStream, audioStream or SceneDescriptionStream. However, descriptors for streams with two of these types shall not be mixed within one OD.

8.7.1.3 Aggregation of Elementary Streams with Different streamTypes

In the following cases ESs with different `streamType` may be aggregated:

- An OD may aggregate zero or one additional ES_Descriptor with `streamType` = ObjectContentInfoStream (see 8.4.2). This ObjectContentInfoStream shall be valid for the content conveyed through the other visual, audio or scene description streams whose descriptors are aggregated in this OD.

- An OD may aggregate zero or one additional ES_Descriptors with `streamType` = ClockReferenceStream (see 10.2.5). This ClockReferenceStream shall be valid for the ES within the name scope that refer to the ES_ID of this ClockReferenceStream in their SLConfigDescriptor.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000058

- An OD may aggregate zero or more additional ES_Descriptors with `streamType` = IPMPStream (see 8.3.2). This IPMPStream shall be valid for the content conveyed through the other visual, audio or scene description streams whose descriptors are aggregated in this OD.

8.7.1.4 Aggregation of scene description streams and object descriptor streams

An object descriptor that aggregates one or more ES_Descriptors of `streamType` = SceneDescriptionStream may aggregate any number of additional ES_Descriptors with `streamType` = ObjectDescriptorStream. ES_Descriptors of `streamType` = ObjectDescriptorStream shall not be aggregated in object descriptors that do not contain ES_Descriptors of `streamType` = SceneDescriptionStream.

This means that scene description and object descriptor streams are always combined within one object descriptor. The dependencies between these streams are defined in 8.7.1.5.2.

8.7.1.5 Elementary Stream Dependencies

8.7.1.5.1 Independent elementary streams

ES_Descriptors within one OD with the same `streamType` of either audioStream, visualStream or SceneDescriptionStream that have `streamDependenceFlag=0` refer to independent elementary streams. Such independent elementary streams shall convey alternative representations of the same content. Only one of these representations shall be selected for use in the scene.

NOTE — Independent ESs should be ordered within an OD according to the content creator's preference. The ES that is first in the list of ES aggregated to one object descriptor should be preferable over an ES that follows later. In case of audio streams, however, the selection should for obvious reasons be done according to the prefered language of the receiving terminal.

8.7.1.5.2 Dependent elementary streams

ES_Descriptors within one OD with the same `streamType` of either audioStream, visualStream, SceneDescriptionStream or ObjectDescriptorStream that have `streamDependenceFlag=1` refer to dependent elementary streams. The ES_ID of the stream on which the dependent elementary stream depends is indicated by `dependsOn_ES_ID`. The ES_Descriptor with this ES_ID shall be aggregated to the same OD. One independent elementary stream per object descriptor and all its dependent elementary streams may be selected for concurrent use in the scene.

Stream dependencies are governed by the following rules:

- For dependent ES of `streamType` equal to either audioStream or visualStream the dependent ES shall have the same `streamType` as the ES on which it depends. This implies that the dependent stream contains enhancement information to the one it depends on. The precise semantic meaning of the dependencies is opaque at this layer.

- An ES with a `streamType` of SceneDescriptionStream shall only depend on an ES with `streamType` of SceneDescriptionStream or ObjectDescriptorStream.

  Dependency on an ObjectDescriptorStream implies that the ObjectDescriptorStream contains the object descriptors that are refered to by this SceneDescriptionStream.

  Dependency on a SceneDescriptionStream implies that the dependent stream contains enhancement information to the one it depends on. The dependent SceneDescriptionStream shall depend on the same ObjectDescriptorStream on which the other SceneDescriptionStream depends.

- An ES with a `streamType` of ObjectDescriptorStream shall only depend on an ES with a `streamType` of SceneDescriptionStream. This dependency does not have implications for the object descriptor stream.

  Only if a second stream with `streamType` of SceneDescriptionStream depends on this stream with `streamType` = ObjectDescriptorStream, it implies that the second SceneDescriptionStream depends on the first SceneDescriptionStream. The object descriptors in the ObjectDescriptorStream shall only be valid for the second SceneDescriptionStream.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000059

- An ES that flows upstream, as indicated by `DecoderConfigDescriptor.upStream = 1` shall always depend upon another ES that has the `upStream` flag set to zero. This implies that this upstream is associated to the downstream it depends on. If the downstream is an ObjectDescriptorStream or SceneDescriptionStream, the upstream shall be associated to all downstreams specified in that ObjectDescriptorStream or SceneDescriptionStream.

- The availability of the dependent stream is undefined if an ES_Descriptor for the stream it depends upon is not available.

8.7.2 Linking Scene Description and Object Descriptors

8.7.2.1 Associating Object Descriptors to BIFS Nodes

Some BIFS nodes contain an **url** field. Such nodes are associated to their elementary stream resources (if any) via an object descriptor. The association is established by means of the `objectDescriptorID`, as specified in 9.3.7.20.2. The name scope for this ID is specified in 8.7.2.4.

Each BIFS node requires a specific streamType (audio, visual, inlined scene description, etc.) for its associated elementary streams. The associated object descriptor shall contain ES_Descriptors with this streamType. The behavior of the terminal is undefined if an object descriptor contains ES_Descriptors with stream types that are incompatible with the associated BIFS node.

Note that commands adding or removing object descriptors need not be co-incident in time with the addition or removal of BIFS nodes in the scene description that refer to such an object descriptor. However, the behavior of the terminal is undefined if a BIFS node in the scene description references an object descriptor that is no longer valid.

The terminal shall gracefully handle references from the scene description to object descriptors that are not currently available.

8.7.2.2 Multiple scene description and object description streams

An object descriptor that is associated to an **Inline** node of the scene description or that represents the primary access to content compliant with the ISO/IEC 14496 specifications (initial object descriptor) aggregates as a minimum, one scene description stream and the corresponding object descriptor stream (if additional elementary streams need to be referenced).

However, it is permissible to split both the scene description and the object descriptors in multiple streams. This allows a bandwidth-scaleable encoding of the scene description. Each stream shall contain a valid sequence of access units as defined in 9.2.1.3 and 8.5.2, respectively. All resulting scene description streams and object descriptor streams shall remain aggregated in a single object descriptor. The dependency mechanism shall be used to indicate how the streams depend on each other.

All streams shall continue to be processed by a single scene description and object descriptor decoding process, respectively. The time stamps of the access units in different streams shall be used to re-establish the original order of access units.

NOTE — This form of partitioning of the scene description and the object descriptor streams in multiple streams is not visible in the scene description itself.

8.7.2.3 Scene and Object Description in Case of Inline Nodes

The BIFS scene description allows to recursively partition a scene through the use of **Inline** nodes (see 9.4.2.62). Each **Inline** node is associated to an object descriptor that points to at least one additional scene description stream as well as another object descriptor stream (if additional elementary streams need to be referenced). An example for such a hierarchical scene description can be found in 8.7.3.8.2.

8.7.2.4 Name Scope of Identifiers

The scope of the `objectDescriptorID`, `ES_ID` and `IPMP_DescriptorID` identifiers that label the object descriptors, elementary stream descriptors and IPMP descriptors, respectively, is defined as follows. This definition

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000060

is based on the restriction that associated scene description and object descriptor streams shall always be aggregated in a single object descriptor, as specified in 8.7.1.4. The following rule defines the name scope:

- Two `objectDescriptorID`, `ES_ID` or `IPMP_DescriptorID` as well as `nodeID` and `ROUTEID` identifiers belong to the same name scope if and only if these identifiers occur in elementary streams with a `streamType` of either ObjectDescriptorStream or SceneDescriptionStream that are aggregated in a single initial object descriptor or a single object descriptor associated to an **Inline** node.

NOTE 1 — Hence, the difference between the two methods specified in 8.7.2.2 and 8.7.2.3 above to partition a scene description in multiple streams is that the first method allows multiple scene description streams that refer to the same name scope while an **Inline** node opens a new name scope.

NOTE 2 — This implies that a URL in an object descriptor opens a new name scope since it points to an object descriptor that is not carried in the same ObjectDescriptorStream.

NOTE 3 — It is recommendable to extend the name scope for the stream related identifiers, namely, ES_ID and IPMP_DescriptorID, to the underlying communication session that is established as described in 8.7.3.6. This implies that those identifiers will be unique within such a communication session.

8.7.2.5 Reuse of identifiers

Within a single name scope an ES_ID identifier shall always refer to a single instance of an elementary stream.

Note: If two ES_Descriptors within two object descriptors reference a given ES_ID, this means that the second reference may not receive the stream content from the beginning if the first reference has already started the stream.

For reasons of error resilience, it is recommended not to reuse `objectDescriptorID` and `ES_ID` identifiers to identify more than one object or elementary stream, respectively, within one presentation. That means, if an object descriptor or elementary stream descriptor is removed by means of an OD command and later on reinstalled with another OD command, then it shall still point to the same content item as before.

8.7.3 ISO/IEC 14496 Content Access

8.7.3.1 Introduction

In order to access ISO/IEC 14496 compliant content it is a pre-condition that an initial object descriptor to such content is known through means outside the scope of ISO/IEC 14496. The subsequent content access procedure is specified conceptually, using a number of walk throughs. Its precise definition depends on the chosen delivery layer.

For applications that implement the DMIF Application Interface (DAI) specified in ISO/IEC 14496-6 which abstracts the delivery layer, a mapping of the conceptual content access procedure to calls of the DAI is specified in 8.7.3.9.

The content access procedure determines the set of required elementary streams, requests their delivery and associates them to the scene description. The selection of a subset of elementary streams suitable for a specific ISO/IEC 14496 terminal is possible, either based on profiles or on inspection of the set of object descriptors.

8.7.3.2 The Initial Object Descriptor

Initial object descriptors convey information about the profiles required by the terminal compliant with ISO/IEC 14496 specifications to be able to process the described content. This profile information summarizes the complexity of the content referenced directly or indirectly through this initial object descriptor, i.e., it indicates the overall terminal capabilities required to decode and present this content. Therefore initial object descriptors constitute self-contained access points to content compliant with ISO/IEC 14496 specifications.

There are two constraints to this general statement:

- If the `includeInlineProfileLevelFlag` of the initial object descriptor is not set, the complexity of any inlined content is not included in the profile indications.

- In addition to the elementary streams that are decodable by the terminal conforming to the indicated profiles, alternate content representations might be available. This is further explained in 8.7.3.4.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000061

An initial object descriptor may be conveyed by means not defined in ISO/IEC 14496. The content may be accessed starting from the elementary streams that are described by this initial object descriptor, usually one or more scene description streams and zero or more object descriptor streams.

Content refered to by an initial object descriptor may itself be referenced from another piece of ISO/IEC 14496 content. In this case, the initial object descriptor will be conveyed in an object descriptor stream and the `OD_IDs` of both initial object descriptors and ordinary object descriptors belong to the same name scope.

Ordinary object descriptors may be used as well to describe scene description and object descriptor streams. However, since they do not carry profile information, they can only be used to access content if that information is either not required by the terminal or is obtained by other means.

8.7.3.3 Usage of URLs in the Object Descriptor Framework

URLs in the object description framework serve to locate either inlined ISO/IEC 14496 content or the elementary stream data associated to individual audio-visual objects.

URLs in ES_Descriptors locate elementary stream data that shall be delivered as SL-packetized stream by the delivery entity associated to the current name scope. The complete description of the stream (its ES_Descriptor) is available locally.

URLs in object descriptors locate an object descriptor at a remote location. Only the content of this object descriptor shall be returned by the delivery entity upon access to this URL. This implies that the description of the resources for the associated BIFS node or the inlined content is only available at the remote location. Note, however, that depending on the value of `includeInlineProfileLevelFlag` in the initial object descriptor, the global resources needed may already be known (i.e., including remote, inlined portions).

8.7.3.4 Selection of Elementary Streams for an Audio-Visual Object

Elementary streams are attached through their object descriptor to appropriate BIFS nodes which, in most cases, constitute the representation of a single audio-visual object in the scene. The selection of one or more ESs for each BIFS node may be governed by the profile indications that are conveyed in the initial object descriptor. All object descriptors shall at least include one elementary stream with suitable object type to satisfy the initially signaled profiles.

Additionally, object descriptors may aggregate ES_Descriptors for elementary streams that require more computing or bandwidth resources. Those elementary streams may be used by the receiving terminal if it is capable of processing them.

In case initial object descriptors do not indicate any profile and level or if profile and level indications are disregarded, an alternative to the profile driven selection of streams exists. The receiving terminal may evaluate the ES_Descriptors of all available elementary streams for each BIFS node and choose by some non-standardized way for which subset it has sufficient resources to decode them while observing the constraints specified in this subclause.

NOTE — Some restrictions on the selection of and access to elementary streams might exist if a set of elementary streams shares a single object time base (see 10.2.6).

8.7.3.5 Content access in “push” and “pull” scenarios

In an interactive, or “pull” scenario, the receiving terminal actively requests the establishment of sessions and the delivery of content, i.e., streams. This usually involves a session and channel set up protocol between sender and receiver. This protocol is not specified here. However, the conceptual steps to be performed are the same in all cases and are specified in the subsequent clauses.

In a broadcast, or “push” scenario, the receiving terminal passively processes what it receives. Instead of issuing requests for session or channel set up the receiving terminal shall evaluate the relevant descriptive information that associates ES_IDs to their transport channel. The syntax and semantics of this information is outside the scope of ISO/IEC 14496, however, it needs to be present in any delivery layer implementation. This allows the terminal to gain access to the elementary streams forming part of the content.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000062

8.7.3.6 Content access through a known Object Descriptor

8.7.3.6.1 Pre-conditions

- An object descriptor has been acquired. This may be an initial object descriptor.
- The object descriptor contains ES_Descriptors pointing to object descriptor stream(s) and scene description stream(s) using ES_IDs.
- A communication session to the source of these streams is established.
- A mechanism exists to open a channel that takes user data as input and provides some returned data as output.

8.7.3.6.2 Content Access Procedure

The content access procedure shall be equivalent to the following:

1. The object descriptor is evaluated and the ES_ID for the streams that are to be opened are determined.
2. Requests for opening the selected ESs are made, using a suitable channel set up mechanism with the ES_IDs as parameter.
3. The channel set up mechanism shall return handles to the streams that correspond to the requested list of ESs.
4. Requests for delivery of the selected ESs are made.
5. Interactive scenarios: Delivery of streams starts. All scenarios: The streams now become accessible.
6. Scene description and object descriptor stream are evaluated.
7. Further streams are opened as needed with the same procedure, starting at step 1.

8.7.3.7 Content access through a URL in an Object Desciptor

8.7.3.7.1 Pre-conditions

- A URL to an object descriptor or an initial object descriptor has been acquired.
- A mechanism exists to open a communication session that takes a URL as input and provides some returned data as output.

8.7.3.7.2 Content access procedure

The content access procedure shall be equivalent to the following:

1. A connection to the source of the URL is made, using a suitable service set up call.
2. The service set up call shall return data consisting of a single object descriptor.
3. Continue at step 1 in 8.7.3.6.2.

8.7.3.8 Content access through a URL in an elementary stream descriptor

8.7.3.8.1 Pre-conditions

- An ES_Descriptor pointing to a stream through a URL has been aquired. (Note that the ES_Descriptor fully specifies the configuration of the stream.)
- A mechanism exists to open a communication session that takes a URL as input and provides some returned data as output.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000063

- A mechanism exists to open a channel that takes user data as input and provides some returned data as output.

8.7.3.8.2 Content access procedure

The content access procedure shall be equivalent to the following:

1. A request to open the communication session is made, using a suitable session set up mechanism with the URL as parameter.

2. The session set up mechanism shall return a handle to the session that corresponds to the requested URL.

3. Request to open the stream is made, using a suitable channel set up mechanism.

4. The channel set up mechanism shall return a handle to the stream that corresponds to the originally requested URL.

5. Requests for delivery of the selected stream are made.

6. Interactive scenarios: Delivery of stream starts. All scenarios: The stream now becomes accessible.

EXAMPLE — Access to Complex Content

The example in Figure 7 shows a complex piece of ISO/IEC 14496 content, consisting of three parts. The upper part is a scene accessed through its initial object descriptor. It contains, among others a visual and an audio stream. A second part of the scene is inlined and accessed through its initial object descriptor that is pointed to (via URL) in the object descriptor stream of the first scene. Utilization of the initial object descriptor allows the signaling of profile information for the second scene. Therefore this scene may also be used without the first scene. The second scene contains, among others, a scaleably encoded visual object and an audio object. A third scene is inlined and accessed via the ES_IDs of its object descriptor and scene description streams. These ES_IDs are known from an object descriptor conveyed in the object descriptor stream of the second scene. Note that this third scene is not accessed through an initial object descriptor. Therefore the profile information for this scene need to be included in the profile information for the second scene.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000064

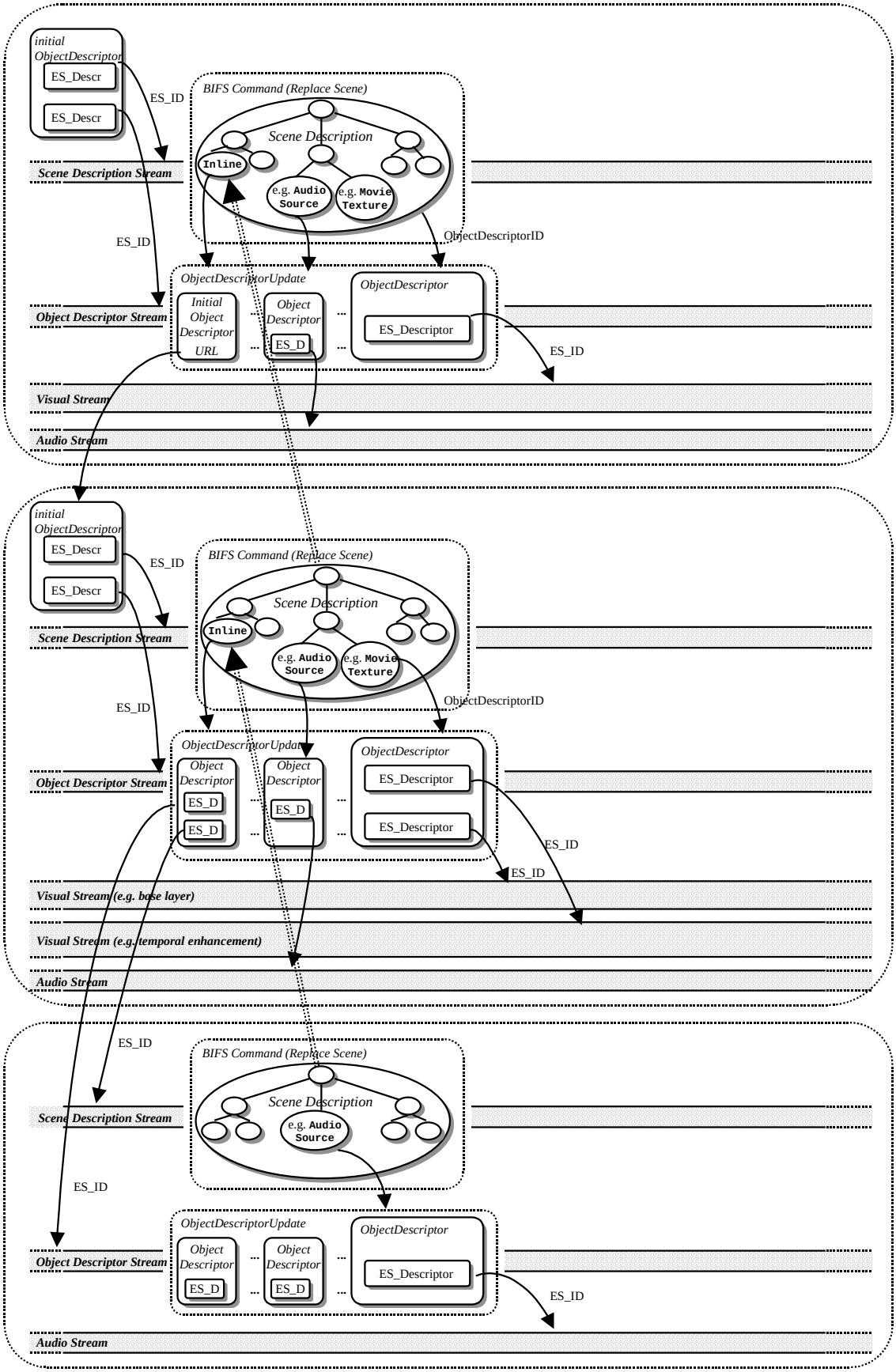


Figure 7 - Complex content example

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000065

8.7.3.9 Mapping of Content Access Procedure to DAI calls

The following two DAI primitives, quoted from ISO/IEC 14496-6, subclause 10.4, are required to implement the content access procedure described in 8.7.3.6 to 8.7.3.8:

DA_ServiceAttach (IN: URL, uuDataInBuffer, uuDataInLen;
OUT: response, serviceSessionId, uuDataOutBuffer, uuDataOutLen)

DA_ChannelAdd (IN: serviceSessionId, loop(qosDescriptor, direction, uuDataInBuffer, uuDataInLen);
OUT: loop(response, channelHandle, uuDataOutBuffer, uuDataOutLen))

DA_ServiceAttach is used to implement steps 1 and 2 of 8.7.3.7.2. The URL shall be passed to the IN: URL parameter. UuDataInBuffer shall remain empty. The returned serviceSessionId shall be kept for future reference to this URL. UuDataOutBuffer shall contain a single object descriptor.

DA_ChannelAdd is used to implement steps 2 and 3 of 8.7.3.6.2. serviceSessionId shall be the identifier for the service session that has supplied the object descriptor that includes the ES_Descriptor that is currently processed. QosDescriptor shall be the QoS_Descriptor of this ES_Descriptor, direction shall indicate upstream or downstream channels according to the `DecoderConfigDescriptor.upstream` flag. UuDataInBuffer shall contain the ES_ID of this ES_Descriptor. On successful return, channelHandle shall contain a valid, however, not normative handle to the accessible stream.

DA_ChannelAdd is used to implement steps 1 and 2 of 8.7.3.8.2. serviceSessionId shall be the identifier for the service session that has supplied the object descriptor that includes the ES_Descriptor that is currently processed. QosDescriptor shall be the QoS_Descriptor of this ES_Descriptor, direction shall indicate upstream or downstream channels according to the `DecoderConfigDescriptor.upstream` flag. UuDataInBuffer shall contain the URL of this ES_Descriptor. On successful return, channelHandle shall contain a valid, however, not normative handle to the accessible stream.

NOTE 1 — It is a duty of the service to discriminate between the two cases with either ES_ID or URL as parameters to uuDataInBuffer in DA_ChannelAdd.

NOTE 2 — Step 4 in 8.7.3.6.2 and step 3 in 8.7.3.8.2 are currently not mapped to a DAI call in a normative way. It may be implemented using the DA_UserCommand() primitive.

The set up example in the following figure conveys an initial object descriptor that points to one SceneDescriptionStream, an optional ObjectDescriptorStream and additional optional SceneDescriptionStreams or ObjectDescriptorStreams. The first request to the DAI will be a DA_ServiceAttach() with the content address as a parameter. This call will return an initial object descriptor. The ES_IDs in the contained ES_Descriptors will be used as parameters to a DA_ChannelAdd() that will return handles to the corresponding channels.

Additional streams (if any) that are identified when processing the content of the object descriptor stream(s) are subsequently opened using the same procedure. The object descriptor stream is not required to be present if no further audio- or visual streams or inlined scene description streams form part of the content.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000066

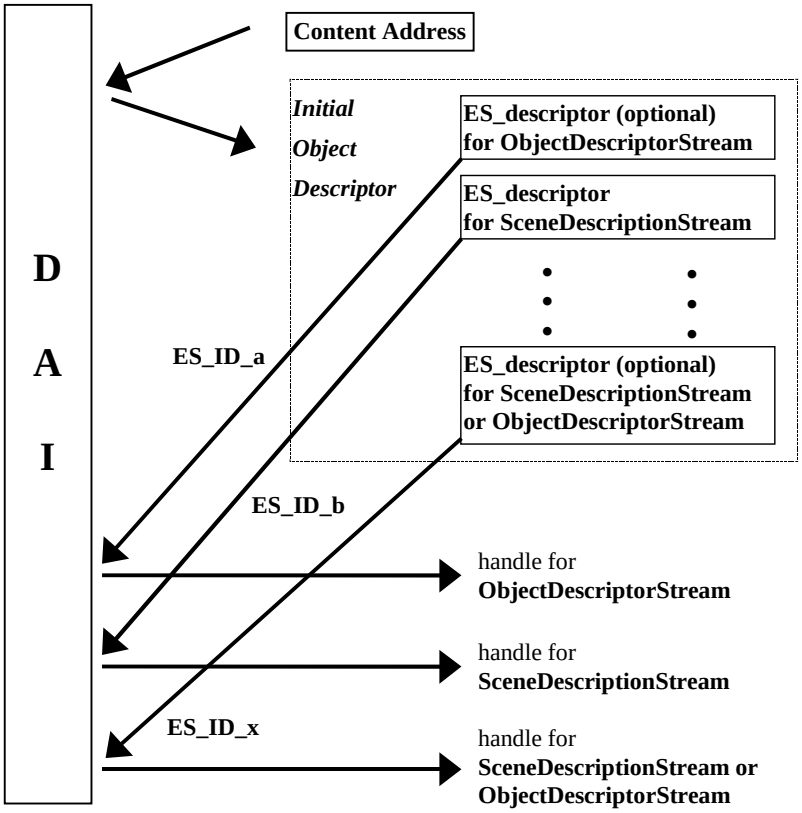


Figure 8 - Requesting stream delivery through the DAI

8.8 Usage of the IPMP System interface

8.8.1 Overview

IPMP elementary streams and descriptors may be used in a variety of ways. For instance, IPMP elementary streams may convey time-variant IPMP information such as keys that change periodically. An IPMP elementary stream may be associated with a given elementary stream or set of elementary streams. Similarly, IPMP descriptors may be used to convey time-invariant or slowly changing IPMP information associated with a given elementary stream or set of elementary streams. This subclause specifies methods how to associate an IPMP system to an elementary stream or a set of elementary streams.

8.8.2 Association of an IPMP System with IS0/IEC 14496 content

8.8.2.1 Association in the initial object descriptor

An IPMP System may be associated with ISO/IEC 14496 content in the initial object descriptor. In that case the initial object descriptor shall aggregate in addition to the ES_Descriptors for scene description and object descriptor streams one or more ES_Descriptors that reference one or more IPMP elementary streams. This implies that all the elementary streams that are described through this initial object descriptor are governed by the one or more IPMP Systems that are identified within the one or more IPMP streams.

8.8.2.2 Association in other object descriptors

An IPMP System may be associated with ISO/IEC 14496 content in an object descriptor in three ways:

In the first case, the object descriptor aggregates in addition to the ES_Descriptors for the content elementary streams one or more ES_Descriptors that reference one or more IPMP elementary streams. This implies that all the content elementary streams described through this object descriptor are governed by the one or more IPMP Systems that are identified within the one or more IPMP streams. Note that an ES_Descriptor that describes an IPMP stream may contain references to IPMP_Descriptors.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000067

The second method is to include one or more IPMP_DescriptorPointers in the object descriptor. This implies that all content elementary streams described by this object descriptor are governed by the IPMP System(s) that is/are identified within the referenced IPMP descriptor(s).

The third method is to include IPMP_DescriptorPointers in the ES_Descriptors embedded in this object descriptor. This implies that the elementary stream referenced by such an ES_Descriptor is controlled by an IPMP System.

8.8.3 IPMP of Object Descriptor streams

Object Descriptor streams shall not be affected by IPMP Systems, i.e., they shall always be available without protection.

An IPMP_Descriptor associated with an object descriptor stream through an IPMP_DescriptorPointer implies that an IPMP System controls all elementary streams that are referred to by this object descriptor stream.

8.8.4 IPMP of Scene Description streams

Scene description streams are treated like any media stream, i.e. they may be managed by an IPMP System.

An IPMP_Descriptor associated with a scene description stream implies that the IPMP System controls this scene description stream.

There are two ways to protect part of a scene description (or to apply different IPMP Systems to different components of a given scene):

The first method exploits the fact that it is permissible to have more than one scene description stream associated with one object descriptor (see 8.7.2.2). Such a split of the scene description can be freely designed by a content author, for example, putting a basic scene description into the first stream and adding one or more additional scene description streams that enhance this basic scene using BIFS updates.

The second method is to structure the scene using one or more **Inline** nodes (see 9.4.2.62). Each **Inline** node refers to one or more additional scene description streams, each of which might use a different IPMP System.

8.8.5 Usage of URLs in managed and protected content

8.8.5.1 URLs in the BIFS Scene Description

ISO/IEC 14496 does not specify compliance points for content that uses BIFS URLs that do not point to an object descriptor. Equally, no normative way to apply an IPMP System to such links exists. The behavior of an IPMP-enabled terminal that encounters such links is undefined.

8.8.5.2 URLs in Object Descriptors

URLs in object descriptors point to other remote object descriptors. This merely constitutes an indirection and should not adversely affect the behavior of the IPMP System that might be invoked through this remote object descriptor.

NOTE — The only difference is that while the original site might be trusted, the referred one might not. Further corrective actions to guard against this condition are not in the scope of ISO/IEC 14496.

8.8.5.3 URLs in ES_Descriptors

URLs in ES descriptors are used to access elementary streams remotely. This merely constitutes an indirection and therefore does not adversely affect the behavior of the IPMP System that might be invoked through this remote object descriptor.

NOTE — The only difference is that while the original site might be trusted, the referred one might not. Further corrective actions to guard against this condition are not in the scope of ISO/IEC 14496.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000068

8.8.6 IPMP Decoding Process

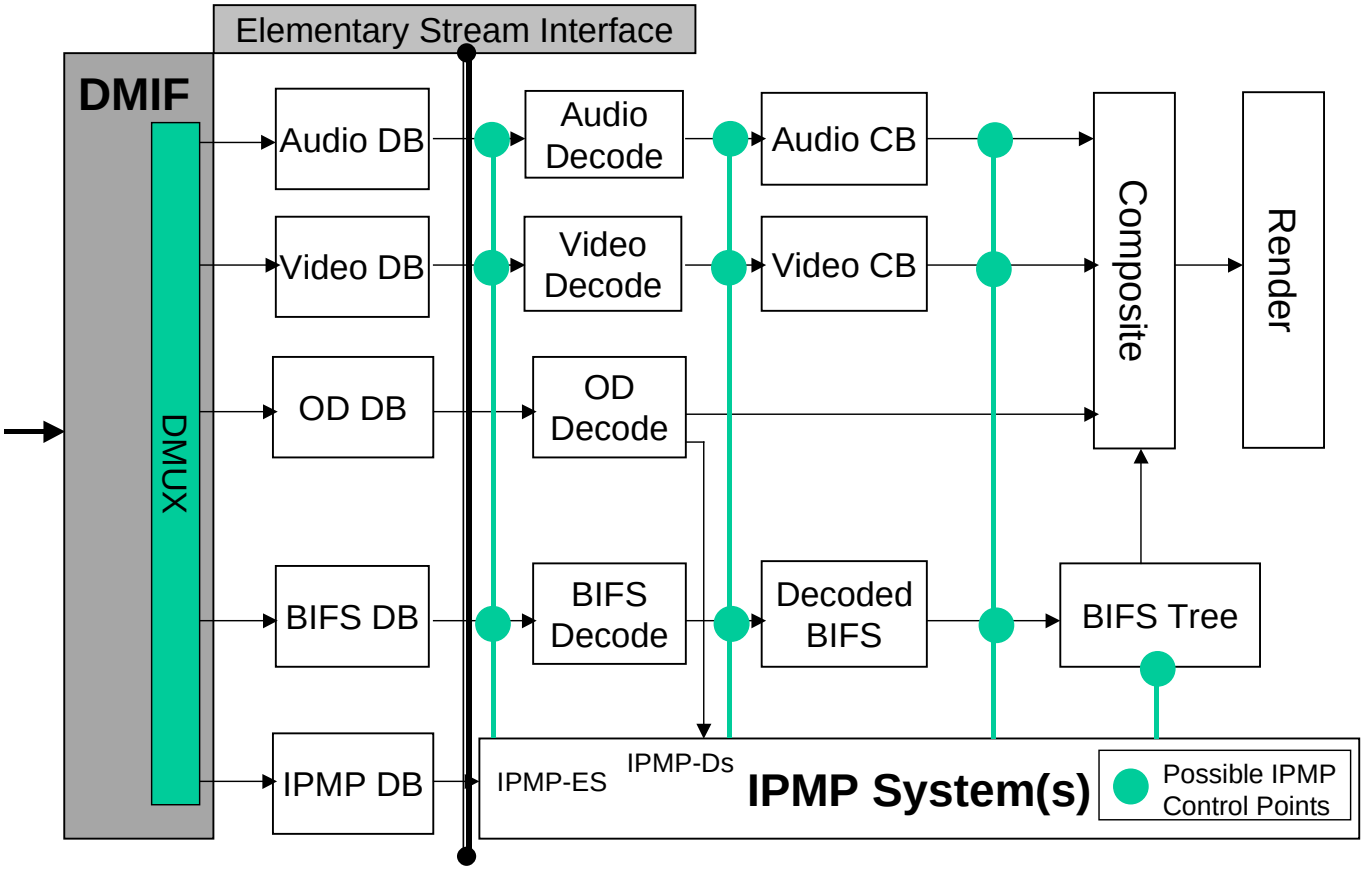


Figure 9 - IPMP system in the ISO/IEC 14496 terminal architecture

Figure 9 depicts the injection of an IPMP System with respect to the MPEG-4 terminal. IPMP System specific data is supplied to the IPMP System via IPMP streams and/or IPMP descriptors, and the IPMP system releases protected content *after the sync layer.*

Each elementary stream under the control of an IPMP System has the conceptual element of a *stream flow controller.* Stream flow control can take place between the the SyncLayer decoder and the decoder buffer. As the figure indicates, elements of IPMP control may take place at other points in the terminal including, after decoding (as with some watermarking systems) or in the decoded BIFS stream, or after the composition buffers have been written, or in the BIFS scene tree. Stream flow controllers either enable or disable processing of an elementary stream in a non-normative way that depends on the status information provided by the IPMP System.

Finally, the IPMP System must at a minimum:

1. Process the IPMP stream and descriptor

2. Appropriately manage (e.g. decrypt and release) protected elementary streams.

The initialization process of the IPMP System is not specified except that it shall not unduly delay the content access process as specified in 8.7.3.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000069

9 Scene Description

9.1 Introduction

9.1.1 Scope

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 10. 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.

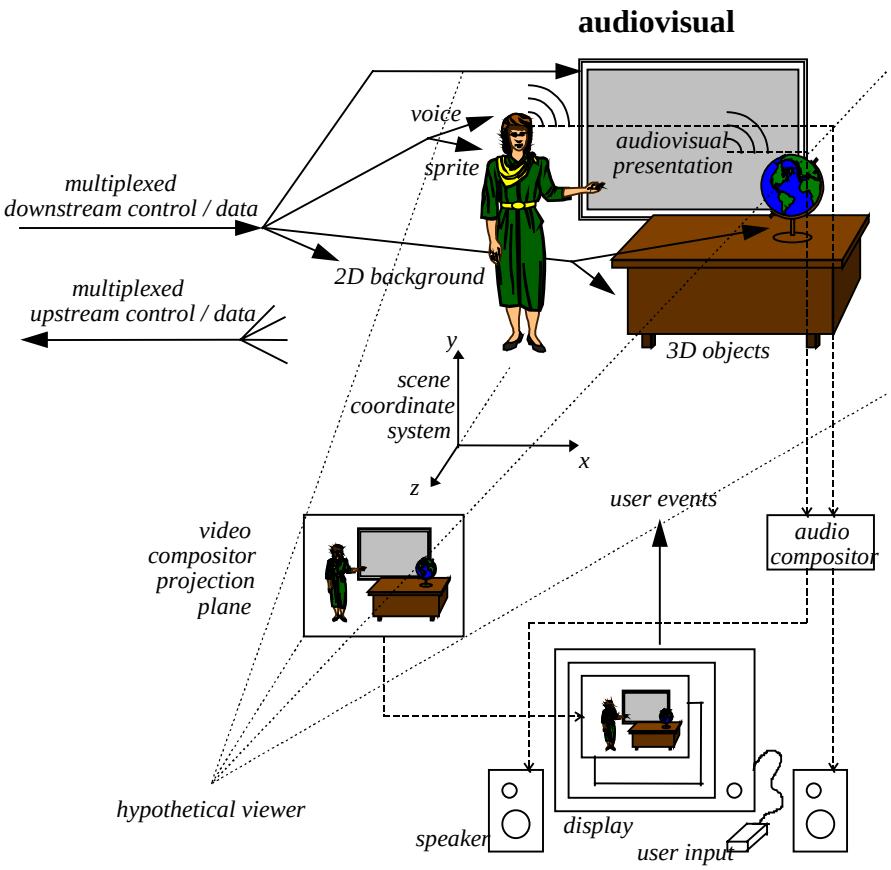


Figure 10 - An example of an object-based multimedia scene

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 ISO/IEC 14496-1 is based largely on ISO/IEC 14772-1:1998 (Virtual Reality Modeling Language – VRML).

9.1.2 Composition and Rendering

ISO/IEC 14496-1 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-dependent; for audio data, subclause 9.2.2.13 and the semantics of the AudioBIFS nodes

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000070

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).

9.1.3 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 14496-1.

9.1.3.1 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 11. The structure is not necessarily static; nodes may be added, deleted or be modified.

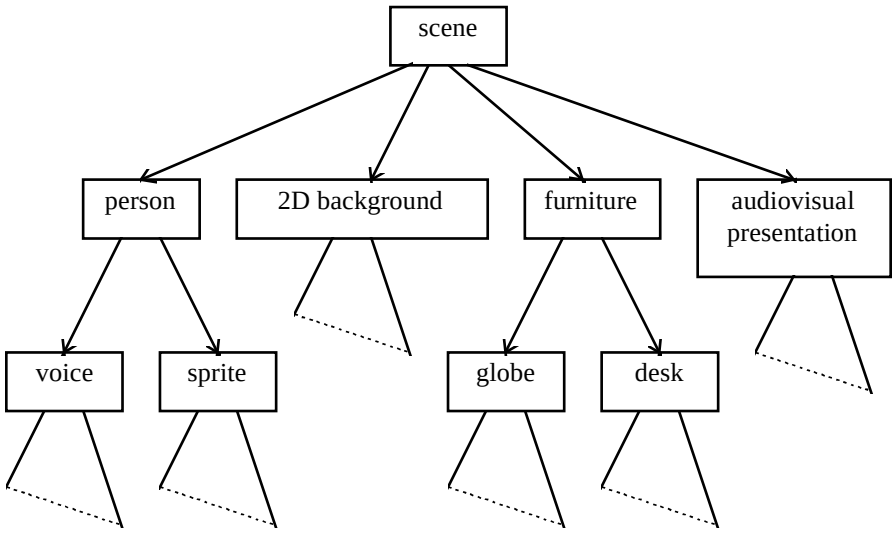


Figure 11 - Logical structure of example scene

9.1.3.2 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.

9.1.3.3 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.

9.1.3.4 Behavior of audio-visual objects

ISO/IEC 14496-1 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 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000071

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 14496-1.

9.2 Concepts

9.2.1 BIFS Elementary Streams

9.2.1.1 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 clause 7). 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.

9.2.1.2 BIFS Decoder Configuration

BIFS configuration information is contained in a `BIFSConfig` (see 9.3.5.2) syntax structure, which is transmitted as `DecoderSpecificInfo` for the BIFS elementary stream in the corresponding object descriptor (see 8.6.7). 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.

9.2.1.3 BIFS Access Units

A BIFS data access unit consists of one BIFS `CommandFrame` or `AnimationFrame`, as defined in 9.3.6.2 and 9.3.8.2, 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 10.2.4). 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'.

9.2.1.4 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 10.2.4) for this stream or by indicating the elementary stream from which this BIFS stream inherits the time base (see 10.2.3). All time stamps in the SL-packetized BIFS stream refer to this time base.

9.2.1.5 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 9.4.2.62) 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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000072

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 8.7.2.2). In this case, these BIFS streams share the 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.

9.2.1.6 Time

9.2.1.6.1 Time-dependent nodes

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 media and do not affect the delivery of the stream attached to the time dependent node. ISO/IEC 14496-1 has the following time-dependent nodes: **AnimationStream**, **AudioBuffer**, **AudioClip**, **AudioSource**, **MovieTexture** and **TimeSensor**.

9.2.1.6.2 Time fields in BIFS nodes

Several BIFS nodes have fields of type SFTime that identify a point in time at which an event occurs (change of a parameter value, start of a media stream, etc). Depending on the individual field semantics, these fields may contain time values that refer either to an absolute position on the time line of the BIFS stream or that define a time duration.

As defined in 9.2.1.4, the speed of the flow of time for events in a BIFS stream is determined by the time base of the BIFS stream. This determines unambiguously durations expressed by relative SFTime values like the **cycleTime** field of the **TimeSensor** node.

The semantics of some SFTime fields is such that the time values shall represent an absolute position on the time line of the BIFS stream (e.g. **startTime** in **MovieTexture**). 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 SFTime field in such a node is updated through a `CommandFrame` in a BIFS access unit (see 9.2.1.3). The value in the SFTime field as coded in the delivered BIFS command is the positive offset from this point in time in seconds. The absolute position on the time line shall therefore be calculated as the sum of the composition time of the BIFS access unit and the value of the SFTime field.

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.

NOTE 2 — The SFTime fields that define the start or stop of a media stream are relative to the BIFS time base. If the time base of the media stream is a different one, it is not generally possible to set a **startTime** that corresponds exactly to the composition time of a composition unit of this media stream.

EXAMPLE — The example in Figure 12 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000073

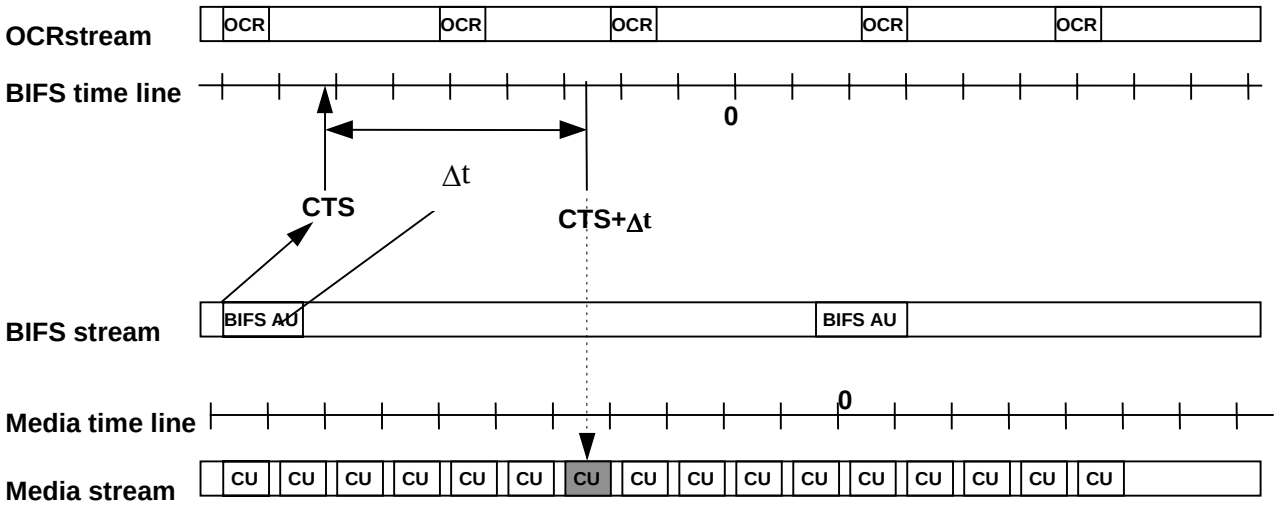


Figure 12 - Media start times and CTS

9.2.2 BIFS Scene Graph

9.2.2.1 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
- 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 13 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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000074

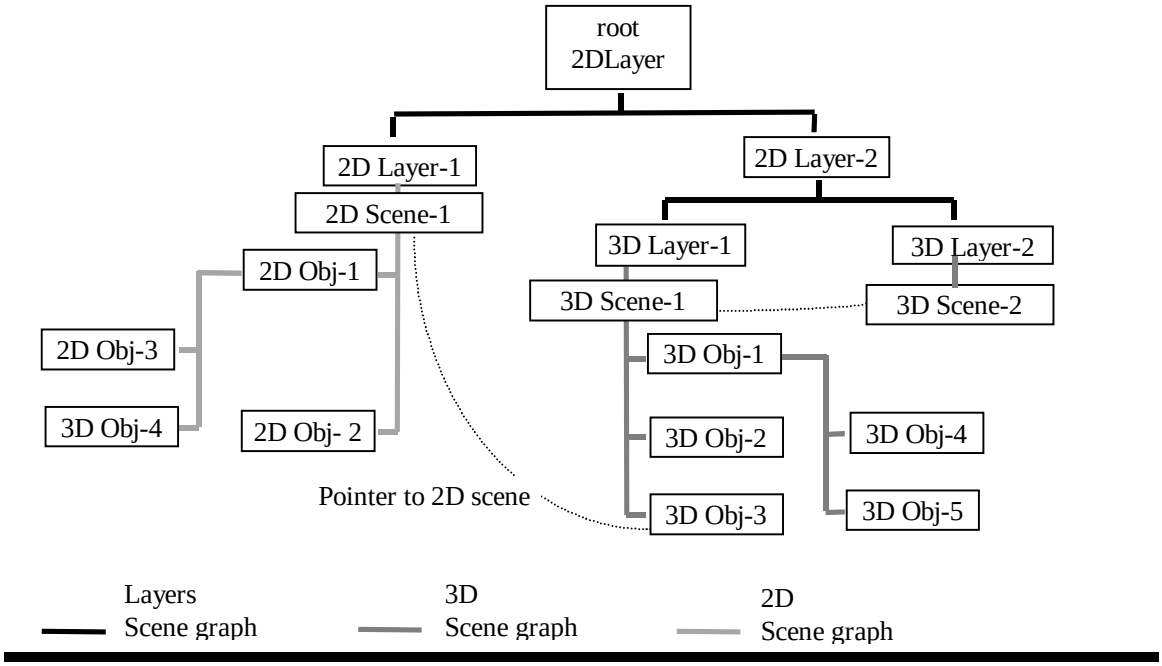


Figure 13 - Scene graph example.

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.

9.2.2.2 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 14). 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 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000075

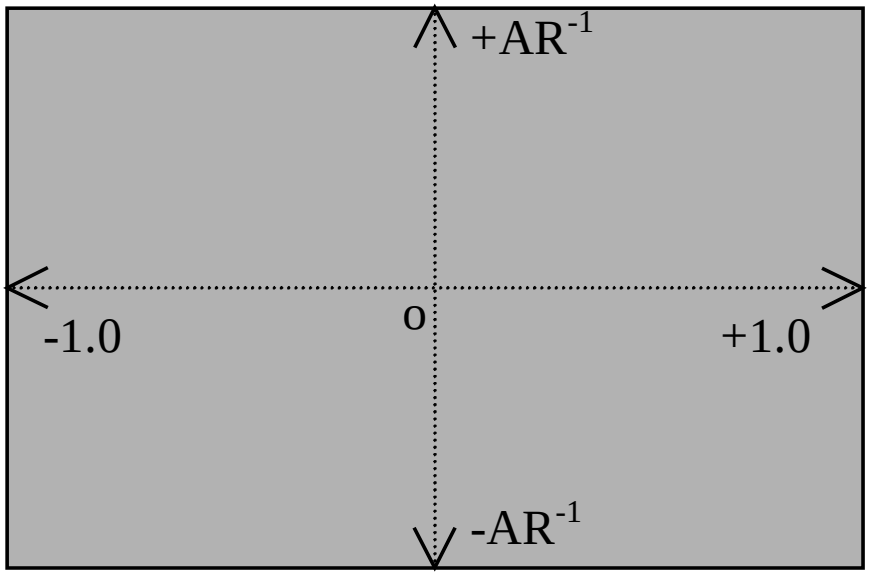


Figure 14 - 2D co-ordinate system (AR = Aspect Ratio)

9.2.2.3 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.

9.2.2.4 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.

9.2.2.5 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 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000076

9.2.2.6 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 9.3.5.2).

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.

9.2.2.7 Nodes and fields

9.2.2.7.1 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.

9.2.2.7.2 Fields and Events

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

9.2.2.8 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 9.2.2.7.2). 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 9.4). 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 9.4. The binary syntax which represents the binary format as transported in streams conforming to ISO/IEC 14496-1 is provided in 9.3 and uses the node coding parameters provided in Annex H.

9.2.2.8.1 Binary Syntax Overview

9.2.2.8.1.1 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000077

9.2.2.8.1.2 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.

9.2.2.8.1.3 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.

9.2.2.8.1.4 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.

9.2.2.9 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.

9.2.2.9.1 Numerical data and string data types

9.2.2.9.1.1 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.

9.2.2.9.1.2 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

if x < min, x := min

9.2.2.9.1.3 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.

9.2.2.9.2 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.

9.2.2.10 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 9.3.5.2).

The following restrictions apply:

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000078

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 9.2.1.5.

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` (see 9.2.2.10).

9.2.2.11 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 14 - Standard units

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

9.2.2.12 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 9.4.2.19) 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.

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

9.2.2.13 Special considerations for audio

9.2.2.13.1 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** and **DirectiveSound** nodes are used to attach audio to 3D scene graphs and the **Sound2D** node is used to attach audio to 2D scene graphs. 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000079

The nodes below the **Sound/DirectiveSound/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**, **AudioDelay**) 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/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.

9.2.2.13.2 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/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/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 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.

9.2.2.13.2.1 Sample-rate conversion

If the various children of a **Sound/ DirectiveSound/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`.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000080

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.

9.2.2.13.2.2 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:

a) 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.

b) 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.

9.2.2.13.3 Audio-specific BIFS Nodes

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

Table 15 - Audio-Specific BIFS Nodes

Node	Purpose	Subclause
AudioBuffer	Interactively trigger snippets of sound	9.4.2.6
AudioClip	Insert an audio clip into a scene	9.4.2.8
AudioDelay	Add delay to sound	9.4.2.9
AudioMix	Mix sounds	9.4.2.11
AudioSource	Define audio source input to a scene	9.4.2.12
AudioFX	Apply post-production effects to sound	9.4.2.10
AudioSwitch	Switching of audio sources in a scene	9.4.2.13
ListeningPoint	Define listening point in a scene	9.4.2.67
Sound, Sound2D, DirectiveSound	Define properties of sound	9.4.2.94, 9.4.2.95, 9.4.2.39

9.2.2.13.4. 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000081

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 16 - 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	**9.4.2.2**
AcousticMaterial	Define sound reflectivity and transmission properties (along with the visual properties) for each acoustically relevant (flat, polygonal) surface.	physical	**9.4.2.1**
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	**9.4.2.39**
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	**9.4.2.78**

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

9.2.2.13.4.1 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

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000082

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.

9.2.2.13.4.2 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.

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.

9.2.3 Sources of modification to the scene

9.2.3.1 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.

9.2.3.1.1 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 9.2.1.5. The following restrictions apply:

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

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000083

b) The scope of ROUTEIDs is given in 9.2.1.5.

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 9.2.2.10). The USE mechanism shall not be used with routes.

9.2.3.1.2 Conditional node

The **Conditional** node (see 9.4.2.30) 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.

9.2.3.2 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.

9.2.3.2.1 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 9.3.6.2) 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

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 9.2.1.5). The identification of a node's fields is provided by sending the INid of the field (see Annex H).

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000084

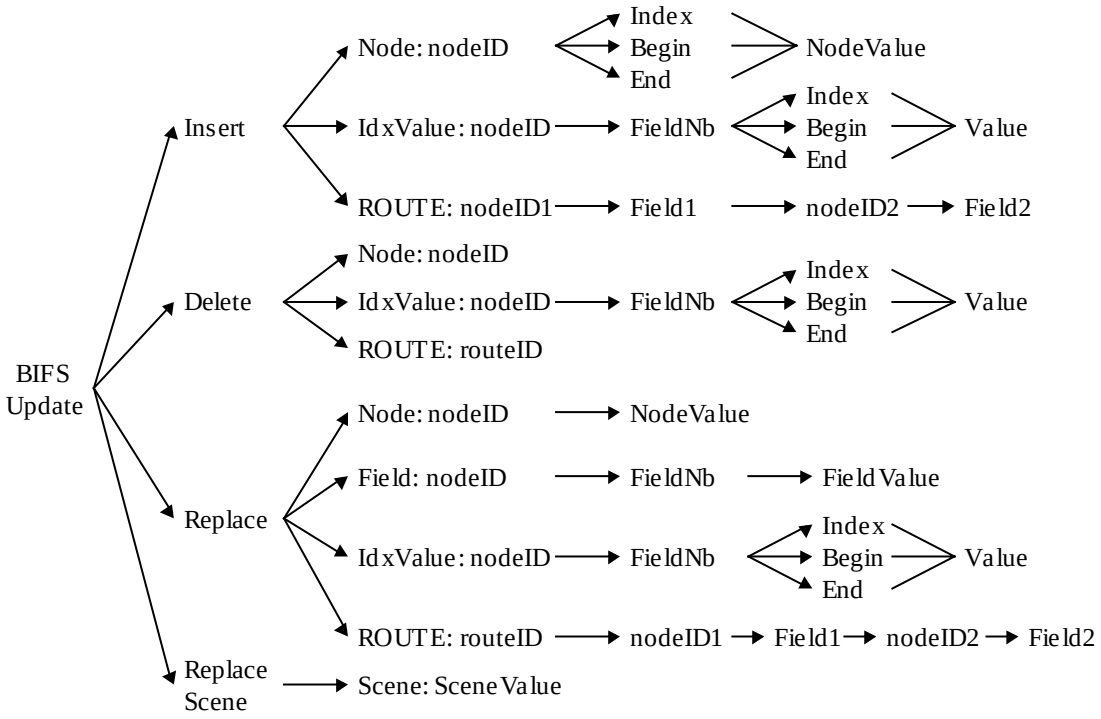


Figure 15 - BIFS-Command Types

9.2.3.2.2 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.

9.2.3.2.3 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 10.2).

9.2.3.3 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 9) 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.

9.2.3.3.1 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 8.6.7).

9.2.3.3.2 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 Annex H and 9.2.3.3.3) can be modified using BIFS-Anim. Such fields are called dynamic fields. In addition, the

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000085

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.

9.2.3.3.3 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 Annex H).

NOTE — Some node tables in Annex H 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

9.2.3.4 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

9.3 BIFS Syntax

9.3.1 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 9.3.5.2 and 9.3.5.3. 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 9.3.6.

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 0.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000086

9.3.2 Decoding tables, data structures and associated functions

9.3.2.1 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).

9.3.2.2 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 Annex H 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 tables in H.2., and extended node data types in Annex H.4. 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 Annex H.2.) 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 9.3.7.2).

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 extended node data types in Annex H.4. Since that NDT exists in tables in Annex H.2. (where the nodes of that data type are encoded with six bits), this node is completely encoded with 9 bits as: 0b000000010.

9.3.2.3 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.

`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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000087

`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.

9.3.2.4 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 9.3.5.2 and 9.3.5.3).

```
Class BIFSConfiguration{
   int nodeIDbits;          The number of bits used to encode the nodeIDs.
   int routeIDbit;          The number of bits used to encode the routeIDs.
   int PROTOIDbits;         The number of bits used to encode the PROTO. This value is in used only
                            if the data for the structure was transmitted by BIFSv2Config
   boolean randomAccess;    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.
   AnimationMask animMask;  The AnimationMask used for BIFS-Anim
}
```

9.3.2.5 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;                      The number of nodes to be animated
   NodeData animNode[numNodes];       The array of animated nodes.
   boolean isIntra;                   The status of the current frame: intra if isIntra is true,
                                      predictive otherwise.
   boolean isActive[numNodes];        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.
}
```

9.3.2.6 NodeData

This data structure is built to decode the relevant information for one node. It is created from the node coding tables in Annex H. The following functions support relevant operations on this data structure:

```
NodeData MakeNode(int nodeType)
```

This function creates a `NodeData` structure from the node coding table matching the given `nodeType`.

```
NodeData GetNodeFromID (int nodeID)
```

This function returns the `NodeData` structure matching the given `nodeID`.

```
class NodeData {
   int nodeType;            The nodeType of the node.
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000088

`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 2nd column of the node coding tables).
`int nINbits;`	The number of bits used for “in” field codes (the width of the codewords in the 3rd column of the node coding tables).
`int nOUTbits;`	The number of bits used for “out” field codes (the width of the codewords in the 4th 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 useQuant 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.

`}`

9.3.2.7 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 Annex H).

`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.
`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 19 in 9.3.3.1.1.
`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 25 in 9.3.3.2.1.
`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 9.3.2.9.
`AnimFieldQP aqp;`	This field is only used in BIFS-Anim. It references an `AnimFieldQP` stucture described in 9.3.2.10.
`QuantizationParameter lqp;`	This field points to the local **QuantizationParameter**

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000089

	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 9.3.2.9) 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 9.3.2.9) 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 9.3.2.9) 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)`
Returns 1 if the field's `fieldType` corresponds to a single field and 0 otherwise.

`int getNbComp(FieldData field)`
Returns the number of quantized components for the field as given below:

Table 17 - 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

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 9.3.3.3).

9.3.2.8 Node Data Type Table Parameters

The following functions provide access to the node data type tables (described in Annex H):

`int GetNodeType(int nodeDataType, int localNodeType)`
Returns the `nodeType` of the node indexed by `localNodeType` in the node data type table. The `nodeType` of a node is its index in the `SFWorldNode` NDT Table.

`int GetNDTnbBits(int nodeDataType)`
Returns the number of bits used to index the nodes of the matching node data type table (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 9.4 and ISO/IEC 14772-1:1998 , Table 4.3).

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000090

9.3.2.9 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 table (Annex H).

```
Class FieldCodingTable {
   float floatMin[];
   float floatMax[];
   float intMin[];
   float intMax[];
   int defaultNbBits;
}
```

`floatMin[]` The minimum default bounds for fields of type SFFloat, SFVec2f and SFVec3f. These values are obtained from the "[m, M]" column of the node coding table.

`floatMax[]` The minimum default bounds for fields of type SFFloat, SFVec2f and SFVec3f. These values are obtained from the "[m, M]" column of the node coding table.

`intMin[]` The minimum default bounds for fields of type SFInt32. These values are obtained from the "[m, M]" column of the node coding table.

`intMax[]` The minimum default bounds for fields of type SFInt32. These values are obtained from the "[m, M]" column of the node coding table.

`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).

9.3.2.10 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;
   boolean useDefault;
   boolean isTotal;
   int numElement;
   int indexList[];
   float[] Imin;
   float[] Imax;
   int[] IminInt;
   int[] Pmin;
   int INbBits;
   int PNbBits;
```

`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 25 in 9.3.3.2.1.

`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.

`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.

`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.

`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}.

`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`.

`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`.

`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.

`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`.

`INbBits` The number of bits used in intra mode for the field. This value is obtained from the `InitialAnimQP` or `AnimQP`.

`PNbBits` The number of bits used in predictive mode for the field. This value is

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000091

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(AnimFieldQP aqp)`
Returns the number of set of bounds matching the animation type (see 9.3.2.3), as follows :

Table 18 - Return values of getNbBounds

aqp.animType	value returned
4,6,7,8 9,10 11,12,13	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.

9.3.3 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.

9.3.3.1 Quantization of BIFS scenes

9.3.3.1.1 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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000092

Table 19 - 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	Reserved

Each field that may be quantized is assigned to one of the quantization categories (see Annex H). Along with quantization parameters, minimum and maximum values are specified for each field of each node.

9.3.3.1.2 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, SFVec2f or SFVec3f.
- The quantization category of the field is not 0.
- The node to which the field belongs has a **QuantizationParameter** (see 9.4.2.89) node in its context
- 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:

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000093

Table 20 - 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	Always TRUE

`nbBits`

In the BIFS scene quantization process, `nbBits` is set in the following way :

Table 21 - 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: $\text{nbBits} = \text{Ceil}(\log_2(N))$ where the function Ceil returns the smallest integer greater than its argument
15	0

`floatMin[]`

In the BIFS scene quantization process, `floatMin` is set in the following way:

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000094

Table 22 - Value of `floatMin`, depending on `quantType` and `fieldType`

quantType	fieldType	floatMin
1	SFVec3fType	lqp.position3Dmin
2	SFVec2fType	lqp.position2Dmin
3	SFFloatType	max(fct.min[0],lqp.drawOrderMin)
4	SFFloatType	lqp.colorMin
	SFColorType	lqp.colorMin, lqp.colorMin, lqp.colorMin
5	SFVec2fType	lqp.textureCoordinateMin
6	SFFloatType	Max(fct.min[0],lqp.angleMin)
7	SFFloatType	lqp.scaleMin
	SFVec2fType	lqp.scaleMin, lqp.scaleMin
	SFVec3fType	lqp.scaleMin, lqp.scaleMin, lqp.scaleMin
8	SFFloatType	Max(fct.min[0],lqp.keyMin)
9	SFVec3fType	0.0
10	SFRotationTyp e	0.0
11,12	SFFloatType	lqp.sizeMin
	SFVec2fType	lqp.sizeMin, lqp.sizeMin
	SFVec3fType	lqp.sizeMin, lqp.sizeMin, lqp.sizeMin
13,14,15		NULL

`floatMax[]`
In the BIFS scene quantization process, `floatMax` is set in the following way:

Table 23 - Value of `floatMax`, depending on `quantType` and `fieldType`

quantType	fieldType	floatMax
1	SFVec3fType	lqp.position3Dmax
2	SFVec2fType	lqp.position2Dmax
3	SFFloatType	min(fct.max[0],lqp.drawOrderMax)
4	SFFloatType	lqp.colorMax
	SFColorType	lqp.colorMax, lqp.colorMax, lqp.colorMax
5	SFVec2fType	lqp.textureCoordinateMax
6	SFFloatType	min(fct.max[0],lqp.angleMax)
7	SFFloatType	lqp.scaleMax
	SFVec2fType	lqp.scaleMax, lqp.scaleMax
	SFVec3fType	lqp.scaleMax, lqp.scaleMax, lqp.scaleMax
8	SFFloatType	min(fct.max[0],lqp.keyMax)
9	SFVec3fType	1.0
10	SFRotationType	1.0
11,12	SFFloatType	lqp.sizeMax
	SFVec2fType	lqp.sizeMax, lqp.sizeMax
	SFVec3fType	lqp.sizeMax, lqp.sizeMax, lqp.sizeMax
13,14,15		NULL

`intMin[]`

In the BIFS scene quantization process, `intMin` is set in the following way:

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000095

Table 24 - 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`

9.3.3.2 Quantization of BIFS-Anim

9.3.3.2.1 Animation Categories

The fields are grouped in the following categories for animation:

Table 25 - 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

9.3.3.2.2 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 26 - 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 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000096

Table 27 - Value of `floatMin`, depending on `animType`

animType		aqp.useDefault	floatMin
4	Color	true	fct.min[0], fct.min[0], fct.min[0]
		false	aqp.IMin[0], aqp.IMin[0], aqp.IMin[0]
8	BoundFloat	true	fct.min[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]
12	Size 2D	false	aqp.IMin[0], 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 28 - Value of `floatMax`, depending on `animType`

animType		aqp.useDefault	floatMax
4	Color	true	fct.max[0], fct.max[0], fct.max[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]
12	Size 2D	false	aqp.IMax[0], 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 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000097

Table 29 - 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

9.3.3.3 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:

$$v(t) = \text{InvQuant}\left(v_q(t)\right)$$

The linear quantization and inverse quantization are:

```
int quantize (float Vmin, float Vmax, float v, int Nb)
```

which returns $$v_q = \frac{v - V_{\min}}{V_{\max} - V_{\min}}(2^{Nb} - 1)$$

```
float invQuantize (float Vmin,float Vmax,int vq, int Nb)
```

which returns $$\hat{v} = V_{\min} + v_q \frac{V_{\max} - V_{\min}}{2^{\max(Nb,1)} - 1}$$

If `isQuantized` is true, the quantization/inverse quantization process is the following :

Table 30 - Quantization and inverse quantization process

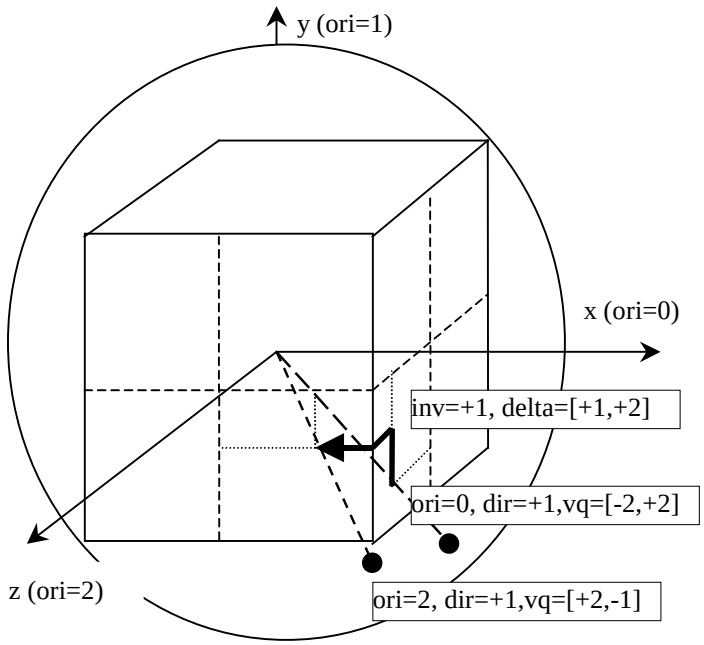
`QuantType`	`animType`	Quantization/Inverse Quantization Process
1,2,3,4,5 6,7,8 11,12	1,2,4 6,7,8 11,12	For each component of the vector, the float quantization is applied: $v_q[i] = \text{quantize}(\text{floatMin}[i], \text{floatMax}[i], \text{v}[i], \text{nbBits})$ For the inverse quantization: $\hat{v}[i] = \text{invQ uantize}(\text{floatMin}[i], \text{floatMax}[i], \text{v}_\text{q}[i], \text{nbBits})$
9,10	9,10	For normals and rotations, the quantization method is as follows. Normals are first renormalized : $v[0] = \frac{n_x}{\sqrt{{n_x}^2 + {n_u}^2 + {n_z}^2}}, \quad v[1] = \frac{n_y}{\sqrt{{n_x}^2 + {n_u}^2 + {n_z}^2}}, \quad v[2] = \frac{n_z}{\sqrt{{n_x}^2 + {n_u}^2 + {n_z}^2}}$ Rotations (axis $\vec{n}$, angle α) are first written as quaternions : $v[0] = \cos(\frac{\alpha}{2}) \quad v[1] = \frac{n_x}{\|\vec{n}\|}.\sin(\frac{\alpha}{2}) \quad v[2] = \frac{n_y}{\|\vec{n}\|}.\sin(\frac{\alpha}{2}) \quad v[3] = \frac{n_z}{\|\vec{n}\|}.\sin(\frac{\alpha}{2})$ 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 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000098

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 : $$v_c[i] = \frac{4}{\pi} \tan^{-1}\left(\frac{v\left[(i+k+1) \bmod (N+1)\right]}{v[k]} \right) \quad i = 0,..., N$$ 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 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000099

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 9.3.4) 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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000100

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.

9.3.4 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 9.3.3.3).

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 0.

The result v_q^2 is then computed in the following way :

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000101

Table 31 - Compensation process

<table>
<tr><th>animType</th><th colspan="3">Compensation Process</th></tr>
<tr><td>1,2,4,6,7,8

11,12,13</td><td colspan="3">The components of v_q^2 are:
vq2[i] = vq1[i] + vDelta[i]</td></tr>
<tr><td>9,10</td><td colspan="3">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>For every index I,

$|vqTemp[i]| \le scale$</td><td colspan="2">v_q^2 is defined by,

vq2[i] = vqTemp[i]

orientation2= orientation1

direction2 = direction1 * inverse</td></tr>
<tr><td></td><td>There is one and only one index k such that

$|vqTemp[k]| > scale$</td><td colspan="2">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>There are several indices k such that

$|vqTemp[k]| > scale$</td><td colspan="2">The result is undefined</td></tr>
</table>

9.3.5 BIFS Configuration

9.3.5.1 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 8.6.7), 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.

The terminal configuration is defined differently for elementary streams compliant only with this part of ISO/IEC 14496-1 and those compliant with this specification, and it is presented in 9.3.5.2 and 9.3.5.3, respectively. The

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000102

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.

9.3.5.2 BIFSConfig

9.3.5.2.1 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();
   }
}
```

9.3.5.2.2 Semantics

`BIFSConfig` is the terminal configuration for the BIFS elementary stream. It is encapsulated within the `specificInfo` fields of the general `DecoderSpecificInfo` structure (see 8.6.7), 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. 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 0 for detailed semantics.
- The `AnimationMask` specifies the animation parameters of the BIFS-Anim elementary stream.

9.3.5.3. BIFSv2Config

9.3.5.3.1. Syntax

```
class BIFSv2Config {
   bit(1) use3DMeshCoding;
   bit(1) reserved;
   bit(5) nodeIDbits;
   bit(5) routeIDbits;
   bit(5) PROTOIDbits;
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000103

```
   bit(1) isCommandStream;
   if(isCommandStream) {
      bit(1) pixelMetric;
      bit(1) hasSize;
      if(hasSize) {
         int(16) pixelWidth;
         int(16) pixelHeight;
      }
   }
   else {
      bit(1)       randomAccess;
      AnimationMask   animMask();
   }
}
```

9.3.5.3.2. Semantics

BIFSv2Config is the terminal configuration for elementary streams compliant with this specifcation but not with this part of ISO/IEC 14496. It is not compatible with BIFSConfig defined in 9.3.5.2. It is encapsulated within the specificInfo fields of the general DecoderSpecificInfo structure (see 8.6.6), which is contained in the DecoderConfigDescriptor that is carried in ES_Descriptors.

The use3DmeshCoding flag is used to signal that the syntax of 3D Mesh as sepecified by ISO/IEC 14496-2:1999 is used to encode IndexedFaceSet nodes.

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.

9.3.5.4 AnimationMask

9.3.5.4.1 Syntax

```
class AnimationMask() {
   int numNodes = 0;
   do {
      ElementaryMask elemMask();
      numNodes++;
        bit(1) moreMasks;
   } while (moreMasks);
}
```

9.3.5.4.2 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.

9.3.5.5 Elementary mask

9.3.5.5.1 Syntax

```
Class ElementaryMask() {
   bit(BIFSConfiguration.nodeIDbits) nodeID;
      NodeUpdateField node = GetNodeFromID(nodeID);
   switch (node.nodeType) {
      case FaceType:
      break;
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000104

```
      case BodyType:
      break;
      case IndexedFaceSet2DType:
      break;
      default:
      InitialFieldsMask initMask(node);
   }
}
```

9.3.5.5.2 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.

9.3.5.6 InitialFieldsMask

9.3.5.6.1 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);
      }
   }
}
```

9.3.5.6.2 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.

9.3.5.7 InitialAnimQP

9.3.5.7.1 Syntax

```
InitialAnimQP(animFieldQP aqp) {
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000105

```
   aqp.useDefault=FALSE;
   uint(4) type;
   aqp.animType = type;
   switch(aqp.animType) {

        case  4:          // Color
        case  8:          // BoundFloats
          bit(1)    aqp.useDefault
        case  1:          // Position 3D
        case  2:          // Position 2D
        case 11:          // Size 3D
        case 12:          // Size 2D
        case  7:          // Floats
          if (!aqp.useDefault) {
            for (i=0;i<getNbBounds(aqp);i++) {
              bit(1)       useEfficientCoding
              GenericFloat aqp.Imin[i](useEfficientCoding);
            }
            for (i=0;i<getNbBounds(aqp);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);i++) {
        int(INbBits+1) vq
        aqp.Pmin[i] = vq-2^aqp.INbBits;
   }

   unsigned int(4)      aqp.PNbBits;

}
```

9.3.5.7.2 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 9.3.3.3) 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 9.3.4) is used in predictive mode only.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000106

9.3.6 BIFS Command Syntax

9.3.6.1 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.

9.3.6.2 Command Frame

9.3.6.2.1 Syntax

```
class CommandFrame() {
   do {
      Command command();
      bit(1) continue;
   } while (continue);
}
```

9.3.6.2.2 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.

9.3.6.3 Command

9.3.6.3.1 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;
}
```

9.3.6.3.2 Semantics

For each `Command`, the 2-bit flag, `code`, signals one of the four basic commands: insertion, deletion, replacement, and scene replacement.

9.3.6.4 Insertion Command

9.3.6.4.1 Syntax

```
class InsertionCommand() {
   bit(2) parameterType ;
   switch parameterType {
   case 0:
      NodeInsertion nodeInsert();
      break;
   case 2:
      IndexedValueInsertion idxInsert();
      break;
   case 3:
      ROUTEInsertion ROUTEInsert();
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000107

```
      break ;
   }
}
```

9.3.6.4.2 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 2, an `IndexedValueInsertion` is expected.

If `parameterType` is 3, a `ROUTEInsertion` is expected.

9.3.6.5 Node Insertion

9.3.6.5.1 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;
   }
}
```

9.3.6.5.2 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.

9.3.6.6 IndexedValue Insertion

9.3.6.6.1 Syntax

```
class IndexedValueInsertion() {
   bit(BIFSConfiguration.nodeIDbits) nodeID;
   NodeUpdateField node=GetNodeFromID(nodeID);
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000108

```
   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;
   }
}
```

9.3.6.6.2 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`.

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.

9.3.6.7 ROUTE Insertion

9.3.6.7.1 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;
}
```

9.3.6.7.2 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000109

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.

9.3.6.8 Deletion Command

9.3.6.8.1 Syntax

```
class DeletionCommand() {
   bit(2) parameterType ;
   switch (parameterType) {
   case 0:
      NodeDeletion nodeDelete();
      break ;

   case 2:
      IndexedValueDeletion idxDelete();
      break ;

   case 3:
      ROUTEDeletion ROUTEDelete();
      break ;
   }
}
```

9.3.6.8.2 Semantics

There are three types of deletion commands, signalled by the 2-bit flag parameterType.

If parameterType is 0, a NodeDeletion is expected.

If parameterType is 2, an IndexedValueDeletion is expected.

If parameterType is 3, a ROUTEDeletion is expected.

9.3.6.9 Node Deletion

9.3.6.9.1 Syntax

```
class NodeDeletion() {
   bit(BIFSConfiguration.nodeIDbits) nodeID;
}
```

9.3.6.9.2 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.

9.3.6.10 IndexedValue Deletion

9.3.6.10.1 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;
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000110

```
   case 2:    // deletion at the beginning of the field
      break;
   case 3:    // deletion at the end of the field
      break;
   }
}
```

9.3.6.10.2 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.

9.3.6.11 ROUTE Deletion

9.3.6.11.1 Syntax

```
class ROUTEDeletion() {
   bit(BIFSConfiguration.routeIDbits) routeID;
}
```

9.3.6.11.2 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.

9.3.6.12 Replacement Command

9.3.6.12.1 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;
   }
}
```

9.3.6.12.2 Semantics

There are 4 replacement commands, signalled by the 2-bit flag `parameterType`.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000111

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.

9.3.6.13 Node Replacement

9.3.6.13.1 Syntax

```
class NodeReplacement() {
   bit(BIFSConfiguration.nodeIDbits) nodeID;
   SFNode node(SFWorldNode);
}
```

9.3.6.13.2 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.

9.3.6.14 Field Replacement

9.3.6.14.1 Syntax

```
class FieldReplacement() {
   bit(BIFSConfiguration.nodeIDbits) nodeID ;
   NodeData node = GetNodeFromID(nodeID);
   int(node.nINbits) inID;
   Field value(node.field[node.in2all[inID]]);
}
```

9.3.6.14.2 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

9.3.6.15 IndexedValueReplacement

9.3.6.15.1 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
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000112

```
      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;
   }
}
```

9.3.6.15.2 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.

9.3.6.16 ROUTE Replacement

9.3.6.16.1 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;
}
```

9.3.6.16.2 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.

9.3.6.17 Scene ReplaceCommand

9.3.6.17.1 Syntax

```
class SceneReplaceCommand() {
   BIFSScene scene();
}
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000113

9.3.6.17.2 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.

9.3.7 BIFS Scene

9.3.7.1 BIFSScene

9.3.7.1.1 Syntax

```
class BIFSScene() {
   bit(6) reserved;
   bit(1) USENAMES;
   PROTOlist protos;
   SFNode nodes(SFTopNode);
   bit(1) hasROUTEs;
   if (hasROUTEs) {
      ROUTEs routes();
   }
}
```

9.3.7.1.2 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 Annex H).

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**.

9.3.7.2. Encoding of PROTOs

This subclause describes how PROTOs, a mechanism that allow 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. A `PROTOlist` is stored in a `BIFSScene` and contains a list of PROTOs that are associated with that scene.

9.3.7.2.1 PROTOlist

9.3.7.2.1.1 Syntax

```
class PROTOlist() {
   bit(1) morePROTOs;
   while (morePROTOs) {
      PROTOdeclaration() proto;
      bit(1) morePROTOs;
   }
}
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000114

9.3.7.2.1.2 Semantics

The PROTOlist stores a list of PROTOs. A one-bit flag `morePROTOs` signals the fact that more PROTOs are being declared.

9.3.7.2.2 PROTOdeclaration

9.3.7.2.2.1 Syntax

```
PROTOdeclaration() {
PROTOinterfaceDefintion interface;
   NodeData protoData = MakePROTOdata(interface);
   PROTOcode code(protoData);
   PROTOcodingTable table(protoData);
}
```

9.3.7.2.2.2 Semantics

The PROTO declaration is made of the `PROTOinterface` defintion, 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.

9.3.7.2.3 PROTOinterfaceDefinition

9.3.7.2.3.1 Syntax

```
class PROTOinterfaceDefinition {
   bit(idBits) id;
   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 = makeFieldData(fieldType,eventType,isSF);
         Field(fieldData) defaultValue;
      }
      bit(1) moreFields;
   }
}
```

9.3.7.2.3.2 Semantics

An id 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, et). The `eventType` is coded using 2 bits according to Table 32. The `fieldType` is coded using 6 bits according to Table 32. 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000115

Table 32 - Field and EventTypes.

Field	0b00
exposedField	0b01
eventIn	0b10
eventOut	0b11

9.3.7.2.4 PROTOcode

9.3.7.2.4.1 Syntax

```
class PROTOcode(NodeData protoData) {
   bit(1) reserved;
   PROTOlist subProtos;
   do {
      SFNode node(SFWorldNodeType,protoData);
      bit(1) moreNodes;
   } while (moreNodes);
   bit(1) hasROUTEs;
   if (hasROUTEs) {
      ROUTEs routes();
   }
}
```

9.3.7.2.4.2 Semantics

The bit `reserved` is reserved for future extension. The bit shall be set to 0.

First a flag signals whether the prototype is a PROTO, which then has its code included in the 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 `SFNodes`, 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.

9.3.7.2.5 PROTOCodingTable

9.3.7.2.5.1 Syntax

```
PROTOCodingTable(NodeData protoData) {
     InterfaceCodingMask mask(protoData);
     InterfaceCodingParameters icp(protoData);
}
```

9.3.7.2.5.2 Semantics

The PROTO coding table defines the Quant and Anim parameters and the parameters necessary to reconstruct a NCT table for the PROTO definition.

9.3.7.2.6 InterfaceCodingMask

9.3.7.2.6.1 Syntax

```
InterfaceCodingMask(NodeData protoData) {
     bit(1) protoData.useQuant;
     bit(1) protoData.useAnim;
}
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000116

9.3.7.2.6.2 Semantics

The mask encodes two Boolean values to store whether the PROTO can further be animated (using BIFS-Anim), or quantized.

9.3.7.2.7 InterfaceCodingParameters

9.3.7.2.7.1 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) {
               CastToSF(Field) minFieldValue(protoData.field[i]));
               CastToSF(Field) maxFieldValue(protoData.field[i]]));
            }
         }
      }
      if (protoData.useAnim) {
         if (protoData.field[i].isIN()) {
            bit(1) isDyn;
            if (isDyn) {
               int(4) animCategory;
            }
         }
      }
   }
}
```

9.3.7.2.7.2 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.

9.3.7.3 SFNode

9.3.7.3.1 Syntax

```
class SFNode(int nodeDataType) {
   bit(1) isReused;
   if (isReused) {
      bit(BIFSConfiguration.nodeIDbits) nodeID;
   }
   else {
      bit(GetNDTnbBits(nodeDataType)) localNodeType;
      nodeType = GetNodeType(nodeDataType,localNodeType);
      if ((nodeType == IndexedFaceSetType) &&
                     (BIFSConfiguration.use3DmeshCoding == 1)) {
         bit(1) isUpdateable;
         if (isUpdateable) {
            bit(BIFSConfiguration.nodeIDbits) nodeID;
            if (USENAMES) {
               String name;
            }
         }
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000117

```
        Mesh3D mnode;
     }
     else {
        if (localNodeType == 0) {
           bit(GetNDTnbBitsExt(nodeDataType)    extLocalNodeType;
           if (extLocalNodeType == 1) {
              bit(BIFSConfiguration.PROTOIDbits) PROTOnodeType;
              nodeType = GetPROTONodeType(PROTODataType,PROTOnodeType)
           }
           if (extLocalNodeType > 1) {
                nodeType = GetExtNodeType(NodeDataType,extLocalNodeType)
           }
        }
        bit(1) isUpdateable;
        if (isUpdateable) {
           bit(BIFSConfiguration.nodeIDbits) nodeID;
           if (USENAMES) {
              String name;
           }
        }
        bit(1) MaskAccess;
        if (MaskAccess) {
           MaskNodeDescription mnode(MakeNode(nodeDataType, nodeType));
        }
        else {
           ListNodeDescription lnode(MakeNode(nodeDataType, nodeType));
        }
     }
   }
}
```

9.3.7.3.2 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.

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).

If a node is detected as an IndexedFaceSet node and the Mesh3D syntax is used (see 9.4.2.56), then the IndexedFaceSet node is coded as a specific visual object (see ISO/IEC 14496-2:1999).

If the node is not an IndexedFaceSet, the localNodeType is checked. If it is 0, then an extra extLocalNodeType is resolved, corresponding to the list of extended node types for this Node Data Type.

If this extLocalNodeType 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 and added to the global node type table of supported nodes. The function GetPROTONodeType(PROTONodeType) returns the ID for the extended global node type of a given PROTO given its PROTO type.

The function, GetExtNodeType(NodeDataType,extLocalNodeType) is used when a node is of extended node type, in which case the NodeDataType and extended local node type allow reconstruction of the extended global node type, i.e. the position of the node in the SFWorldNode extended table.

If the extLocalType is greater than 1 (0 is reserved), then the global node type is resolved using the Node Type for extended nodes.

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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000118

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 node data type 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 node data type tables in Annex H).

3. The `nodeType` gives the nature of the node to be parsed.

EXAMPLE — The **Shape** node has 2 fields defined as:

```
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
```

9.3.7.4 MaskNodeDescription

9.3.7.4.1 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]]);
            }
         }
      }
   } 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]]);
         }
      }
   }
}
```

9.3.7.4.2 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 Annex H. The field types are thus known and permit the field’s value to be decoded.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000119

9.3.7.5 ListNodeDescription

9.3.7.5.1 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]]);
         }
      }
      else {
                bit(node.nDEFbits) fieldRef;
                Field value(node.field[node.def2all[fieldRef]]);
      }
      bit(1) endFlag;
   }
}
```

9.3.7.5.2 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 9.3.2.3). When the fields belong to a PROTO, they may be ISed fields, indicated by `isedField`. In this case, a reference to the PROTO interface in coded in `protoField`. 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`. PROTO fields that are not Ised may have a default value assigned to them.

Non-PROTO fields always have a default value coded.

9.3.7.6 Field

9.3.7.6.1 Syntax

```
class Field(FieldData field) {
   if (isSF(field))
      SFField svalue(field);
   else
      MFField mvalue(field);
}
```

9.3.7.6.2 Semantics

A field is encoded according to its type: single (SFField) or multiple (MFField). A multiple field is a collection of single fields.

9.3.7.7 MFField

9.3.7.7.1 Syntax

```
class MFField(FieldData field)  {
   bit(1) reserved;
   if (!reserved) {
      bit(1) isListDescription;
      if (isListDescription)
         MFListDescription lfield(field);
      else
         MFVectorDescription vfield(field);
   }
}
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000120

9.3.7.7.2 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.

9.3.7.8 MFListDescription

9.3.7.8.1 Syntax

```
class MFListDescription(FieldData field)  {
   bit(1) endFlag;
   while (!endFlag) {
      SFField field(field);
      bit(1) endFlag;
   }
}
```

9.3.7.8.2 Semantics

The MFField type is encoded as a list of single fields.

9.3.7.9 MFVectorDescription

9.3.7.9.1 Syntax

```
class MFVectorDescription(FieldData field) {
   int(5) NbBits;
   int(NbBits)  numberOfFields;
   SFField field[numberOfFields](field);
}
```

9.3.7.9.2 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.

9.3.7.10 SFField

9.3.7.10.1 Syntax

```
class SFField(FieldData field) {
   switch (field.fieldType) {

      case SFNodeType:
         SFNode nValue(field.fieldType);
         break;

      case SFBoolType:
         SFBool bValue;
         break;

      case SFColorType:
         SFColor cValue(field);
         break;

      case SFFloatType:
         SFFloat fValue(field);
         break;

      case SFInt32Type:
         SFInt32 iValue(field);
         break;
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000121

```
      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;

      case SFImageType:
         SFImage imageValue(field);
         break;

      case SFCommandBufferType:
         SFCommandBuffer commandValue(field);
         break;

      case SFScriptType:
         SFScript scriptValue();
         break;
   }
}
```

9.3.7.10.2 Semantics

Each field is encoded according to its `fieldType`.

9.3.7.11 GenericFloat

9.3.7.11.1 Syntax

```
class GenericFloat(boolean useEfficientCoding) {
   if (!useEfficientCoding)
      float(32) value;
   else {
      EfficientFloat  value;
}
```

9.3.7.11.2 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.

9.3.7.12 EfficientFloat

9.3.7.12.1 Syntax

```
class EfficientFloat {
   unsigned int(4) mantissaLength;
   if (mantissaLength != 0) {
      int(3) exponentLength;
      int(1) mantissaSign;
      int(mantissaLength-1) mantissa;
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000122

```
      if (exponentLength != 0) {
         int(1) exponentSign;
         int(exponentLength-1) exponent;
      }
   }
}
```

9.3.7.12.2 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.

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:

$$(1-2.\text{mantissaSign}).(2^{\text{mantissaLength-1}}+\text{mantissa}).2^{(1-2.\text{exponentSign}).(2^{\text{exponentLength}-1}+\text{exponent})}$$

9.3.7.13 SFBool

9.3.7.13.1 Syntax

```
class SFBool {
   bit(1) value;
}
```

9.3.7.13.2 Semantics

If `value` is 1 the decoded boolean is set to TRUE. If `value` is 0, the decoded boolean is set to FALSE.

9.3.7.14 SFColor

9.3.7.14.1 Syntax

```
class SFColor(FieldData field) {
   if (field.isQuantized)
      QuantizedField qvalue(field);
   else {
      GenericFloat rValue(field.useEfficientCoding);
      GenericFloat gValue(field.useEfficientCoding);
      GenericFloat bValue(field.useEfficientCoding);
   }
}
```

9.3.7.14.2 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000123

9.3.7.15 SFFloat

9.3.7.15.1 Syntax

```
class SFFloat(FieldData field) {
   if (field.isQuantized)
      QuantizedField qvalue(field);
   else
      GenericFloat value(field.useEfficientCoding);
}
```

9.3.7.15.2 Semantics

If the field's `isQuantized` bit is TRUE, the `QuantizedField` scheme shall be used. Otherwise the SFFloat is coded using the `GenericFloat` scheme.

9.3.7.16 SFInt32

9.3.7.16.1 Syntax

```
class SFInt32(FieldData field) {
   if (field.isQuantized)
      QuantizedField qvalue(field);
   else
      int(32) value;
}
```

9.3.7.16.2 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.

9.3.7.17 SFRotation

9.3.7.17.1 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);
   }
}
```

9.3.7.17.2 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.

9.3.7.18 SFString

9.3.7.18.1 Syntax

```
class SFString {
   unsigned int(5) lengthBits;
   unsigned int(lengthBits) length;
   char(8) value[length];
}
```

9.3.7.18.2 Semantics

The `SFString` is coded as an array of characters whose length is first specified.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000124

`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).

9.3.7.19 SFTime

9.3.7.19.1 Syntax

```
class SFTime {
   double(64) value;
}
```

9.3.7.19.2 Semantics

The SFTime value is coded as a 64-bit double.

9.3.7.20 SFUrl

9.3.7.20.1 Syntax

```
class SFUrl {
   bit(1) isOD;
   if (isOD)
      bit(10) ODid;
   else
      SFString urlValue;
}
```

9.3.7.20.2 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. Otherwise the URL is sent as an `SFString`.

9.3.7.21 SFVec2f

9.3.7.21.1 Syntax

```
class SFVec2f(FieldData field) {
   if (field.isQuantized)
      QuantizedField qvalue(field);
   else {
      GenericFloat value1;
      GenericFloat value2;
   }
}
```

9.3.7.21.2 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.

9.3.7.22 SFVec3f

9.3.7.22.1 Syntax

```
class SFVec3f(FieldData field) {
   if (field.isQuantized)
      QuantizedField qvalue(field);
   else {
      GenericFloat value1(field.useEfficientCoding);
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000125

```
      GenericFloat value2(field.useEfficientCoding);
      GenericFloat value3(field.useEfficientCoding);
   }
}
```

9.3.7.22.2 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.

9.3.7.23 SFImage

9.3.7.23.1 Syntax

```
class SFImage {
   unsigned int(12)  width;
   unsigned int(12)  height;
   bit(2)  numComponents;
   bit(8)  pixels[(numComponents+1)*width*height];
}
```

9.3.7.23.2 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.

Pixels shall be decoded as unsigned char, 8-bit encoded pixel values.

9.3.7.24 SFCommandBuffer

9.3.7.24.1 Syntax

```
class SFCommandBuffer {
   unsigned int(5) lengthBits;
   unsigned int(lengthBits) length;
   bit(8) value[length];
}
```

9.3.7.24.2 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.

9.3.7.25 QuantizedField

9.3.7.25.1 Syntax

```
class QuantizedField(FieldData field) {
   switch (field.quantType) {
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000126

```
      case 9:
         int(1) direction
      case 10:
         int(2) orientation
      default:
         break;
    }
    for (i=0;i<getNbComp(field);i++)
      int(field.nbBits) vq[i]
}
```

9.3.7.25.2 Semantics

The value is quantized using the quantization process described in 9.3.3.

For normals, the direction and orientation values specified in the quantization process are first coded. For rotations, only the orientation value is coded.

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.

9.3.7.26 SFScript

9.3.7.26.1 Syntax

```
class SFScript() {
   bit(1) isListDescription;
   if (isListDescription)
      ScriptFieldsListDescripion();
   else
      ScriptFieldsVectorDescripion();
   const bit(1) reserved=1;
   EncodedScript();
}
```

9.3.7.26.2 Semantics

The `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.

9.3.7.27 ScriptFieldsListDescription

9.3.7.27.1 Syntax

```
class ScriptFieldsListDescription() {
   bit(1) endFlag; // List description of the fields
   while (!EndFlag) {
      ScriptField();
      bit(1) endFlag;
   }
}
```

9.3.7.27.2 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.

9.3.7.28 ScriptFieldsVectorDescription

9.3.7.28.1 Syntax

```
class ScriptFieldsVectorDescription() {
   bit(4) fieldBits;  // Number of bits for number of fields
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000127

```
   bit(fieldBits)  numFields; // Number of fields in the script
   for (i=0; i<numFields; ++i) {
      ScriptField();
   }
}
```

9.3.7.28.2 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`.

9.3.7.29 ScriptField

9.3.7.29.1 Syntax

```
class ScriptField() {
   bit(2) eventType;
   bit(6) fieldType;
   String fieldName;
   if (eventType == FIELD) {
      bit(1) hasInitialValue;
      if (hasInitialValue){
         NodeData node = makeNode(ScriptNodeType);
         Field(node.field[fieldType]) value;
      }
   }
}
```

9.3.7.29.2 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 33. 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).

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000128

Table 33 - 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	FVec2f
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

9.3.7.30 EncodedScript

9.3.7.30.1 Syntax

```
class EncodedScript {
   bit(1) hasFunction
   while (hasFunction) {
      Function function;
      bit(1) hasFunction
   }
}
```

9.3.7.30.2 Semantics

A script is a collection of functions, listed sequantially while `hasFunction` is TRUE.

9.3.7.31 Function

9.3.7.31.1 Syntax

```
class Function {
   Identifier identifier;
   Arguments arguments;
   StatementBlock statementBlock;
}
```

9.3.7.31.2 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.

9.3.7.32 Arguments

9.3.7.32.1 Syntax

```
class Arguments {
   bit(1) hasArgument
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000129

```
   while (hasArgument) {
      Identifier identifier;
      bit(1) hasArgument
   }
}
```

9.3.7.32.2 Semantics

The argument list is of arbitrary length, and terminates when `hasArgument` is 0. Each argument consists of one `identifier`.

9.3.7.33 StatementBlock

9.3.7.33.1 Syntax

```
class StatementBlock {
   bit(1) isCompoundStatement
   if (isCompoundStatement) {
      bit(1) hasStatement
      while (hasStatement) {
         Statement statement;
         bit(1) hasStatement
      }
      else {
         Statement statement;
      }
   }
}
```

9.3.7.33.2 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.

9.3.7.34 Statement

9.3.7.34.1 Syntax

```
class Statement {
   bit(3) statementType
   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;
   }
}
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000130

9.3.7.34.2 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.

9.3.7.35 IFStatement

9.3.7.35.1 Syntax

```
class IFStatement {
   CompoundExpression compoundExpression;
   StatementBlock statementBlock;
   bit(1) hasELSEStatement
   if (hasELSEStatement) {
      StatementBlock statementBlock;
   }
}
```

9.3.7.35.2 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 hasElseStatement is 1. This second, optional compoundStatement is interpreted when the CompoundExpression evaluates to a zero or empty value.

9.3.7.36 FORStatement

9.3.7.36.1 Syntax

```
class FORStatement {
   OptionalExpression optionalExpression;
   OptionalExpression optionalExpression;
   OptionalExpression optionalExpression;
   StatementBlock statementBlock;
}
```

9.3.7.36.2 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000131

9.3.7.37 WHILEStatement

9.3.7.37.1 Syntax

```
class WHILEStatement {
   CompoundExpression compoundExpression;
   StatementBlock statementBlock;
}
```

9.3.7.37.2 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.

9.3.7.38 RETURNStatement

9.3.7.38.1 Syntax

```
class RETURNStatement {
   bit(1) returnValue
   if (returnValue) {
      CompoundExpression compoundExpression;
   }
}
```

9.3.7.38.2 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`.

9.3.7.39 CompoundExpression

9.3.7.39.1 Syntax

```
class CompoundExpression {
   do {
      Expression expression;
      bit(1) hasExpression
   } while (hasExpression);
}
```

9.3.7.39.2 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.

9.3.7.40 SWITCHStatement

9.3.7.40.1 Syntax

```
class SWITCHStatement {
   do {
      CompoundExpression compoundExpression; // the switched value
      bit(5) numbits // number of bits for the case value
      bit(numbits) caseValue compoundExpression; #a case value
      StatementBlock statementBlock; // statements  in case
      bit(1) hasMoreCases
   } while (hasMoreCases);
   bit(1) hasDefault;
   if (hasDefault) {
      StatementBlock statementBlock; // default statements in case
   }
}
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000132

9.3.7.40.2 Semantics

A SWTICHStatement 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 StatementBlocks. The values represent the value of a case statement, which are encoded repeatedly until hasMoreCases is 0. An optinal default StatementBlock is then encoded.

9.3.7.41 optionalExpression

9.3.7.41.1 Syntax

```
class optionalExpression {
   bit(1) hasCompoundExpression
   if (hasCompoundExpression) {
      CompoundExpression compoundExpression;
   }
}
```

9.3.7.41.2 Semantics

An optionalExpression may be an empty expression, containing no executable statements, or a compoundExpression. This is indicated by the value of hasCompoundExpression.

9.3.7.42 Expression

9.3.7.42.1 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++
      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;
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000133

```
        Params params;
        break;
     case arrayDereferenceExpressionType:
        Expression expression;
        CompoundExpression compoundExpression;
        break;
     default: // =, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, >>>=,
        // ==, !=, <, <=, >, >=, +, -, *, /, %, &&, ||, &, |,
        // ^, <<, >>, >>>
        Expression expression;
        Expression expression;
        break;
   }
}
```

9.3.7.42.2 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.

`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:`

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000134

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:
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:

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000135

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:`
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:`

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000136

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`.

9.3.7.43 Params

9.3.7.43.1 Syntax

```
class Params {
   bit(1) hasParam
   while(hasParam) {
      Expression expression;
      bit(1) hasParam
   }
}
```

9.3.7.43.2 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`.

9.3.7.44 Identifier

9.3.7.44.1 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;
   }
}
```

9.3.7.44.2 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 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000137

9.3.7.45 String

9.3.7.45.1 Syntax

```
class String {
   bit(8) char
   while (char!=0) {
      bit(8) char
   }
}
```

9.3.7.45.2 Semantics

A `String` type consist of a null-terminated list of 8 bit characters.

9.3.7.46 Number

9.3.7.46.1 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
      }
   }
}
```

9.3.7.46.2 Semantics

A number shall be represented as an integer, indicated by `isInteger`, or as a list of 4 bit characters, represending (in order) the characters 0 ,1 ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,9, . , E,- , END-SYMBOL. The END-SYMBOL value 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.

9.3.7.47 Boolean

9.3.7.47.1 Syntax

```
class Boolean {
   bit(1) value
}
```

9.3.7.47.2 Semantics

A Boolean value is represented by a one-bit value.

9.3.7.48 ROUTEs

9.3.7.48.1 Syntax

```
class ROUTEs() {
   bit(1) ListDescription;
   if (ListDescription)
      ListROUTEs lroutes();
   else
      VectorROUTEs vroutes();
}
```

9.3.7.48.2 Semantics

`ROUTEs` may be encoded with a list (`ListROUTEs`) or vector (`VectorROUTEs`) description.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000138

9.3.7.49 ListROUTEs

9.3.7.49.1 Syntax

```
class ListROUTEs() {
   do {
      ROUTE route();
      bit(1) moreROUTEs;
   }
   while (moreROUTEs);
}
```

9.3.7.49.2 Semantics

The ROUTEs are coded as a list, with the `moreROUTEs` flag used to indicate the end of the list (when set to false).

9.3.7.50 VectorROUTEs

9.3.7.50.1 Syntax

```
class VectorROUTEs() {
   int(5) nBits;
   int(nBits) length;
   ROUTE route[length]();
}
```

9.3.7.50.2 Semantics

The ROUTEs are coded as a vector whose dimension, `length`, is first specified.

9.3.7.51 ROUTE

9.3.7.51.1 Syntax

```
class ROUTE() {
   bit(1) isUpdateable;
   if (isUpdateable) {
      bit(BIFSConfiguration.routeIDbits) routeID;
      if (USENAMES) {
         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;
}
```

9.3.7.51.2 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`.

9.3.8 BIFS-Anim

9.3.8.1 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

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000139

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 Annex H are known at the receiving terminal. The decoding data structure AnimationMask (see 9.3.2.5) 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 Annex G.

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 9.3.5.2), 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.

9.3.8.2 AnimationFrame

9.3.8.2.1 Syntax

```
class AnimationFrame() {
   AnimationFrameHeader header(BIFSConfiguration.animMask);
   AnimationFrameData data(BIFSConfiguration.animMask);
}
```

9.3.8.2.2 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.

9.3.8.3 AnimationFrameHeader

9.3.8.3.1 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;
}
```

9.3.8.3.2 Semantics

In the AnimationFrameHeader, a start code may be sent at each intra or prdictive frame to enable resynchronization. The first 23 bits are read ahead, and stored as the integer next.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000140

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.

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.

9.3.8.4 FrameRate

9.3.8.4.1 Syntax

```
class FrameRate {
   unsigned int(8) frameRate;
   unsigned int(4) seconds;
   bit(1) frequencyOffset;
}
```

9.3.8.4.2 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).

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).

9.3.8.5 SkipFrame

9.3.8.5.1 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);
}
```

9.3.8.5.2 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.

9.3.8.6 AnimationFrameData

9.3.8.6.1 Syntax

```
class AnimationFrameData (AnimationMask mask) {
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000141

```
   int i;
   for (i=0; i<mask.numNodes; i++) {
      if (mask.isActive[i]) {
           NodeData node = mask.animNode[i]
         switch (node.nodeType) {
         case FaceType:
            FaceFrameData fdata;
            break;
         case BodyType:
            BodyFrameData bdata;
            break;
         case IndexedFaceSet2DType:
            Mesh2DframeData mdata;
            break;
         default
            int j;
            for(j=0; j<node.numDYNfields; j++) {
               if (node.isAnimField[j])
                  AnimationField AField(node.field[node.dyn2all[j]],mask.isIntra);
            }
         }
      }
   }
}
```

9.3.8.6.2 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 9.3.8.7. In other cases, for each field declared as an animated field is the AnimationMask, the AnimationField is sent.

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 Annex G. Then, the animated values are sent using this adaptive arithmetic coder, using and updating their own models.

9.3.8.7 AnimationField

9.3.8.7.1 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);
   }
}
```

9.3.8.7.2 Semantics

In an AnimationField, if in intra mode, a new animation quantization parameter value may be sent. The intra or predictive 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 Annex G. If BIFSConfiguration.randomAccess is FALSE, the field's models are reset only if a new AnimQP is received.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000142

- 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.

In intra mode, if BIFSConfig.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 Annex G. If BIFSConfig.randomAccess is FALSE, the field's models are reset only if a new AnimQP is received.

The value is then sent: in intra mode, an AnimIValue is expected, in predictive mode an AnimPValue is expected.

9.3.8.8 AnimQP

9.3.8.8.1 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 11:   // Size 3D
        case 12:   // Size 2D
        case  7:   // Floats
           if (!aqp.useDefault) {
              for (i=0;i<getNbBounds(aqp);i++) {
                 bit(1)       useEfficientCoding
                 GenericFloat aqp.Imin[i](useEfficientCoding);
              }
              for (i=0;i<getNbBounds(aqp);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);i++) {
           int(INbBits+1) vq
           aqp.Pmin[i] = vq-2^aqp.INbBits;
      }
    }

   bit (1)hasPNbBits;
   if (hasPNbBits)
      unsigned int(4)        aqp.PNbBits;

}
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000143

9.3.8.8.2 Semantics

The AnimQP specifies the quantization parameters that shall be used until the next intra frame is received. AnimQP is identical to InitialAnimQP (subclause 9.3.5.7) 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.

9.3.8.9 AnimIValue

9.3.8.9.1 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];
}
```

9.3.8.9.2 Semantics

The AnimIValue represents the quantized intra value of a field. The value is coded according to the quantization process described in 9.3.3.3.

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.

9.3.8.10 AnimPValue

9.3.8.10.1 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];
}
```

9.3.8.10.2 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 9.3.4.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000144

The values are decoded from the adaptive arithmetic coder bitstream with the procedure v_{aac}=aa_decode(model) defined in Annex G. 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 Annex G).

9.4 Node Semantics

9.4.1 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.

9.4.2 Node specifications

9.4.2.1. AcousticMaterial

9.4.2.1.1. 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 Annex H.3.1.

9.4.2.1.2. 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

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000145

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** node. 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** node (see 9.4.2.39).

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:

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

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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000146

9.4.2.2. AcousticScene

9.4.2.2.1. 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 Annex H.3.2.

9.4.2.2.2. 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000147

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.

9.4.2.3 Anchor

9.4.2.3.1 Node interface

Anchor {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFString	**description**	""
exposedField	MFString	**parameter**	[]
exposedField	MFString	**url**	[]

}

NOTE — For the binary encoding of this node see Annex H.1.1.

9.4.2.3.2 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**). Constraints on URLs are defined by profiles and levels.

9.4.2.4 AnimationStream

9.4.2.4.1 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 Annex H.1.2.

9.4.2.4.2 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 9.2.1.6.1.

The semantics of the **speed** exposedField are identical to those for the **MovieTexture** node (see 9.4.2.72).

The **url** field specifies the data source to be used. The data source referred to shall be a BIFS-Anim stream (see also 9.2.3.3) or a BIFS-Command stream. In both cases, the stream shall operate within the same name scope as the scene containing the **AnimationStream** node.

9.4.2.5 Appearance

9.4.2.5.1 Node interface

Appearance {

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000148

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

}

NOTE — For the binary encoding of this node see Annex H.1.3.

9.4.2.5.2 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 **Layer2D** node, if no **Appearance** and therefore no **Material2D** are defined, the default values and behavior of the **Material2D** node shall be used.

9.4.2.6. ApplicationWindow

9.4.2.6.1. 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 Annex H.3.3.

9.4.2.6.2. 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000149

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.

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.

9.4.2.7 AudioBuffer

9.4.2.7.1 Node interface

AudioBuffer {

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

}

NOTE — For the binary encoding of this node see Annex H.1.4.

9.4.2.7.2 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 **AnimationStream** node are described in 9.2.1.6.1.

The semantics of the **speed** exposedField are identical to those for the **MovieTexture** node (see 9.4.2.72).

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** or **AudioBuffer**.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000150

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 9.2.2.13 and 9.4.2.12.

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.

9.4.2.8 AudioClip

9.4.2.8.1 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 Annex H.1.5.

9.4.2.8.2 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 9.2.1.6.1.

The **url** field specifies the data source to be used (see 9.2.2.7.1).

9.4.2.9 AudioDelay

9.4.2.9.1 Node interface

AudioDelay {

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000151

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 Annex H.1.6.

9.4.2.9.2 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.

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** or **AudioBuffer**.

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 9.2.1.6.1.

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.

9.4.2.10 AudioFX

9.4.2.10.1 Node interface

AudioFX {

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000152

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

}

NOTE — For the binary encoding of this node see Annex H.1.7.

9.4.2.10.2 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, clause 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** or **AudioBuffer**. 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** string 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 clause 5.8.

The **score** string 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 clause 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, clause 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 9.2.1.6.1.

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, subclauses 5.7.3.

9.4.2.11 AudioMix

9.4.2.11.1 Node interface

AudioMix {

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000153

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 Annex H.1.8.

9.4.2.11.2 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** or **AudioBuffer**.

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 9.2.1.6.1.

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.

9.4.2.12 AudioSource

9.4.2.12.1 Node interface

AudioSource {

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000154

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 Annex H.1.9.

9.4.2.12.2 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** data to be used as sound samples within a structured audio decoding process. Only **AudioBuffer** 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 and the parametric speech (HVXC) 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). The structured audio is the only decoder that may be controlled in this manner; 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, subpart 5). 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. The structured audio decoder is the only decoder that may be controlled in this manner; to adjust the speed of other decoder types, use the **AudioFX** node with an appropriate orchestra.

The **startTime** and **stopTime** exposedFields and their effects on the **AudioSource** node are described in 9.2.1.6.1.

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 9.2.2.7.1).

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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000155

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, clause 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.

9.4.2.13 AudioSwitch

9.4.2.13.1 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 Annex H.1.10.

9.4.2.13.2 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** or **AudioBuffer**.

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 9.2.1.6.1.

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*.

9.4.2.14 Background

9.4.2.14.1 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**	[]

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000156

exposedField	MFString	**topURL**	[]
exposedField	MFFloat	**skyAngle**	[]
exposedField	MFColor	**skyColor**	0, 0, 0
eventOut	SFBool	**isBound**	

}

NOTE — For the binary encoding of this node see Annex H.1.11.

9.4.2.14.2 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 9.2.2.7.1).

9.4.2.15 Background2D

9.4.2.15.1 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 Annex H.1.12.

9.4.2.15.2 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 9.2.2.7.1).

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.

EXAMPLE — Changing the background for 5 seconds.

```
Group {
   children [
      …
      DEF TIS TimeSensor {
         startTime 5.0
         stopTime  10.0
      }
      DEF BG1 Background2D {
         …
      }
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000157

```
   ]
}
ROUTE TIS.isActive TO BG1.set_bind
```

9.4.2.16 BAP

9.4.2.16.1 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

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000158

exposedField	SFInt32	**vc2torsion**	+I
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
exposedField	SFInt32	**vl3tilt**	+I

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000159

exposedField	SFInt32	**vl4roll**	+I
exposedField	SFInt32	**vl4torsion**	+I
exposedField	SFInt32	**vl4tilt**	+I
exposedField	SFInt32	**vl5roll**	+I
exposedField	SFInt32	**vl5torsion**	+I
exposedField	SFInt32	**vl5tilt**	+I
exposedField	SFInt32	**l_pinky0_flexion**	+I
exposedField	SFInt32	**r_pinky0_flexion**	+I
exposedField	SFInt32	**l_pinky1_flexion**	+I
exposedField	SFInt32	**r_pinky1_flexion**	+I
exposedField	SFInt32	**l_pinky1_pivot**	+I
exposedField	SFInt32	**r_pinky1_pivot**	+I
exposedField	SFInt32	**l_pinky1_twisting**	+I
exposedField	SFInt32	**r_pinky1_twisting**	+I
exposedField	SFInt32	**l_pinky2_flexion**	+I
exposedField	SFInt32	**r_pinky2_flexion**	+I
exposedField	SFInt32	**l_pinky3_flexion**	+I
exposedField	SFInt32	**r_pinky3_flexion**	+I
exposedField	SFInt32	**l_ring0_flexion**	+I
exposedField	SFInt32	**r_ring0_flexion**	+I
exposedField	SFInt32	**l_ring1_flexion**	+I
exposedField	SFInt32	**r_ring1_flexion**	+I
exposedField	SFInt32	**l_ring1_pivot**	+I
exposedField	SFInt32	**r_ring1_pivot**	+I
exposedField	SFInt32	**l_ring1_twisting**	+I
exposedField	SFInt32	**r_ring1_twisting**	+I
exposedField	SFInt32	**l_ring2_flexion**	+I
exposedField	SFInt32	**r_ring2_flexion**	+I
exposedField	SFInt32	**l_ring3_flexion**	+I
exposedField	SFInt32	**r_ring3_flexion**	+I
exposedField	SFInt32	**l_middle0_flexion**	+I
exposedField	SFInt32	**r_middle0_flexion**	+I
exposedField	SFInt32	**l_middle1_flexion**	+I
exposedField	SFInt32	**r_middle1_flexion**	+I
exposedField	SFInt32	**l_middle1_pivot**	+I
exposedField	SFInt32	**r_middle1_pivot**	+I
exposedField	SFInt32	**l_middle1_twisting**	+I
exposedField	SFInt32	**r_middle1_twisting**	+I
exposedField	SFInt32	**l_middle2_flexion**	+I
exposedField	SFInt32	**r_middle2_flexion**	+I
exposedField	SFInt32	**l_middle3_flexion**	+I
exposedField	SFInt32	**r_middle3_flexion**	+I
exposedField	SFInt32	**l_index0_flexion**	+I
exposedField	SFInt32	**r_index0_flexion**	+I
exposedField	SFInt32	**l_index1_flexion**	+I
exposedField	SFInt32	**r_index1_flexion**	+I
exposedField	SFInt32	**l_index1_pivot**	+I
exposedField	SFInt32	**r_index1_pivot**	+I
exposedField	SFInt32	**l_index1_twisting**	+I
exposedField	SFInt32	**r_index1_twisting**	+I
exposedField	SFInt32	**l_index2_flexion**	+I
exposedField	SFInt32	**r_index2_flexion**	+I
exposedField	SFInt32	**l_index3_flexion**	+I
exposedField	SFInt32	**r_index3_flexion**	+I
exposedField	SFInt32	**l_thumb1_flexion**	+I
exposedField	SFInt32	**r_thumb1_flexion**	+I
exposedField	SFInt32	**l_thumb1_pivot**	+I
exposedField	SFInt32	**r_thumb1_pivot**	+I
exposedField	SFInt32	**l_thumb1_twisting**	+I
exposedField	SFInt32	**r_thumb1_twisting**	+I
exposedField	SFInt32	**l_thumb2_flexion**	+I
exposedField	SFInt32	**r_thumb2_flexion**	+I

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000160

exposedField	SFInt32	**l_thumb3_flexion**	+l
exposedField	SFInt32	**r_thumb3_flexion**	+l
exposedField	SFInt32	**humanoidRoot_tr_vertical**	+l
exposedField	SFInt32	**humanoidRoot_tr_lateral**	+l
exposedField	SFInt32	**humanoidRoot_tr_frontal**	+l
exposedField	SFInt32	**humanoidRoot_rt_body_turn**	+l
exposedField	SFInt32	**humanoidRoot_rt_body_roll**	+l
exposedField	SFInt32	**humanoidRoot_rt_body_tilt**	+l
exposedField	SFInt32	**extensionBap187**	+l
exposedField	SFInt32	**extensionBap188**	+l
exposedField	SFInt32	**extensionBap189**	+l
exposedField	SFInt32	**extensionBap190**	+l
exposedField	SFInt32	**extensionBap191**	+l
exposedField	SFInt32	**extensionBap192**	+l
exposedField	SFInt32	**extensionBap193**	+l
exposedField	SFInt32	**extensionBap194**	+l
exposedField	SFInt32	**extensionBap195**	+l
exposedField	SFInt32	**extensionBap196**	+l
exposedField	SFInt32	**extensionBap197**	+l
exposedField	SFInt32	**extensionBap198**	+l
exposedField	SFInt32	**extensionBap199**	+l
exposedField	SFInt32	**extensionBap200**	+l
exposedField	SFInt32	**extensionBap201**	+l
exposedField	SFInt32	**extensionBap202**	+l
exposedField	SFInt32	**extensionBap203**	+l
exposedField	SFInt32	**extensionBap204**	+l
exposedField	SFInt32	**extensionBap205**	+l
exposedField	SFInt32	**extensionBap206**	+l
exposedField	SFInt32	**extensionBap207**	+l
exposedField	SFInt32	**extensionBap208**	+l
exposedField	SFInt32	**extensionBap209**	+l
exposedField	SFInt32	**extensionBap210**	+l
exposedField	SFInt32	**extensionBap211**	+l
exposedField	SFInt32	**extensionBap212**	+l
exposedField	SFInt32	**extensionBap213**	+l
exposedField	SFInt32	**extensionBap214**	+l
exposedField	SFInt32	**extensionBap215**	+l
exposedField	SFInt32	**extensionBap216**	+l
exposedField	SFInt32	**extensionBap217**	+l
exposedField	SFInt32	**extensionBap218**	+l
exposedField	SFInt32	**extensionBap219**	+l
exposedField	SFInt32	**extensionBap220**	+l
exposedField	SFInt32	**extensionBap221**	+l
exposedField	SFInt32	**extensionBap222**	+l
exposedField	SFInt32	**extensionBap223**	+l
exposedField	SFInt32	**extensionBap224**	+l
exposedField	SFInt32	**extensionBap225**	+l
exposedField	SFInt32	**extensionBap226**	+l
exposedField	SFInt32	**extensionBap227**	+l
exposedField	SFInt32	**extensionBap228**	+l
exposedField	SFInt32	**extensionBap229**	+l
exposedField	SFInt32	**extensionBap230**	+l
exposedField	SFInt32	**extensionBap231**	+l
exposedField	SFInt32	**extensionBap232**	+l
exposedField	SFInt32	**extensionBap233**	+l
exposedField	SFInt32	**extensionBap234**	+l
exposedField	SFInt32	**extensionBap235**	+l
exposedField	SFInt32	**extensionBap236**	+l
exposedField	SFInt32	**extensionBap237**	+l
exposedField	SFInt32	**extensionBap238**	+l
exposedField	SFInt32	**extensionBap239**	+l
exposedField	SFInt32	**extensionBap240**	+l

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000161

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
exposedField	SFInt32	**extensionBap296**	+I

}

NOTE - For the binary encoding of this node see Annex H.3.4.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000162

9.4.2.16.2 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: 1999.

9.4.2.17 BDP

9.4.2.17.1 Semantic Table

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

NOTE - For the binary encoding of this node see Annex H.3.5

9.4.2.17.2 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 Annex **M.2**.

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 Annex M.

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 14496-2:1999).

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 Annex M.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000163

- 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 9.4.2.21.

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

bodySegmentConnectionHint contains a **BodySegmentConnectionHint** node (see 9.4.2.22).

9.4.2.18 Billboard

9.4.2.18.1 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 Annex H.1.13.

9.4.2.18.2 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**).

9.4.2.19 Bitmap

9.4.2.19.1 Node interface

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

NOTE — For the binary encoding of this node see Annex H.1.14.

9.4.2.19.2 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.

Example — To specify semi-transparent video:

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000164

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

9.4.2.20 Body

9.4.2.20.1 Node interface

Body {

exposedField	SFNode	**bdp**	NULL
exposedField	SFNode	**bap**	NULL
exposedField	MFNode	**renderedBody**	[]

}

NOTE - For the binary encoding of this node see Annex H.3.6.

9.4.2.20.2 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.

9.4.2.21 BodyDefTable

9.4.2.21.1 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 Annex H.3.7.

9.4.2.21.2 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.

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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000165

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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000166

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 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 $\mathbf{d_1, d_2, d_3...d_n}$ 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 $\mathbf{f_i}$

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

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000167

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**.

9.4.2.22 BodySegmentConnectionHint

9.4.2.22.1 Node Interface

BodySegmentConnectionHint {			
exposedField	SFString	**firstSegmentNodeName**	NULL
exposedField	SFString	**secondSegmentNodeName**	NULL
exposedField	MFInt32	**firstVertexIdList**	[]
exposedField	MFInt32	**secondVertexIdList**	[]
}			

NOTE - For the binary encoding of this node see Annex H.3.8.

9.4.2.22.2 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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000168

9.4.2.23 Box

9.4.2.23.1 Node interface

```
Box {
    field       SFVec3f     size        2, 2, 2
}
```

NOTE — For the binary encoding of this node see Annex H.1.15.

9.4.2.23.2 Functionality and semantics

The semantics of the **Box** node are specified in ISO/IEC 14772-1:1998, subclause 6.7.

9.4.2.24 Circle

9.4.2.24.1 Node interface

```
Circle {
    exposedField    SFFloat     radius      1.0
}
```

NOTE — For the binary encoding of this node see Annex H.1.16.

9.4.2.24.2 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.

9.4.2.25 Collision

9.4.2.25.1 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 Annex H.1.17.

9.4.2.25.2 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).

9.4.2.26 Color

9.4.2.26.1 Node interface

```
Color {
    exposedField    MFColor     color       []
}
```

NOTE — For the binary encoding of this node see Annex H.1.18.

9.4.2.26.2 Functionality and semantics

The semantics of the **Color** node are specified in ISO/IEC 14772-1:1998, subclause 6.9.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000169

9.4.2.27 ColorInterpolator

9.4.2.27.1 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 Annex H.1.19.

9.4.2.27.2 Functionality and semantics

The semantics of the **ColorInterpolator** node are specified in ISO/IEC 14772-1:1998, subclause 6.10.

9.4.2.28 CompositeTexture2D

9.4.2.28.1 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
}
```

NOTE — For the binary encoding of this node see Annex H.1.20.

9.4.2.28.2 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.

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 9.4.2.63) fields of the same name.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000170

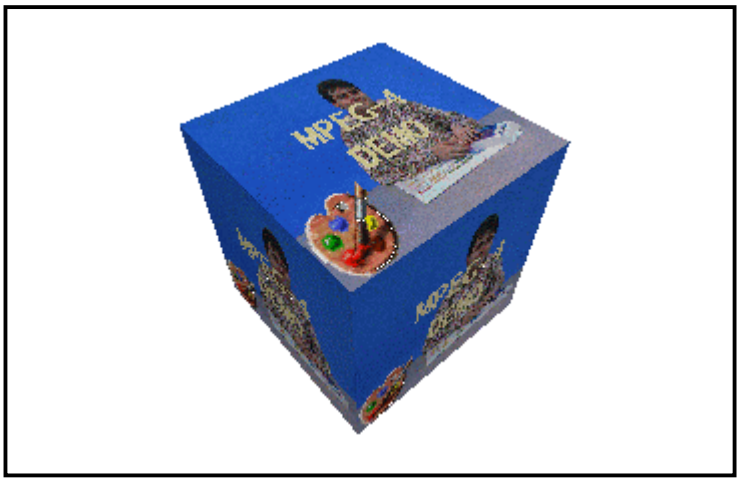


Figure 16 - A CompositeTexture2D **example. The 2D scene is projcted onto the 3D cube.**

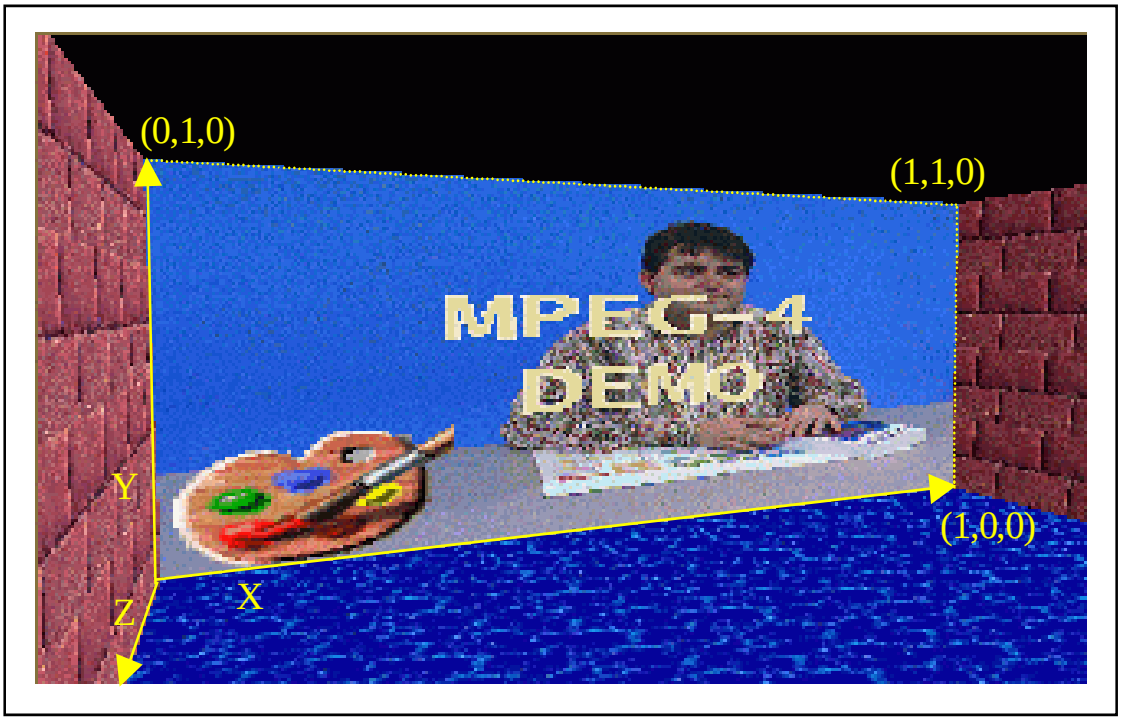


Figure 17 - A CompositeTexture2D **example.**

Here the 2D scene as defined in Figure 16 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.

9.4.2.29 CompositeTexture3D

9.4.2.29.1 Node interface

CompositeTexture3D {			
eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000171

exposedField	SFInt32	**pixelWidth**	-1
exposedField	SFInt32	**pixelHeight**	-1
exposedField	SFNode	**background**	
exposedField	SFNode	**fog**	
exposedField	SFNode	**navigationInfo**	
exposedField	SFNode	**viewpoint**	

}

NOTE — For the binary encoding of this node see Annex H.1.21.

9.4.2.29.2 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**.

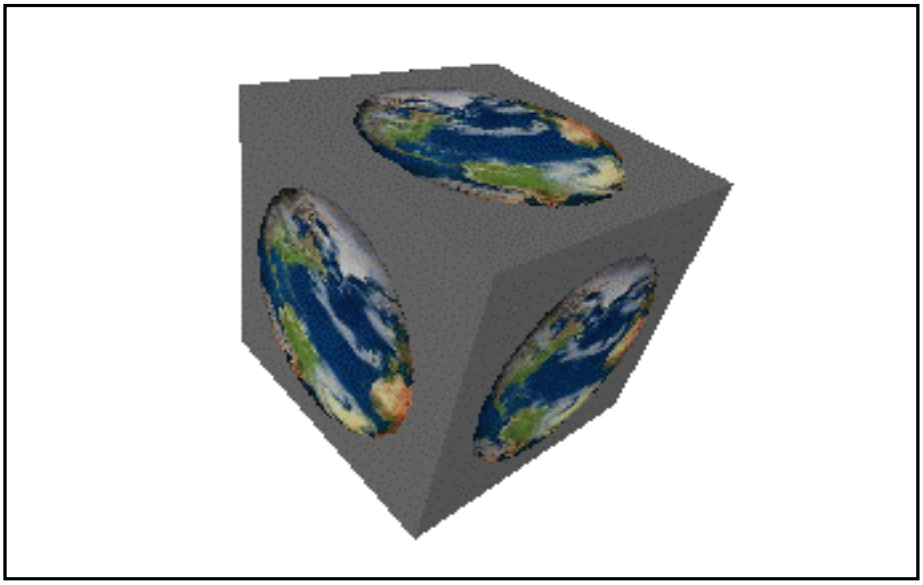


Figure 18 - CompositeTexture3D **example. The 3D view of the earth is projected onto the 3D cube**

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000172

9.4.2.30 Conditional

9.4.2.30.1 Node interface

```
Conditional {
    eventIn         SFBool      activate
    eventIn         SFBool      reverseActivate
    exposedField    SFString    buffer          ""
    eventOut        SFBool      isActive
}
```

NOTE — For the binary encoding of this node see Annex H.1.22.

9.4.2.30.2 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 9.3.6.2). These updates are not time-stamped; they are executed at the time of the event, assuming a zero-decoding time.

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.

9.4.2.31 Cone

9.4.2.31.1 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 Annex H.1.23.

9.4.2.31.2 Functionality and semantics

The semantics of the **Cone** node are specified in ISO/IEC 14772-1:1998, subclause 6.11.

9.4.2.32 Coordinate

9.4.2.32.1 Node interface

```
Coordinate {
    exposedField    MFVec3f     point           []
}
```

NOTE — For the binary encoding of this node see Annex H.1.24.

9.4.2.32.2 Functionality and semantics

The semantics of the **Coordinate** node are specified in ISO/IEC 14772-1:1998, subclause 6.12.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000173

9.4.2.33 Coordinate2D

9.4.2.33.1 Node interface

```
Coordinate2D {
    exposedField    MFVec2f     point       []
}
```

NOTE — For the binary encoding of this node see Annex H.1.25.

9.4.2.33.2 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 9.2.2.2).

9.4.2.34 CoordinateInterpolator

9.4.2.34.1 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 Annex H.1.26.

9.4.2.34.2 Functionality and semantics

The semantics of the **CoordinateInterpolator** node are specified in ISO/IEC 14772-1:1998, subclause 6.13.

9.4.2.35 CoordinateInterpolator2D

9.4.2.35.1 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 Annex H.1.27.

9.4.2.35.2 Functionality and semantics

CoordinateInterpolator2D is the 2D equivalent of **CoordinateInterpolator** (see 9.4.2.34).

9.4.2.36 Curve2D

9.4.2.36.1 Node interface

```
Curve2D {
    exposedField    SFNode      point       NULL
    exposedField    SFInt32     fineness    0
    exposedField    MFInt32     type        []
}
```

NOTE — For the binary encoding of this node see Annex H.1.28.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000174

9.4.2.36.2 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 which is also the number of points in the internal polygon. 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
         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
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000175

```
            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:

$$\mathbf{C_{1x}} = 2.\mathbf{P_x} - \mathbf{C'_{2x}} \text{and} \qquad \mathbf{C_{1y}} = 2.\mathbf{P_y} - \mathbf{C'_{2y}}$$

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.

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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000176

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 19, the curve is drawn in wide black, and the control polygon is drawn in dotted gray. The curve has two connex components.

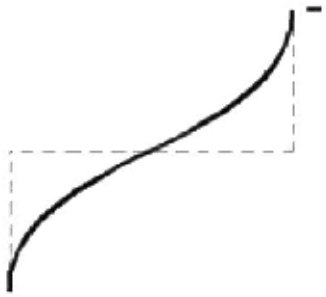


Figure 19 - Curve **node example**

9.4.2.37 Cylinder

9.4.2.37.1 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 Annex H.1.29.

9.4.2.37.2 Functionality and semantics

The semantics of the **Cylinder** node are specified in ISO/IEC 14772-1:1998, subclause 6.14.

9.4.2.38 CylinderSensor

9.4.2.38.1 Node interface

CylinderSensor {			
exposedField	SFBool	**autoOffset**	TRUE
exposedField	SFFloat	**diskAngle**	0.262
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 Annex H.1.30.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000177

9.4.2.38.2 Functionality and semantics

The semantics of the **CylinderSensor** node are specified in ISO/IEC 14772-1:1998, subclause 6.15.

9.4.2.39 DirectiveSound

9.4.2.39.1 Node interface

DirectiveSound {			
field	MFFloat	**angles**	0
field	MFFloat	**directivity**	1
field	MFFloat	**frequency**	[]
field	SFFloat	**speedOfSound**	340
field	SFFloat	**distance**	100
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 Annex H.3.9.

9.4.2.39.2 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 **9.2.2.13.4**).

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.

frequency field defines the frequencies at which the directivity gains are valid (similarly as **refFrequency** and **transFrequency** for **reffunc** and **transfunc** in **AcousticMaterial**, see 9.4.2.1).

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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000178

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):

$$n = \frac{length(\textbf{directivity})}{length(\textbf{angles})} = 2 \cdot M + 1 = m_b + m_a, ,$$

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:

[$freq_0$, $freq_1$, ...$freq_{nf-1}$]

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000179

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:

[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 9.4.2.1). 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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000180

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. Within one meter 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 gain function will be linearly attenuated on a dB scale between the source and the given cutoff distance. 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.

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.

9.4.2.40 DiscSensor

9.4.2.40.1 Node interface

DiscSensor {

exposedField	SFBool	**autoOffset**	TRUE
exposedField	SFBool	**enabled**	TRUE
exposedField	SFFloat	**maxAngle**	-1.0
exposedField	SFFloat	**minAngle**	-1.0
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 Annex H.1.31.

9.4.2.40.2 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000181

9.4.2.41 DirectionalLight

9.4.2.41.1 Node interface

```
DirectionalLight {
    exposedField    SFFloat     ambientIntensity    0.0
    exposedField    SFColor     color               1, 1, 1
    exposedField    SFVec3f     direction           0, 0, -1
    exposedField    SFFloat     intensity           1.0
    exposedField    SFBool      on                  TRUE
}
```

NOTE — For the binary encoding of this node see Annex H.1.32.

9.4.2.41.2 Functionality and semantics

The semantics of the **DirectionalLight** node are specified in ISO/IEC 14772-1:1998, subclause 6.16.

9.4.2.42 ElevationGrid

9.4.2.42.1 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 Annex H.1.33.

9.4.2.42.2 Functionality and semantics

The semantics of the **ElevationGrid** node are specified in ISO/IEC 14772-1:1998, subclause 6.17.

9.4.2.43 Expression

9.4.2.43.1 Node interface

```
Expression {
    Field           SFInt32     expression_select1      0
    Field           SFInt32     expression_intensity1   0
    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 Annex H.1.34.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000182

9.4.2.43.2 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.

9.4.2.44 Extrusion

9.4.2.44.1 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 Annex H.1.35.

9.4.2.44.2 Functionality and semantics

The semantics of the **Extrusion** node are specified in ISO/IEC 14772-1:1998, subclause 6.18.

9.4.2.45 Face

9.4.2.45.1 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 Annex H.1.36.

9.4.2.45.2 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000183

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 Annex I.

renderedFace is the scene graph of the face after it is rendered (all FAP's applied).

9.4.2.46 FaceDefMesh

9.4.2.46.1 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 Annex H.1.37.

9.4.2.46.2 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**:

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000184

Table 34 - Vertex displacements

Vertex no.	1st Interval [I1, I2]	2nd Interval [I2, I3]	…
Index 1	Displacement D11	Displacement D12	…
Index 2	Displacement D21	Displacement D22	…
…	…	…	…

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.

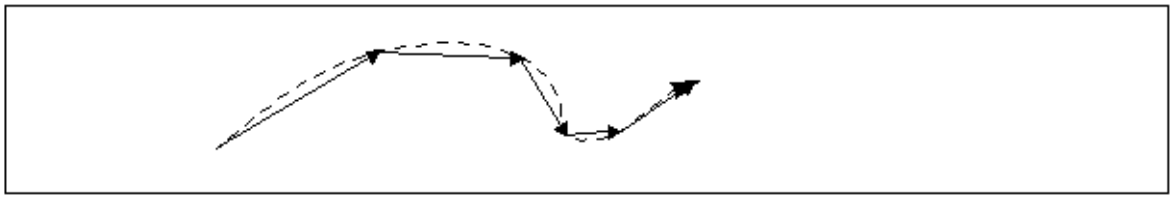


Figure 20 - An arbitrary motion trajectory is approximated as a piece-wise linear one.

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 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000185

9.4.2.47 FaceDefTables

9.4.2.47.1 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 Annex H.1.38.

9.4.2.47.2 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.

9.4.2.47.3 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.

9.4.2.48 FaceDefTransform

9.4.2.48.1 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 Annex H.1.39.

9.4.2.48.2 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

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000186

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.

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 FCD 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**

9.4.2.49 FAP

9.4.2.49.1 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

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000187

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
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 Annex H.1.40.

9.4.2.49.2 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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000188

9.4.2.50 FDP

9.4.2.50.1 Node interface

FDP {

exposedField	SFNode	**featurePointsCoord**	NULL
exposedField	SFNode	**textureCoords**	NULL
exposedField	SFBool	**useOrthoTexture**	FALSE
exposedField	MFNode	**faceDefTables**	[]
exposedField	MFNode	**faceSceneGraph**	[]

}

NOTE — For the binary encoding of this node see Annex H.1.41.

9.4.2.50.2 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

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000189

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.

9.4.2.51 FIT

9.4.2.51.1 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 Annex H.1.42.

9.4.2.51.2 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 21. Each node has a `nodeID`. The numerical label on each incoming link indicates the order of these links.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000190

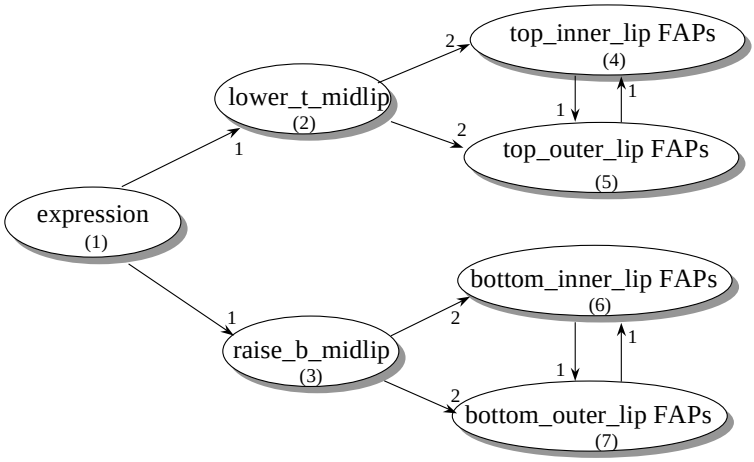


Figure 21 - A FIG example

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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000191

$$I(F_1, F_2, ..., F_n) = \sum_{i=0}^{K-1} \delta(F_{s_i} - a_i)(c_i \prod_{j=1}^{n} F_j{}^{l_{ij}}) \Big/ \sum_{i=0}^{P-1} (b_i \prod_{j=1}^{n} F_j{}^{m_{ij}})$$

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 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,

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000192

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

$$F_{12} = \delta(F_{72} - 6)(0.6F_{74}) + \delta(F_{73} - 6)(0.6F_{75})$$

$$F_{17} = \delta(F_{72} - 6)(-1.5F_{74}) + \delta(F_{73} - 6)(-1.5F_{75}) .$$

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]`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000193

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.

9.4.2.52 Fog

9.4.2.52.1 Node interface

Fog {

exposedField	SFColor	**color**	1 1 1
exposedField	SFString	**fogType**	"LINEAR"
exposedField	SFFloat	**visibilityRange**	0.0
eventIn	SFBool	**set_bind**	
eventOut	SFBool	**isBound**	

}

NOTE — For the binary encoding of this node see Annex H.1.43.

9.4.2.52.2 Functionality and semantics

The semantics of the **Fog** node are specified in ISO/IEC 14772-1:1998, subclause 6.19.

9.4.2.53 FontStyle

9.4.2.53.1 Node interface

FontStyle {

field	MFString	**family**	["SERIF"]
field	SFBool	**horizontal**	TRUE
field	MFString	**justify**	["BEGIN"]
field	SFString	**language**	""
field	SFBool	**leftToRight**	TRUE
field	SFFloat	**size**	1.0
field	SFFloat	**spacing**	1.0
field	SFString	**style**	"PLAIN"
field	SFBool	**topToBottom**	TRUE

}

NOTE — For the binary encoding of this node see Annex H.1.44.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000194

9.4.2.53.2 Functionality and semantics

The semantics of the **FontStyle** node are specified in ISO/IEC 14772-1:1998, subclause 6.20. The distance between adjacent text baselines is the sum of the size and the spacing. (Note, that this makes that the text size is the sum of the ascent + descent + leading, the latter which is the interline spacing, the logical amount of space to be reserved between the descent of one line of text and the ascent of the next line).

9.4.2.54 Form

9.4.2.54.1 Node interface

```
Form {
    eventIn         MFNode      addChildren
    eventIn         MFNode      removeChildren
    exposedField    MFNode      children        []
    exposedField    SFVec2f     size            -1, -1
    exposedField    MFInt32     groups          []
    exposedField    MFInt32     constraints     []
    exposedField    MFInt32     groupsIndex     []
}
```

NOTE — For the binary encoding of this node see Annex H.1.45.

9.4.2.54.2 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.

Constraints belong to two categories: alignment and distribution constraints.

Components referred to in the tables below are components 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.

Table 35 - Alignment Constraints

Alignment Constraints	Type Index	Effect
AL: Align Left edges	"AL"	The xmin of constrained components becomes equal to the xmin of the left-most component.
AH: Align centers Horizontally	"AH"	The (xmin+xmax)/2 of constrained components becomes equal to the (xmin+xmax)/2 of the group of constrained components as computed before this constraint is applied.
AR: Align Right edges	"AR"	The xmax of constrained components becomes equal to the xmax of the right-most component.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000195

Alignment Constraints	Type Index	Effect
AT: Align Top edges	“AT”	The ymax of all constrained components becomes equal to the ymax of the top-most component.
AV: Align centers Vertically	“AV”	The (ymin+ymax)/2 of constrained components becomes equal to the (ymin+ymax)/2 of the group of constrained components as computed before this constraint is applied.
AB: Align Bottom edges	“AB”	The ymin of constrained components becomes equal to the ymin of the bottom-most component.
ALspace: Align Left edges by specified space	“AL <s>”	The xmin of the second and following components become equal to the xmin of the first component plus the specified space.
ARspace: Align Right edges by specified space	“AR <s>”	The xmax of the second and following components becomes equal to the xmax of the first component minus the specified space.
ATspace: Align Top edges by specified space	“AT <s>”	The ymax of the second and following components becomes equal to the ymax of the first component minus the specified space.
ABspace: Align Bottom edges by specified space	“AB <s>”	The ymin of the second and following components become equal to the ymin of the first component plus the specified space.

The purpose of distribution constraints is to specify the space between components, by making such pairwise gaps equal either to a given value or to the effect of filling available space.

Table 36 - Distribution Constraints

Distribution Constraints	Type Index	Effect
SH: Spread Horizontally	“SH”	The differences between the xmin of each component and the xmax of the previous one all become equal. The first and the last component shall be constrained horizontally already.
SHin: Spread Horizontally in container	“SHin”	The differences between the xmin of each component 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 component and the xmax of the previous one all become equal to the specified space. The first component is not moved.
SV: Spread Vertically	“SV”	The differences between the ymin of each component and the ymax of the previous one all become equal. The first and the last component shall be constrained vertically already.
SVin: Spread Vertically in container	“SVin”	The differences between the ymin of each component 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 component and the ymax of the previous one all become equal to the specified space. The first component 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 {
   Geometry2D Rectangle { size 50 55 } // draw the Form’s frame.
   VisualProps use VPSRect
}

Transform2D {
   translation 10 10 {
      children [
              Form {
           children [
              Shape2D { use OBJ1 }
              Shape2D { use OBJ2 }
              Shape2D { use OBJ3 }
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000196

```
            Shape2D { use OBJ4 }
            Shape2D { use OBJ5 }
          ]
          size 50 55
groups [ 1 -1 2 -1 3 -1 4 -1 5 -1 1 3 -1]
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

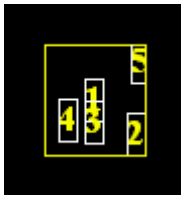


Figure 22 - Visual result of the Form node example

9.4.2.55 Group

9.4.2.55.1 Node interface

Group {

eventIn	MFNode	**addChildren**
eventIn	MFNode	**removeChildren**
exposedField	MFNode	**children** []

}

NOTE — For the binary encoding of this node see Annex H.1.46.

9.4.2.55.2 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 9.4.2.94.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000197

9.4.2.56 Hierarchical3Dmesh

9.4.2.56.1 Node Interface

Hierarchical3DMesh {

eventIn	SFInt32	**TriangleBudget**	
exposedField	SFFloat	**level**	
field	MFString	**url**	[]
eventOut	SFBool	**doneLoading**	

}

NOTE - For the binary encoding of this node see Annex H.3.10.

9.4.2.56.2 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 14496-2:1999). 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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000198

9.4.2.57 ImageTexture

9.4.2.57.1 Node interface

```
ImageTexture {
    exposedField    MFString    url         []
    field           SFBool      repeatS     TRUE
    field           SFBool      repeatT     TRUE
}
```

NOTE — For the binary encoding of this node see Annex H.1.47.

9.4.2.57.2 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 9.2.2.7.1).

9.4.2.58 IndexedFaceSet

9.4.2.58.1 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 Annex H.1.48.

9.4.2.58.2 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000199

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.

9.4.2.59 IndexedFaceSet2D

9.4.2.59.1 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 Annex H.1.49.

9.4.2.59.2 Functionality and semantics

The **IndexedFaceSet2D** node is the 2D equivalent of the **IndexedFaceSet** node as defined in 9.4.2.58. 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 9.4.2.58), 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 23 illustrates the default texture mapping coordinates for a simple **IndexedFaceSet2D** shape consisting of a single polygonal face.

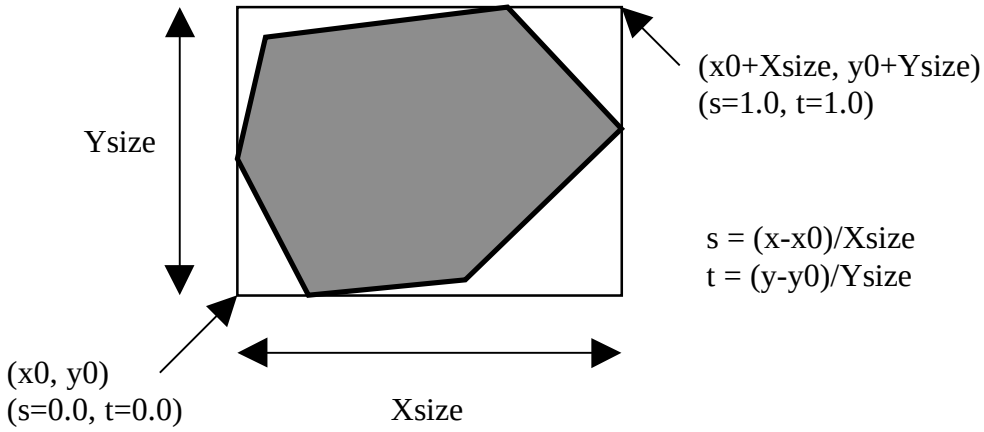


Figure 23 - IndexedFaceSet2D **default texture mapping coordinates for a simple shape**

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000200

9.4.2.60 IndexedLineSet

9.4.2.60.1 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 Annex H.1.50.

9.4.2.60.2 Functionality and semantics

The semantics of the **IndexedLineSet** node are specified in ISO/IEC 14772-1:1998, subclause 6.24.

9.4.2.61 IndexedLineSet2D

9.4.2.61.1 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 Annex H.1.51.

9.4.2.61.2 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 9.4.2.60).

9.4.2.62 Inline

9.4.2.62.1 Node interface

```
Inline {
    exposedField    MFString    url             []
}
```

NOTE — For the binary encoding of this node see Annex H.1.52.

9.4.2.62.2 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**).

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000201

The **url** field specifies the data source to be used (see 9.2.2.7.1). The external source must contain a valid BIFS scene, and may include BIFS-Commands and BIFS-Anim frames.

9.4.2.63 Layer2D

9.4.2.63.1 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 Annex H.1.53.

9.4.2.63.2 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 24).

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 9.2.2.1).

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 9.2.2.2). 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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000202

Layers are transparent to user input, which means that if two layers are overlapping at a given location on the screen, a user input will affect both layers, regardless of which is drawn on top of the other. For instance, if two buttons placed in two different layers are overlapping, the click of the user at the location of the topmost button will also affect the button contained in the layer behind. Authors should carefully design behaviors in the overlapping 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
         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
```

9.4.2.64 Layer3D

9.4.2.64.1 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
}			

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000203

NOTE — For the binary encoding of this node see Annex H.1.54.

9.4.2.64.2 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 9.4.2.63) 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 9.2.2.1).

The following fields specify bindable children nodes for **Layer3D**:

- **background** for **Background** 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.

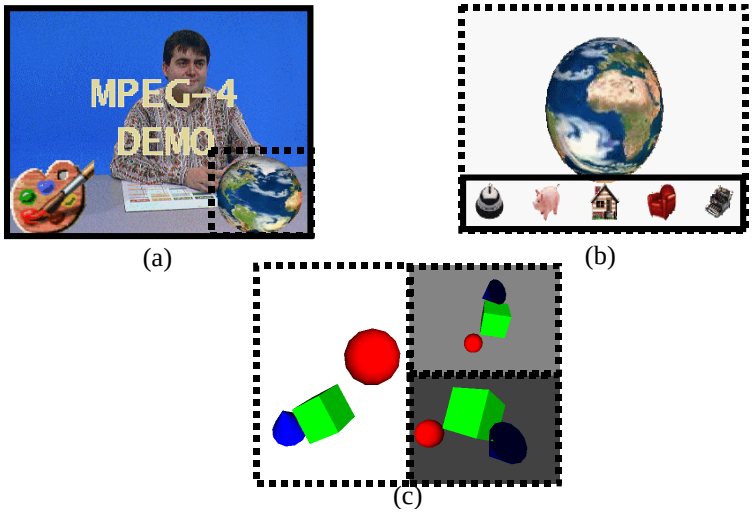


Figure 24 - Three Layer2D **and** Layer3D **examples composed in a 2D space.**

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**.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000204

9.4.2.65 Layout

9.4.2.65.1 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
}			

NOTE — For the binary encoding of this node see Annex H.1.55.

9.4.2.65.2 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.

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 9.4.2.53).

The **scrollRate** field specifies the scroll rate in meters per second. 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000205

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.

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).

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000206

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.

9.4.2.66 LineProperties

9.4.2.66.1 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 Annex H.1.56.

9.4.2.66.2 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 37 - 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 apparent width depends on 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 25.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000207

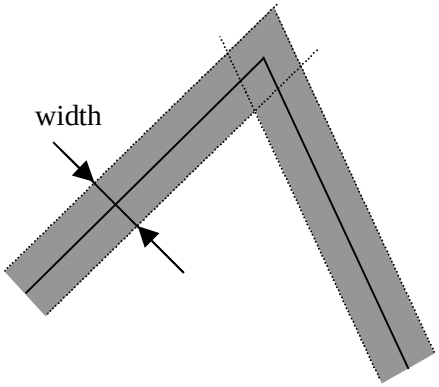


Figure 25 - Cap and join style for LineProperties

9.4.2.67 ListeningPoint

9.4.2.67.1 Node interface

ListeningPoint {

eventIn	SFBool	**set_bind**	
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 Annex H.1.57.

9.4.2.67.2 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 9.4.2.109).

9.4.2.68 LOD

9.4.2.68.1 Node interface

LOD {

exposedField	MFNode	**level**	[]
field	SFVec3f	**center**	0, 0, 0
field	MFFloat	**range**	[]

}

NOTE — For the binary encoding of this node see Annex H.1.58.

9.4.2.68.2 Functionality and semantics

The semantics of the **LOD** node are specified in ISO/IEC 14772-1:1998, subclause 6.26.

9.4.2.69 Material

9.4.2.69.1 Node interface

Material {

exposedField	SFFloat	**ambientIntensity**	0.2
exposedField	SFColor	**diffuseColor**	0.8, 0.8, 0.8
exposedField	SFColor	**emissiveColor**	0, 0, 0

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000208

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 Annex H.1.59.

9.4.2.69.2 Functionality and semantics

The semantics of the **Material** node are specified in ISO/IEC 14772-1:1998, subclause 6.27.

9.4.2.70 Material2D

9.4.2.70.1 Node interface

Material2D {

exposedField	SFColor	**emissiveColor**	0.8, 0.8, 0.8
exposedField	SFBool	**filled**	FALSE
exposedField	SFNode	**lineProps**	NULL
exposedField	SFFloat	**transparency**	0.0

}

NOTE — For the binary encoding of this node see Annex H.1.60.

9.4.2.70.2 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 9.4.2.5.2)

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. If the field is null the line properties take on a default behaviour identical to the default settings of the **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, whether explicitly specified or default values, are taken from the **lineProps** node. See 9.4.2.66 for more information on **LineProperties**.

The **transparency** field specifies the transparency of the 2D **Shape**.

9.4.2.71 MaterialKey

9.4.2.71.1 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 Annex H.3.11.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000209

9.4.2.71.2 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.
- 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 Annex N.

9.4.2.72 MovieTexture

9.4.2.72.1 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 Annex H.1.61.

9.4.2.72.2 Functionality and semantics

The **loop**, **startTime**, and **stopTime** exposedFields and the **isActive** eventOut, and their effects on the **MovieTexture** node, are described in 9.2.1.6.1.

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**)

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000210

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 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 9.2.2.7.1).

9.4.2.73 NavigationInfo

9.4.2.73.1 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 Annex H.1.62.

9.4.2.73.2 Functionality and semantics

The semantics of **NavigationInfo** are specified in ISO/IEC 14772-1:1998, subclause 6.29.

9.4.2.74 Normal

9.4.2.74.1 Node interface

Normal {

exposedField	MFVec3f	**vector**	[]

}

NOTE — For the binary encoding of this node see Annex H.1.63.

9.4.2.74.2 Functionality and semantics

The semantics of the **Normal** node are specified in ISO/IEC 14772-1:1998, subclause 6.30.

9.4.2.75 NormalInterpolator

9.4.2.75.1 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 Annex H.1.64.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000211

9.4.2.75.2 Functionality and semantics

The semantics of the **NormalInterpolator** node are specified in ISO/IEC 14772-1:1998, subclause 6.31.

9.4.2.76 OrderedGroup

9.4.2.76.1 Node interface

```
OrderedGroup {
    eventIn         MFNode    addChildren
    eventIn         MFNode    removeChildren
    exposedField    MFNode    children        []
    exposedField    MFFloat   order           []
}
```

NOTE — For the binary encoding of this node see Annex H.1.65.

9.4.2.76.2 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.

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.

9.4.2.77 OrientationInterpolator

9.4.2.77.1 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 Annex H.1.66.

9.4.2.77.2 Functionality and semantics

The semantics of the **OrientationInterpolator** node are specified in ISO/IEC 14772-1:1998, subclause 6.32.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000212

9.4.2.78 PerceptualParameters

9.4.2.78.1 Node interface

PerceptualParameters {			
exposedField	Float	**sourcePresence**	1.0
exposedField	Float	**sourceWarmth**	1.0
exposedField	Float	**sourceBrilliance**	1.0
exposedField	Float	**roomPresence**	1.0
exposedField	SFTime	**runningReverberance**	1.0
exposedField	Float	**envelopment**	0.0
exposedField	SFTime	**lateReverberance**	1.0
exposedField	Float	**heavyness**	1.0
exposedField	Float	**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
}			

NOTE - For the binary encoding of this node see Annex H.3.12.

9.4.2.78.2 Functionality and Semantics

PerceptualParameters is a node that contains information about the perceptual properties of **DirectiveSound** objects to be processed in the same auralization process 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}.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000213

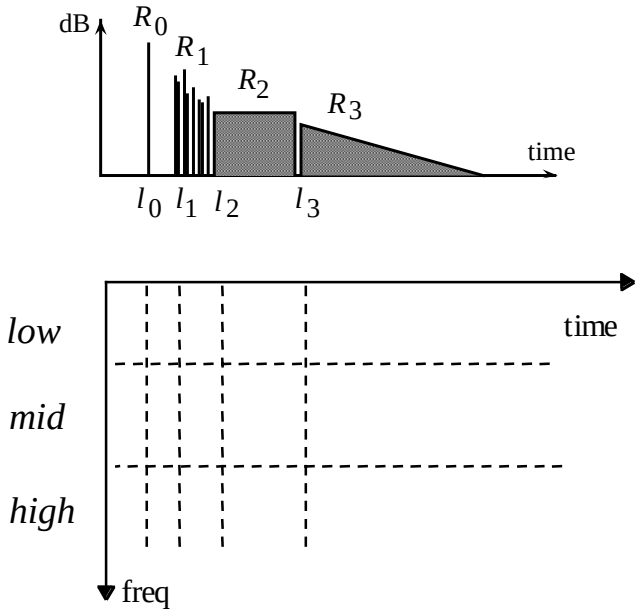


Figure 26 - Generic reverberation response model. R0 represents the direct sound, R1 the directional early relfections, R2 the diffuse reflections, ad R3 the exponentially decaying, diffuse late reverberation.

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}

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 factor. These objective 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 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 nine perceptual factors that have been determined in the experiments are denominated as follows, and can be divided in three groups:

Three perceptual factors describe effects which are characteristic of the room (the corresponding objective criteria are indicated in parentheses):

- late reverberance (late decay time)
- heaviness and liveness (variation of decay time with frequency)

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000214

The six other factors describe effects that depend of the position, directivity and orientation of the source. The first three are perceived as characteristics of the source, while the remaining three are perceptually associated with the room:

- source presence (energy of the direct sound and early room effect)
- brilliance and warmth (variation of early energy with frequency)
- room presence (energy of late room effect)
- running reverberance (early decay time)
- envelopment (energy of early room effect relative to direct sound)

A variation of the source presence 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. Late reverberance differs from running reverberance by the fact that it is essentially perceived during interruptions of the sound radiated by the source, for example when the source falls silent. Running reverberance, on the contrary, remains perceived during continuous music.

- **sourcePresence** field indicates the total absolute energy of the R1 in the room acoustic response in Figure 26.
- **sourceWarmth** and **sourceBrilliance** fields are used for calculation of energies of R1 at the frequency limits **freqLow**, and **freqHigh**.
- **roomPresence** is the total absolut energy of R3 in Figure 26.
- **runningReverberance** is a relative early decay time (with respect to its minimum and maximum values). It is used to compute the absolute value of the decay time in seconds from its variation boundaries as explained below.
- **envelopment** is the ratio between R1 and R0. This value is relative with respect to its variation boundaries.
- **heaviness** and **liveness** define the decay time as a function of frequency. They are used in the computation of the late reverberance at the different frequency bands defined by **freqLow** and **freqHigh**.

omniDirectivity is the diffuse-field spectrum for the source is required in the Room processor. 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 0 Hz is $\mathbf{gain}_0$.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000215

By default, the gain for frequencies above $f_{nf-1}$is **gain** $_{nf-1}$.

directFilterGains specifies a filter applied to the direct path only (power 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**.

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).

9.4.2.78.2.1 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 38 - Perceptual parameters

Perceptual parameter field	*Notation*	*Min*	*max*
sourcePresence	Es	0.0	1.0
sourceBrilliance	Desl	0.1	10.0
sourceWarmth	Desh	0.1	10.0
roomPresence	Rev	0.0	1.0
runningReverberance	Edt_{rel}	0.0	0.1
envelopment	Rdl_{rel}	0.0	0.1
lateReverberance	Rt (s)	0.1	100.0
heaviness	Drtl	0.1	10.0
liveness	Drth	0.1	1.0

Es and Rev are absolute energies, and Edt_{rel} is a relative energy value. Desl, Desh, Drtl, Drth are multiplicative factors. Rt (reverberation time) is expressed in seconds, and Edt 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)

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000216

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)]

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 in seconds is computed as:

Rdl = Rdl_{min} + (Rdl_{max} - Rdl_{min})* Rdl_{rel}

R1= (Es*Rdl - 0.05*R2) / 0.3

$R1_{low}$= R1*Desl

$R1_{high}$= R1*Desh

R0= Es - R1

$R0_{low}$= R0*Desl

$R0_{high}$= R0*Desh

Rt_{low} = Drtl * Rt

Rt_{high} = Drth * Rt

NOTE - All the values are energies expressed in the linear domain

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000217

9.4.2.78.2.2 Mapping from positional to perceptual parameters

If the source is not placed at the reference distance d_{ref} for which the perceptual "preset" is defined, the following correction is applied (when **useAttenuation** is TRUE):

$Es=Es*[d / d_{ref}]^{-2}$ where d is the actual distance between the source and the viewpoint. For an example implementation of percpetual approach, see Annex O.

9.4.2.79 PixelTexture

9.4.2.79.1 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 Annex H.1.67.

9.4.2.79.2 Functionality and semantics

The semantics of the **PixelTexture** node are specified in ISO/IEC 14772-1:1998, subclause 6.33.

9.4.2.80 PlaneSensor

9.4.2.80.1 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**	

}

9.4.2.80.2 Fnctionality and semantics

The semantics of the **PlaneSensor** node are specified in ISO/IEC 14772-1:1998, subclause 6.34.

9.4.2.81 PlaneSensor2D

9.4.2.81.1 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 Annex H.1.68.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000218

9.4.2.81.2 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 9.4.2.80).

9.4.2.82 PointLight

9.4.2.82.1 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
}			

NOTE — For the binary encoding of this node see Annex H.1.69.

9.4.2.82.2 Functionality and semantics

The semantics of the **PointLight** node are specified in ISO/IEC 14772-1:1998, subclause 6.35.

9.4.2.83 PointSet

9.4.2.83.1 Node interface

PointSet {			
exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL
}			

NOTE — For the binary encoding of this node see Annex H.1.70.

9.4.2.83.2 Functionality and semantics

The semantics of the **PointSet** node are specified in ISO/IEC 14772-1:1998, subclause 6.36.

9.4.2.84 PointSet2D

9.4.2.84.1 Node interface

PointSet2D {			
exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL
}			

NOTE — For the binary encoding of this node see Annex H.1.71.

9.4.2.84.2 Functionality and semantics

This is a 2D equivalent of the **PointSet** node (see 9.4.2.83), with semantics that are the 2D restriction of that node.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000219

9.4.2.85 PositionInterpolator

9.4.2.85.1 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 Annex H.1.72.

9.4.2.85.2 Functionality and semantics

The semantics of the **PositionInterpolator** node are specified in ISO/IEC 14772-1:1998, subclause 6.37.

9.4.2.86 PositionInterpolator2D

9.4.2.86.1 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 Annex H.1.73.

9.4.2.86.2 Functionality and semantics

This is a 2D equivalent of the **PositionInterpolator** node (see 9.4.2.85) with semantics that are the 2D restriction of that node.

9.4.2.87 ProximitySensor

9.4.2.87.1 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 Annex H.1.74.

9.4.2.87.2 Functionality and semantics

The semantics of the **ProximitySensor** node are specified in ISO/IEC 14772-1:1998, subclause 6.38.

9.4.2.88 ProximitySensor2D

9.4.2.88.1 Node interface

```
ProximitySensor2D {
    exposedField    SFVec2f     center              0, 0
    exposedField    SFVec2f     size                0, 0
    exposedField    SFBool      enabled             TRUE
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000220

```
    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 Annex H.1.75.

9.4.2.88.2 Functionality and semantics

This is the 2D equivalent of the **ProximitySensor** node (see 9.4.2.87) with semantics that are the 2D restriction of the that node.

9.4.2.89 QuantizationParameter

9.4.2.89.1 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          TRUE
    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                2¶
    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 Annex H.1.76.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000221

9.4.2.89.2 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:

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.

In all cases, 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 9.3.3.

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 9.3.7.12. 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.

9.4.2.90 Rectangle

9.4.2.90.1 Node interface

Rectangle {
exposedField SFVec2f **size** 2, 2
}

NOTE — For the binary encoding of this node see Annex H.1.77.

9.4.2.90.2 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.

9.4.2.91 ScalarInterpolator

9.4.2.91.1 Node interface

ScalarInterpolator {
eventIn SFFloat **set_fraction**
exposedField MFFloat **key** []

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000222

exposedField	MFFloat	**keyValue**	[]
eventOut	SFFloat	**value_changed**	

}

NOTE — For the binary encoding of this node see Annex H.1.78.

9.4.2.91.2 Functionality and semantics

The semantics of the **ScalarInterpolator** node are specified in ISO/IEC 14772-1:1998, subclause 6.39.

9.4.2.92 Script

9.4.2.92.1 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 Annex H.1.79.

9.4.2.92.2 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
- 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.

9.4.2.92.2.1 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}
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000223

```
      }
    }
    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
```

9.4.2.93 Shape

9.4.2.93.1 Node interface

Shape {

exposedField	SFNode	**appearance**	NULL
exposedField	SFNode	**geometry**	NULL

}

NOTE — For the binary encoding of this node see Annex H.1.80.

9.4.2.93.2 Functionality and semantics

The semantics of the **Shape** node are specified in ISO/IEC 14772-1:1998, subclause 6.41.

9.4.2.94 Sound

9.4.2.94.1 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

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000224

exposedField	SFNode	**source**	NULL
field	SFBool	**spatialize**	TRUE

}

NOTE — For the binary encoding of this node see Annex H.1.81.

9.4.2.94.2 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 9.2.2.13.2.2.

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 9.2.2.13.2.2.

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, 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]

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000225

9.4.2.95 Sound2D

9.4.2.95.1 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 Annex H.1.82.

9.4.2.95.2 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 9.2.2.1). 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 9.4.2.94). 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 9.4.2.94. If **Sound** and **Sound2D** nodes are both used in a scene, all shall be treated the same way according to these semantics.

9.4.2.96 Sphere

9.4.2.96.1 Node interface

```
Sphere {
    field           SFFloat     Radius          1.0
}
```

NOTE — For the binary encoding of this node see Annex H.1.83.

9.4.2.96.2 Functionality and semantics

The semantics of the **Sphere** node are specified in ISO/IEC 14772-1:1998, subclause 6.43.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000226

9.4.2.97 SphereSensor

9.4.2.97.1 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 Annex H.1.84.

9.4.2.97.2 Functionality and semantics

The semantics of the **SphereSensor** node are specified in ISO/IEC 14772-1:1998, subclause 6.44.

9.4.2.98 SpotLight

9.4.2.98.1 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 Annex H.1.85.

9.4.2.98.2 Functionality and semantics

The semantics of the **SpotLight** node are specified in ISO/IEC 14772-1:1998, subclause 6.45.

9.4.2.99 Switch

9.4.2.99.1 Node interface

```
Switch {
    exposedField    MFNode          choice                  []
    exposedField    SFInt32         whichChoice             -1
}
```

NOTE — For the binary encoding of this node see Annex H.1.86.

9.4.2.99.2 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000227

9.4.2.100 TermCap

9.4.2.100.1 Node interface

```
TermCap {
    eventIn     SFTime      evaluate
    field       SFInt32     capability          0
    eventOut    SFInt32     value
}
```

NOTE — For the binary encoding of this node see Annex H.1.87.

9.4.2.100.2 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 39 - Semantics of value**, dependent on** capability

capability	Semantics of value
0	frame rate
1	color depth
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 40 - 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

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000228

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 41 - 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 42 - 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

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 43 - 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)

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000229

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 44 - 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 45 - 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.

Table 46 - 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%.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000230

Table 47 - 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 48 - 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

9.4.2.101 Text

9.4.2.101.1 Node interface

```
Text {
    exposedField    MFString    string      []
    exposedField    MFFloat     length      []
    exposedField    SFNode      fontStyle   NULL
    exposedField    SFFloat     maxExtent   0.0
}
```

NOTE — For the binary encoding of this node see Annex H.1.88.

9.4.2.101.2 Functionality and semantics

The semantics of the **Text** node are specified in ISO/IEC 14772-1:1998, subclause 6.47.

9.4.2.102 TextureCoordinate

9.4.2.102.1 Node interface

```
TextureCoordinate {
    exposedField    MFVec2f     point       []
}
```

NOTE — For the binary encoding of this node see Annex H.1.89.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000231

9.4.2.102.2 Functionality and semantics

The semantics of the **TextureCoordinate** node are specified in ISO/IEC 14772-1:1998, subclause 6.48.

9.4.2.103 TextureTransform

9.4.2.103.1 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 Annex H.1.90.

9.4.2.103.2 Functionality and semantics

The semantics of the **TextureTransform** node are specified in ISO/IEC 14772-1:1998, subclause 6.49.

9.4.2.104 TimeSensor

9.4.2.104.1 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 Annex H.1.91.

9.4.2.104.2 Functionality and semantics

The semantics of the **TimeSensor** node are specified in ISO/IEC 14772-1:1998, subclause 6.50.

9.4.2.105 TouchSensor

9.4.2.105.1 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 Annex H.1.92.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000232

9.4.2.105.2 Functionality and semantics

The semantics of the **TouchSensor** node are specified in ISO/IEC 14772-1:1998, subclause 6.51.

9.4.2.106 Transform

9.4.2.106.1 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 Annex H.1.93.

9.4.2.106.2 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 9.4.2.94.

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.

9.4.2.107 Transform2D

9.4.2.107.1 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 Annex H.1.94.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000233

9.4.2.107.2 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 **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 9.4.2.94.

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.

9.4.2.108 Valuator

9.4.2.108.1 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**
eventIn	SFVec3f	**inSFVec3f**
eventIn	MFVec3f	**inMFVec3f**
eventOut	SFBool	**outSFBool**
eventOut	SFColor	**outSFColor**

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000234

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 Annex H.1.95.

9.4.2.108.2 Functionality and semantics

A **Valuator** node can receive an event of any type, and on reception of such an event, will trigger eventOuts of many different types. Upon reception of an event on any of its eventIns, on each eventOut connected to a ROUTE an event will be generated. The value of this event is governed by the equation below. This node serves as a simple type casting method.

Each output value is dependent on the input value with the following relationship:

output value = factor * x + offset

In the above equation, factor is one of the exposedField values and offset is one of the eventOut values specified in the node inteface. All values specified in the above equation are floating point values.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000235

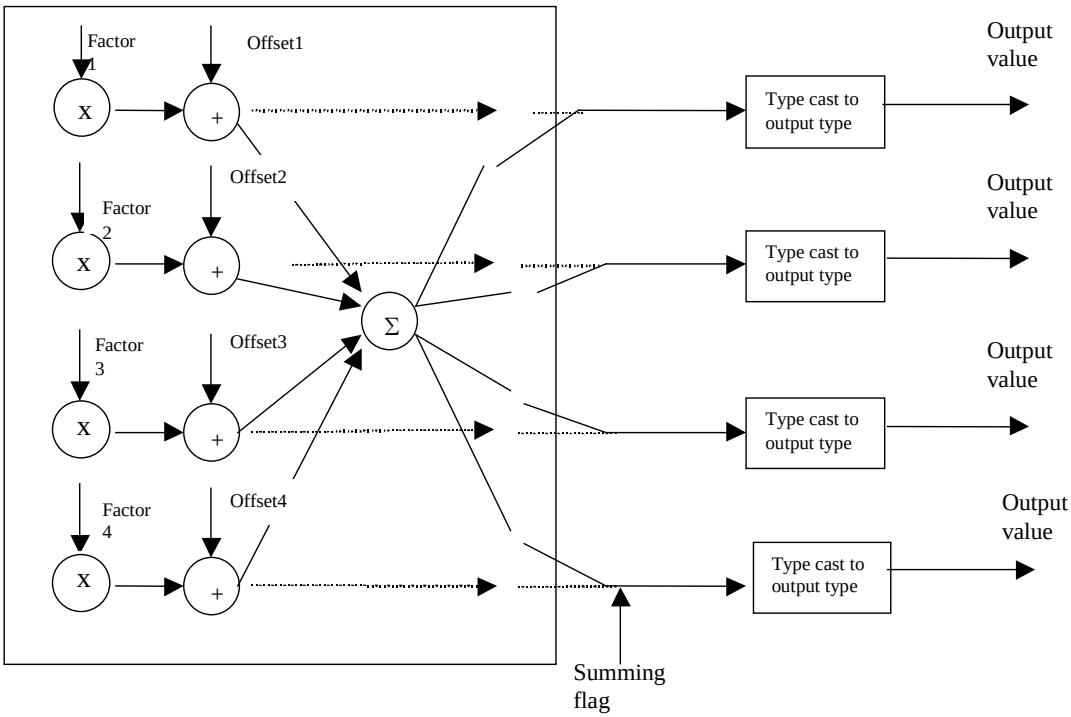


Figure 27 - Valuator functionaliy

Referring to the above figure, there are input paths each catering to an input value. Depending on the data type, there may be one to four input values. For example the SFRotation will require four input paths but the SFInt32 will only require the first input path. Each input path will operate identically.

Table 49 - 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

Table 50 - Simple typecasting conversion from float to other data types.

To	Conversion from float
integer	Truncate floating point. eg (1.11 to 1)
Boolean	0.0 to False Any other values to true
double	Direct conversion

Each input value is converted to a floating-point value using a simple typecasting rule as illustrated in Table 50. After conversion, the values are multiplied by the corresponding factor value and added to the corresponding offset value. Depending on whether the summer is enabled, either the summed value or the individual values are presented at the output.

Depending on the output data type required, the corresponding number of output values are retrieved and converted to the output types according to Table 49.

In the event that the input value is of a multi-valued type and the output is of a single value type, the first value of the multi-valued input is used.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000236

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**.

9.4.2.109 Viewpoint

9.4.2.109.1 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 Annex H.1.96.

9.4.2.109.2 Functionality and semantics

The semantics of the **Viewpoint** node are specified in ISO/IEC 14772-1:1998, subclause 6.53.

9.4.2.110 Viseme

9.4.2.110.1 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 Annex H.1.97.

9.4.2.110.2 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 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000237

9.4.2.111 VisibilitySensor

9.4.2.111.1 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 Annex H.1.98.

9.4.2.111.2 Functionality and semantics

The semantics of the **VisibilitySensor** node are specified in ISO/IEC 14772-1:1998, subclause 6.54.

9.4.2.112 WorldInfo

9.4.2.112.1 Node interface

```
WorldInfo {
    field           MFString    info            []
    field           SFString    title           ""
}
```

NOTE — For the binary encoding of this node see Annex H.1.99.

9.4.2.112.2 Functionality and semantics

The semantics of the **WorldInfo** node are specified in ISO/IEC 14772-1:1998, subclause 6.55.

10 Synchronization of Elementary Streams

10.1 Introduction

This subclause defines the tools to maintain temporal synchronisation within and among elementary streams. The conceptual elements that are required for this purpose, namely time stamps and clock reference information, have already been introduced in clause 7. The syntax and semantics to convey these elements to a receiving terminal are embodied in the sync layer, specified in 10.2. This syntax is configurable to adapt to the needs of different types of elementary streams. The required configuration information is specified in 10.2.3.

On the sync layer, an elementary stream is mapped into a sequence of packets, called an SL-packetized stream (SPS). Packetization information has to be exchanged between the entity that generates an elementary stream and the sync layer. This relation may be described by a conceptual elementary stream interface (ESI) between both layers (see Annex L). The ESI is a concept to explain the information flow between layers, however, need not be accessible in an implementation.

SL-packetized streams are conveyed through a delivery mechanism that is outside the scope of ISO/IEC 14496-1. This delivery mechanism is only described in terms of the DMIF Application Interface (DAI) whose semantics are specified in ISO/IEC 14496-6. It specifies the information that needs to be exchanged between the sync layer and the delivery mechanism. The basic data transport feature that this delivery mechanism shall provide is the framing of the data packets generated by the sync layer. The DAI is a reference point that need not be accessible in an implementation. The required properties of the DAI are described in 10.3.

The items specified in this clause are depicted in Figure 28 below.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000238

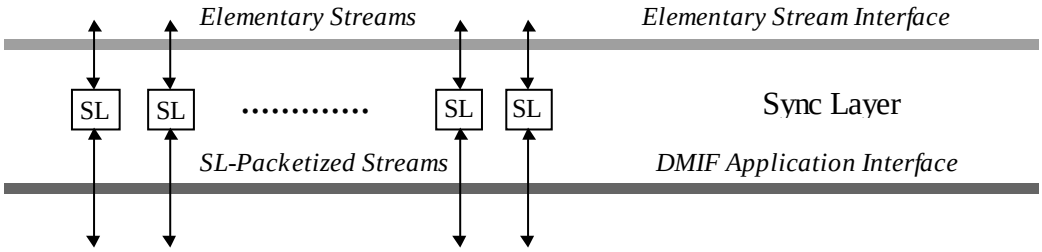


Figure 28 - The sync layer

10.2 Sync Layer

10.2.1 Overview

The sync layer (SL) specifies a syntax for the packetization of elementary streams into access units or parts thereof. Such a packet is called SL packet. The sequence of SL packets resulting from one elementary stream is called an SL-packetized stream (SPS). Access units are the only semantic entities at this layer that need to be preserved from end to end. Their content is opaque. Access units are used as the basic unit for synchronisation.

An SL packet consists of an SL packet header and an SL packet payload. The SL packet header provides means for continuity checking in case of data loss and carries the coded representation of the time stamps and associated information. The detailed semantics of the time stamps are specified in 7.3 that defines the timing aspects of the systems decoder model. The SL packet header is configurable as specified in 10.2.3. The SL packet header itself is specified in 10.2.4.

An SL packet does not contain an indication of its length. Therefore, SL packets must be framed by a suitable lower layer protocol using, e.g., the FlexMux tool specified in 12. Consequently, an SL-packetized stream is not a self-contained data stream that can be stored or decoded without such framing.

An SL-packetized stream does not provide identification of the ES_ID associated to the elementary stream (see 8.6.5) in the SL packet header. This association must be conveyed through a stream map table using the appropriate signalling means of the delivery mechanism.

10.2.2 SL Packet Specification

10.2.2.1 Syntax

```
class SL_Packet (SLConfigDescriptor SL) {
   aligned(8) SL_PacketHeader slPacketHeader(SL);
   aligned(8) SL_PacketPayload slPacketPayload;
}
```

10.2.2.2 Semantics

In order to properly parse an `SL_Packet`, it is required that the `SLConfigDescriptor` for the elementary stream to which the SL_Packet belongs is known, since the `SLConfigDescriptor` conveys the configuration of the syntax of the SL packet header.

`slPacketHeader` – an `SL_PacketHeader` element as specified in 10.2.4.

`slPacketPayload` – an `SL_PacketPayload` that contains an opaque payload.

10.2.3 SL Packet Header Configuration

10.2.3.1 Syntax

```
class SLConfigDescriptor extends BaseDescriptor : bit(8) tag=SLConfigDescrTag {
   bit(8) predefined;
   if (predefined==0) {
      bit(1) useAccessUnitStartFlag;
      bit(1) useAccessUnitEndFlag;
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000239

```
      bit(1) useRandomAccessPointFlag;
      bit(1) hasRandomAccessUnitsOnlyFlag;
      bit(1) usePaddingFlag;
      bit(1) useTimeStampsFlag;
      bit(1) useIdleFlag;
      bit(1) durationFlag;
      bit(32) timeStampResolution;
      bit(32) OCRResolution;
      bit(8) timeStampLength; // must be ≤ 64
      bit(8) OCRLength;    // must be ≤ 64
      bit(8) AU_Length;    // must be ≤ 32
      bit(8) instantBitrateLength;
      bit(4) degradationPriorityLength;
      bit(5) AU_seqNumLength; // must be ≤ 16
      bit(5) packetSeqNumLength; // must be ≤ 16
      bit(2) reserved=0b11;
   }
   if (durationFlag) {
      bit(32) timeScale;
      bit(16) accessUnitDuration;
      bit(16) compositionUnitDuration;
   }
   if (!useTimeStampsFlag) {
      bit(timeStampLength) startDecodingTimeStamp;
      bit(timeStampLength) startCompositionTimeStamp;
   }
}
```

10.2.3.2 Semantics

The SL packet header may be configured according to the needs of each individual elementary stream. Parameters that can be selected include the presence, resolution and accuracy of time stamps and clock references. This flexibility allows, for example, a low bitrate elementary stream to incur very little overhead on SL packet headers.

For each elementary stream the configuration is conveyed in an `SLConfigDescriptor`, which is part of the associated `ES_Descriptor` within an object descriptor.

The configurable parameters in the SL packet header can be divided in two classes: those that apply to each SL packet (e.g. OCR, sequenceNumber) and those that are strictly related to access units (e.g. time stamps, accessUnitLength, instantBitrate, degradationPriority).

`predefined` – allows to default the values from a set of predefined parameter sets as detailed below.

NOTE — This table will be updated by amendments to ISO/IEC 14496 to include predefined configurations as required by future profiles.

Table 51 - Overview of predefined `SLConfigDescriptor` values

`Predefined` field value	**Description**
0x00	Custom
0x01	null SL packet header
0x02	Reserved for use in MP4 files
0x03 – 0xFF	Reserved for ISO use

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000240

Table 52 – Detailed predefined SLConfigDescriptor values

Predefined field value	0x01	0x02
UseAccessUnitStartFlag	0	0
UseAccessUnitEndFlag	0	0
UseRandomAccessPointFlag	0	0
UsePaddingFlag	0	0
UseTimeStampsFlag	0	1
UseIdleFlag	0	0
DurationFlag	-	0
TimeStampResolution	1000	-
OCRResolution	-	-
TimeStampLength	32	0
OCRlength	-	0
AU_length	0	0
InstantBitrateLength	-	0
DegradationPriorityLength	0	0
AU_seqNumLength	0	0
PacketSeqNumLength	0	0
TimeScale	-	-
AccessUnitDuration	-	-
CompositionUnitDuration	-	-
StartDecodingTimeStamp	-	-
StartCompositionTimeStamp	-	-

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

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

If neither useAccessUnitStartFlag nor useAccessUnitEndFlag are set this implies that each SL packet corresponds to a complete access unit.

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

hasRandomAccessUnitsOnlyFlag – indicates that each SL packet corresponds to a random access point. In that case the randomAccessPointFlag need not be used.

usePaddingFlag – indicates that the paddingFlag is present in each SL packet header of this elementary stream.

useTimeStampsFlag – indicates that time stamps are used for synchronisation of this elementary stream. They are conveyed in the SL packet headers. Otherwise, the parameters accessUnitRate, compositionUnitRate, startDecodingTimeStamp and startCompositionTimeStamp conveyed in this SL packet header configuration shall be used for synchronisation.

NOTE — The use of start time stamps and durations (useTimeStampsFlag=0) may only be feasible under some conditions, including an error free environment. Random access is not easily possible.

useIdleFlag – indicates that idleFlag is used in this elementary stream.

durationFlag – indicates that the constant duration of access units and composition units for this elementary stream is subsequently signaled.

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

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000241

OCRResolution – is the resolution of the object time base in cycles per second.

timeStampLength – is the length of the time stamp fields in SL packet headers. timeStampLength shall take values between zero and 64 bit.

OCRlength – is the length of the objectClockReference field in SL packet headers. A length of zero indicates that no objectClockReferences are present in this elementary stream. If OCRstreamFlag is set, OCRLength shall be zero. Else OCRlength shall take values between zero and 64 bit.

AU_Length – is the length of the accessUnitLength fields in SL packet headers for this elementary stream. AU_Length shall take values between zero and 32 bit.

instantBitrateLength – is the length of the instantBitrate field in SL packet headers for this elementary stream.

degradationPriorityLength – is the length of the degradationPriority field in SL packet headers for this elementary stream.

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

packetSeqNumLength – is the length of the packetSequenceNumber field in SL packet headers for this elementary stream.

timeScale – used to express the duration of access units and composition units. One second is evenly divided in timeScale parts.

accessUnitDuration – the duration of an access unit is accessUnitDuration * 1/timeScale seconds.

compositionUnitDuration – the duration of a composition unit is compositionUnitDuration * 1/timeScale seconds.

startDecodingTimeStamp – conveys the time at which the first access unit of this elementary stream shall be decoded. It is conveyed in the resolution specified by timeStampResolution.

startCompositionTimeStamp – conveys the time at which the composition unit corresponding to the first access unit of this elementary stream shall be decoded. It is conveyed in the resolution specified by timeStampResolution.

10.2.4 SL Packet Header Specification

10.2.4.1 Syntax

```
aligned(8) class SL_PacketHeader (SLConfigDescriptor SL) {
   if (SL.useAccessUnitStartFlag)
      bit(1) accessUnitStartFlag;
   if (SL.useAccessUnitEndFlag)
      bit(1) accessUnitEndFlag;
   if (SL.OCRLength>0)
      bit(1) OCRflag;
   if (SL.useIdleFlag)
      bit(1) idleFlag;
   if (SL.usePaddingFlag)
      bit(1) paddingFlag;
   if (paddingFlag)
      bit(3) paddingBits;

   if (!idleFlag && (!paddingFlag || paddingBits!=0)) {
      if (SL.packetSeqNumLength>0)
         bit(SL.packetSeqNumLength) packetSequenceNumber;
      if (SL.degradationPriorityLength>0)
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000242

```
      bit(1) DegPrioflag;
   if (DegPrioflag)
      bit(SL.degradationPriorityLength) degradationPriority;
   if (OCRflag)
      bit(SL.OCRLength) objectClockReference;

   if (accessUnitStartFlag) {
      if (SL.useRandomAccessPointFlag)
         bit(1) randomAccessPointFlag;
      if (SL.AU_seqNumLength >0)
         bit(SL.AU_seqNumLength) AU_sequenceNumber;
      if (SL.useTimeStampsFlag) {
         bit(1) decodingTimeStampFlag;
         bit(1) compositionTimeStampFlag;
      }
      if (SL.instantBitrateLength>0)
         bit(1) instantBitrateFlag;
      if (decodingTimeStampFlag)
         bit(SL.timeStampLength) decodingTimeStamp;
      if (compositionTimeStampFlag)
         bit(SL.timeStampLength) compositionTimeStamp;
      if (SL.AU_Length > 0)
         bit(SL.AU_Length)    accessUnitLength;
      if (instantBitrateFlag)
         bit(SL.instantBitrateLength)      instantBitrate;
   }
  }
}
```

10.2.4.2 Semantics

`accessUnitStartFlag` – when set to one indicates that the first byte of the payload of this SL packet is the start of an access unit. If this syntax element is omitted from the SL packet header configuration its default value is known from the previous SL packet with the following rule:

`accessUnitStartFlag` = (previous-SL packet has `accessUnitEndFlag`==1) ? 1 : 0.

`accessUnitEndFlag` – when set to one indicates that the last byte of the SL packet payload is the last byte of the current access unit. If this syntax element is omitted from the SL packet header configuration its default value is only known after reception of the subsequent SL packet with the following rule:

`accessUnitEndFlag` = (subsequent-SL packet has `accessUnitStartFlag`==1) ? 1 : 0.

If neither `AccessUnitStartFlag` nor `AccessUnitEndFlag` are configured into the SL packet header this implies that each SL packet corresponds to a single access unit, hence both `accessUnitStartFlag` = `accessUnitEndFlag` = 1.

NOTE — When the SL packet header is configured to use `accessUnitStartFlag` but neither `accessUnitEndFlag` nore `accessUnitLength`, it is not guaranteed that the terminal can determine the end of an access unit before the subsequent one is received.

`OCRflag` – when set to one indicates that an `objectClockReference` will follow. The default value for `OCRflag` is zero.

`idleFlag` – indicates that this elementary stream will be idle (i.e., not produce data) for an undetermined period of time. This flag may be used by the decoder to discriminate between deliberate and erroneous absence of subsequent SL packets.

`paddingFlag` – indicates the presence of padding in this SL packet. The default value for `paddingFlag` is zero.

`paddingBits` – indicate the mode of padding to be used in this SL packet. The default value for `paddingBits` is zero.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000243

If paddingFlag is set and paddingBits is zero, this indicates that the subsequent payload of this SL packet consists of padding bytes only. accessUnitStartFlag, randomAccessPointFlag and OCRflag shall not be set if paddingFlag is set and paddingBits is zero.

If paddingFlag is set and paddingBits is greater than zero, this indicates that the payload of this SL packet is followed by paddingBits of zero stuffing bits for byte alignment of the payload.

packetSequenceNumber – if present, it shall be continuously incremented for each SL packet as a modulo counter. A discontinuity at the decoder corresponds to one or more missing SL packets. In that case, an error shall be signalled to the sync layer user. If this syntax element is omitted from the SL packet header configuration, continuity checking by the sync layer cannot be performed for this elementary stream.

Duplication of SL packets: elementary streams that have a sequenceNumber field in their SL packet headers may use duplication of SL packets for error resilience. The duplicated SL packet(s) shall immediately follow the original. The packetSequenceNumber of duplicated SL packets shall have the same value and each byte of the original SL packet shall be duplicated, with the exception of an objectClockReference field, if present, which shall encode a valid value for the duplicated SL packet.

degPrioFlag - when set to one indicates that degradationPriority is present in this packet.

degradationPriority – indicates the importance of the payload of this SL packet. The streamPriority defines the base priority of an ES. degradationPriority defines a decrease in priority for this SL packet relative to the base priority. The priority for this SL packet is given by:

SL_PacketPriority = streamPriority – degradationPriority

degradationPriority remains at this value until its next occurrence. This indication may be for graceful degradation by the decoder of this elementary stream as well as by the adaptor to a specific delivery layer instance. The relative amount of complexity degradation among SL packets of different elementary streams increases as SL_PacketPriority decreases.

objectClockReference – contains an Object Clock Reference time stamp. The OTB time value t is reconstructed from this OCR time stamp according to the following formula:

$$t = (\texttt{objectClockReference}/\texttt{SL.OCRResolution}) + k*(2^{\texttt{SL.OCRLength}}/\texttt{SL.OCRResolution})$$

where k is the number of times that the objectClockReference counter has wrapped around.

objectClockReference is only present in the SL packet header if OCRflag is set.

NOTE — It is possible to convey just an OCR value and no payload within an SL packet.

The following is the semantics of the syntax elements that are only present at the start of an access unit when explicitly signaled by accessUnitStartFlag in the bitstream:

randomAccessPointFlag – when set to one indicates that random access to the content of this elementary stream is possible here. randomAccessPointFlag shall only be set if accessUnitStartFlag is set. If this syntax element is omitted from the SL packet header configuration, its default value is the value of SLConfigDescriptor.hasRandomAccessUnitsOnlyFlag for this elementary stream.

AU_sequenceNumber – if present, it shall be continuously incremented for each access unit as a modulo counter. A discontinuity at the decoder corresponds to one or more missing access units. In that case, an error shall be signalled to the sync layer user. If this syntax element is omitted from the SL packet header configuration, access unit continuity checking by the sync layer cannot be performed for this elementary stream.

Duplication of access units: elementary streams that have a AU_sequenceNumber field in their SL packet headers may use duplication of access units. The duplicated access unit(s) shall immediately follow the original. The AU_sequenceNumber of such access units shall have the same value and each byte of the original one or more SL packets shall be duplicated, with the exception of an objectClockReference field, if present, which shall encode a valid value for the duplicated access unit.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000244

decodingTimeStampFlag – indicates that a decoding time stamp is present in this packet.

compositionTimeStampFlag – indicates that a composition time stamp is present in this packet.

accessUnitLengthFlag – indicates that the length of this access unit is present in this packet.

instantBitrateFlag – indicates that an instantBitrate is present in this packet.

decodingTimeStamp – is a decoding time stamp as configured in the associated SLConfigDescriptor. The decoding time td of this access unit is reconstructed from this decoding time stamp according to the formula:

$$td = (decodingTimeStamp/SL.timeStampResolution + k * 2^{SL.timeStampLength}/SL.timeStampResolution$$

where k is the number of times that the decodingTimeStamp counter has wrapped around.

A decodingTimeStamp shall only be present if the decoding time is different from the composition time for this access unit.

compositionTimeStamp – is a composition time stamp as configured in the associated SLConfigDescriptor. The composition time tc of the first composition unit resulting from this access unit is reconstructed from this composition time stamp according to the formula:

$$td = (compositionTimeStamp/SL.timeStampResolution + k * 2^{SL.timeStampLength}/SL.timeStampResolution$$

where k is the number of times that the compositionTimeStamp counter has wrapped around.

accessUnitLength – is the length of the access unit in bytes. If this syntax element is not present or has the value zero, the length of the access unit is unknown.

instantBitrate – is the instantaneous bit rate in bits per second of this elementary stream until the next instantBitrate field is found.

10.2.5 Clock Reference Stream

An elementary stream of streamType = ClockReferenceStream may be declared by means of the object descriptor. It is used for the sole purpose of conveying Object Clock Reference time stamps. Multiple elementary streams in a name scope may make reference to such a ClockReferenceStream by means of the OCR_ES_ID syntax element in the SLConfigDescriptor to avoid redundant transmission of Clock Reference information. Note, however, that circular references between elementary streams using OCR_ES_ID are not permitted.

On the sync layer a ClockReferenceStream is realized by configuring the SL packet header syntax for this SL-packetized stream such that only OCR values of the required OCRresolution and OCRlength are present in the SL packet header.

There shall not be any SL packet payload present in an SL-packetized stream of streamType = ClockReferenceStream.

In the DecoderConfigDescriptor for a clock reference stream ObjectTypeIndication shall be set to '0xFF', hasRandomAccessUnitsOnlyFlag to one and bufferSizeDB to '0'.

The following indicates recommended values for the SLConfigDescriptor of a Clock Reference Stream:

Table 53 – SLConfigDescriptor parameter values for a ClockReferenceStream

useAccessUnitStartFlag	0
useAccessUnitEndFlag	0
useRandomAccessPointFlag	0
usePaddingFlag	0

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000245

useTimeStampsFlag	0
useIdleFlag	0
durationFlag	0
timeStampResolution	0
timeStampLength	0
AU_length	0
degradationPriorityLength	0
AU_seqNumLength	0

10.2.6 Restrictions for elementary streams sharing the same object time base

While it is possible to share an object time base between multiple elementary streams through OCR_ES_ID, a number of restrictions for the access to and processing of these elementary streams exist as follows:

1. When several elementary streams share a single object time base, the elementary streams without embedded object clock reference information shall not be used by the player, even if accessible, until the elementary stream carrying the object clock reference information becomes accessible (see 8.7.3 for the stream access procedure).

2. If an elementary stream without embedded object clock reference information is made available to the terminal at a later point in time than the elementary stream carrying the object clock reference information, it shall be delivered in synchronization with the other stream(s). Note that this implies that such a stream might not start playing from its beginning, depending on the current value of the object time base.

3. When an elementary stream carrying object clock reference information becomes unavailable or is otherwise manipulated in its delivery (e.g., paused), all other elementary streams which use the same object time base shall follow this behavior, i.e., become unavailable or be manipulated in the same way.

4. When an elementary stream without embedded object clock reference information becomes unavailable this has no influence on the other elementary streams that share the same object time base.

10.2.7 Usage of configuration options for object clock reference and time stamp values

10.2.7.1 Resolution of ambiguity in object time base recovery

Due to the limited length of objectClockReference values these time stamps may be ambiguous. The OTB time value can be reconstructed each time an objectClockReference is transmitted in the headers of an SL packet according to the following formula:

$t_{OTB\_reconstructed}$=(objectClockReference/SL.OCRResolution)+k*($2^{SL.OCRLength}$/SL.OCRResolution)

with k being an integer value denoting the number of wrap-arounds. The resulting time base $t_{OTB\_reconstructed}$ is measured in seconds.

When the first objectClockReference for an elementary stream is acquired, the value k shall be set to one. For each subsequent occurence of objectClockReference the value k is estimated as follows:

The terminal shall implement a mechanism to estimate the value of the object time base for any time instant.

Each time an objectClockReference is received, the current estimated value of the OTB $t_{OTB\_estimated}$ shall be sampled. Then, $t_{OTB\_rec}(k)$ is evaluated for different values of k. The value k that minimizes the term $| t_{OTB\_estimated} - t_{OTB\_rec}(k)|$ shall be assumed to yield the correct value of $t_{OTB\_reconstructed}$. This value may be used as new input to the object time base estimation mechanism.

The application shall ensure that this procedure yields an unambiguous value of k by selecting an appropriate length and resolution of the objectClockReference element and a sufficiently high frequency of insertion of objectClockReference values in the elementary stream. The choices for these values depend on the delivery jitter for SL packets as well as the anticipated maximum drift between the clocks of the transmitting and receiving terminal.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000246

10.2.7.2 Resolution of ambiguity in time stamp recovery

Due to the limited length of `decodingTimeStamp` and `compositionTimeStamp` values these time stamps may become ambiguous according to the following formula:

$t_{ts}(m)=(\texttt{TimeStamp/SL.timeStampResolution})+m*(2^{\texttt{SL.timeStampLength}}/\texttt{SL.timeStampResolution})$

with `TimeStamp` being either a `decodingTimeStamp` or a `compositionTimeStamp` and m being an integer value denoting the number of wrap-arounds.

The correct value $t_{timestamp}$ of the time stamp can be estimated as follows:

Each time a `TimeStamp` is received, the current estimated value of the OTB $t_{OTB\_estimated}$ shall be sampled. $t_{ts}(m)$ is evaluated for different values of m. The value m that minimizes the term $| t_{OTB\_estimated} - t_{ts}(m)|$ shall be assumed to yield the correct value of $t_{timestamp}$.

The application may choose, separately for every individual elementary stream, the length and resolution of time stamps so as to match its requirements on unambiguous positioning of time events. This choice depends on the maximum time that an SL packet with a `TimeStamp` may be sent prior to the point in time indicated by the `TimeStamp` as well as the required precision of temporal positioning.

10.2.7.3 Usage considerations for object clock references and time stamps

The time line of an object time base allows to discriminate two time instants separated by more than 1/`SL.OCRResolution`. `OCRResolution` should be chosen sufficiently high to match the accuracy needed by the application to synchronize a set of elementary streams.

The decoding and composition time stamp allow to discriminate two time instants separated by more than 1/`SL.timeStampResolution`. `timeStampResolution` should be chosen sufficiently high to match the accuracy needed by the application in terms of positioning of access units for a given elementary stream.

A `TimeStampResolution` higher than the `OCRResolution` will not achieve better discrimination between events. If `TimeStampResolution` is lower than the `OCRResolution`, events for this specific stream cannot be positioned with the maximum precision possible with this given `OCRResolution`.

The parameter `OCRLength` is signaled in the SL header configuration. $2^{\texttt{SL.OCRLength}}/\texttt{SL.OCRResolution}$ is the time interval covered by the `objectClockReference` counter before it wraps around. `OCRLength` should be chosen sufficiently high to match the application needs for unambiguous positioning of time events from a set of elementary streams.

When an application knows the value k defined in 10.2.7.1, the OTB time line is unambiguous for any time value. When the application cannot reconstruct the k factor, as for example in any application that permits random access without additional side information, the OTB time line is ambiguous modulo $2^{\texttt{SL.OCRLength}}/\texttt{SL.OCRResolution}$. Therefore, any time stamp refering to this OTB is ambiguous. Therefore, any time stamp refering to this OTB is ambiguous. It may, however, be considered unambiguous within an application environment through knowledge about the maximum expected delivery jitter and constraints on the time by which an access unit can be sent prior to its decoding time.

Note that elementary streams that choose the time interval $2^{\texttt{SL.timeStampLength}}/\texttt{SL.timeStampResolution}$ higher than $2^{\texttt{SL.OCRLength}}/\texttt{SL.OCRResolution}$ can still only position time events unambiguously in the smaller of the two intervals.

In cases, where k and m can not be estimated correctly, the buffer model may be violated, resulting in unpredictable performance of the decoder.

EXAMPLE — Let's assume an application that wants to synchronize elementary streams with a precision of 1 ms. `OCRResolution` should be chosen equal to or higher than 1000 (the time between two successive ticks of the OCR is then equal to 1ms). Let's assume `OCRResolution=2000`.

The application assumes a drift between the STB and the OTB of 0.1% (i.e. 1ms every second). The clocks need therefore to be adjusted at least every second (i.e. in the worst case, the clocks will have drifted 1ms which is the precision constraint). Let's assume that `objectClockReference` are sent every 1s.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000247

The application wants to have an unambiguous OTB time line of 24h without need to reconstruct the k factor. The `OCRLength` is therefore chosen accordingly such that $2^{SL.OCRLength}$/`SL.OCRResolution=24h`.

Let's assume now that the application wants to synchronize events within a single elementary stream with a precision of 10 ms. `TimeStampResolution` should be chosen equal to or higher than 100 (the time between two successive ticks of the `TimeStamp` is then equal to 10ms). Let's assume `TimeStampResolution=200`.

The application wants to be able to send access units at maximum 1 minute ahead of their decoding or composition time. The `timeStampLength` is therefore chosen as

$2^{SL.timeStampLength}$/`SL.timeStampResolution` = 2 minutes.

10.3 DMIF Application Interface

The DMIF Application Interface is a conceptual interface that specifies which data need to be exchanged between the sync layer and the delivery mechanism. Communication between the sync layer and the delivery mechanism includes SL-packetized data as well as additional information to convey the length of each SL packet.

An implementation of ISO/IEC 14496-1 does not have to expose the DMIF Application Interface. A terminal compliant with ISO/IEC 14496-1, however, shall have the functionality described by the DAI to be able to receive the SL packets that constitute an SL-packetized stream. Specifically, the delivery mechanism below the sync layer shall supply a method to frame or otherwise encode the length of the SL packets transported through it.

The DMIF Application Interface specified in ISO/IEC 14496-6 embodies a superset of the required data delivery functionality. The DAI has data primitives to receive and send data, which include indication of the data size. With this interface, each invocation of a DA_Data or a DA_DataCallback shall transfer one SL packet between the sync layer and the delivery mechanism below.

11 MPEG-J

11.1 Introduction

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 identified, 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.

This part of the document specifies the format, delivery, interactions and behavior of Java byte code to the MPEG-4 player. More specifically:

- The format and delivery are normatively specified by specifying the MPEG-J stream format and the delivery mechanism of such a stream (Java byte code and associated data).
- MPEG-J Session and the application lifecycle
- The interactions and behavior of byte code is normatively defined through the Java APIs.

11.1.1 Organization of this document

Subclause 0 gives an overall architecture of the MPEG-J system. MPEG-J Session start up is walked through in subclause 11.3. The Delivery of MPEG-J data to the terminal is specified in subclause 11.4. Subclause 11.5 specifies the different categories of APIs that a program in the form of Java bytecode would use. **Annex S** is an informative annex on starting the Java Virtual Machine. **Annex V** of this document gives normative javadoc listings of the MPEG-J APIs in the word97 format, while **Annex U** gives the same in the HTML format. **Annex T** illustrates the usage of MPEG-J APIs through a few examples.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000248

11.2 Architecture

11.2.1 Parametric versus Programmatic System

11.2.1.1 Overview of a Parametric MPEG-4 System

Figure 29 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, FlexMux-PDU's 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.

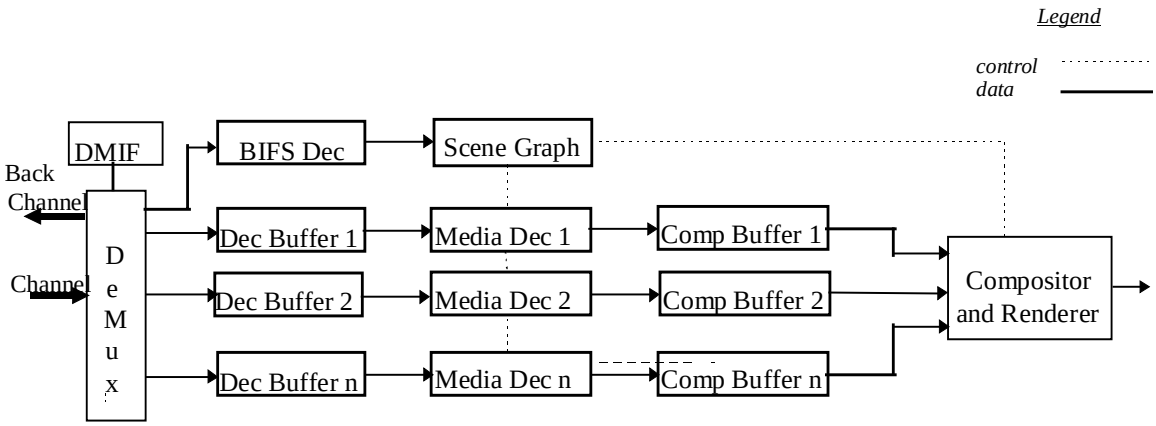


Figure 29 - An MPEG-4 Player

11.2.2 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 30 shows an example of the components of the MPEG-J operating environment.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000249

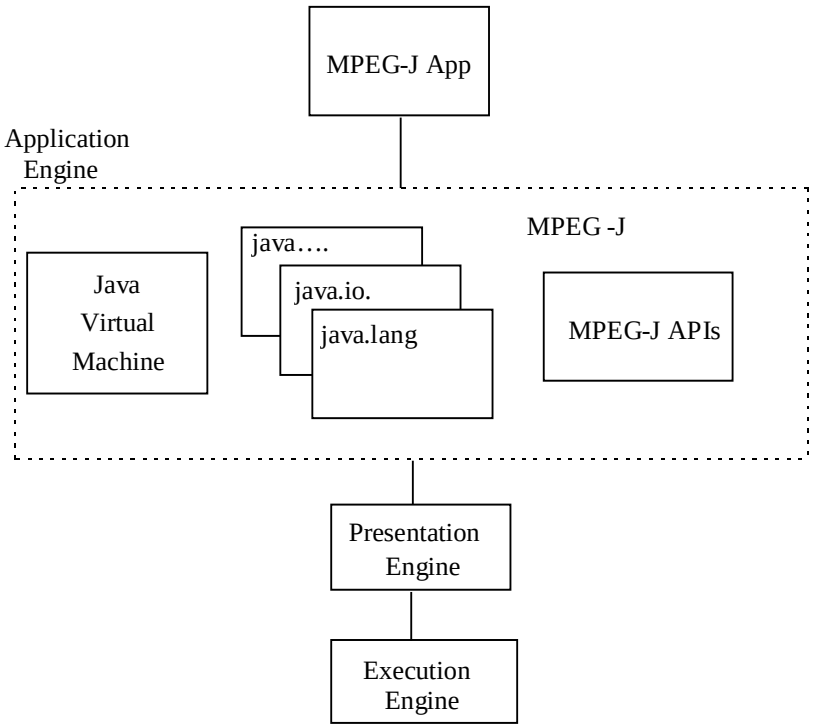


Figure 30 - MPEG-J Software architecture

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 31. The lower half of this drawing depicts the parametric MPEG-4 player of Figure 1 and is also referred to as the Presentation Engine. The upper half of Figure 31 illustrates this the MPEG-J system controlling the Presentation Engine is also referred to as the Application Engine.. The APIs shown in Figure 31 will be specified later in this document.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000250

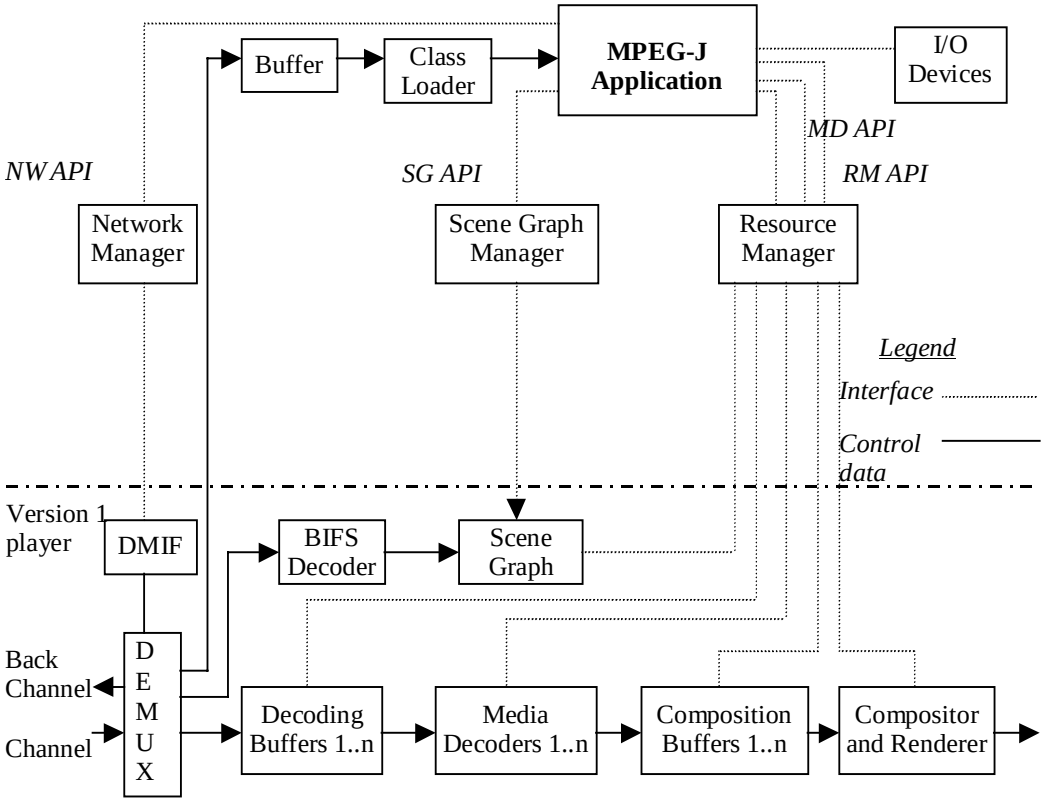


Figure 31 - Architecture of an MPEG-J enabled MPEG-4 System

11.3 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.

11.3.1 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:

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.

11.3.2 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.

11.3.3 MPEG-J Elementary Stream, Object Descriptor, and the Name Scope

11.3.3.1 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

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000251

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 9 of this document.

11.3.3.2 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.

11.3.3.3 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.

11.3.4 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 section 11.4.2. 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.

11.3.5 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.

11.4 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.

11.4.1 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 FlexMux, RTP, and even MPEG2 transport stream, can be used to transport the data to the client side.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000252

11.4.1.1 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.

11.4.1.2 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.

11.4.1.3 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.[2)]

11.4.1.4 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.

11.4.2 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).

[2)] 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 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000253

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.

11.4.3 Streaming Header

11.4.3.1 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.

11.4.3.2 JavaStreamHeader

11.4.3.2.1 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];

}
```

11.4.3.2.2 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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000254

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.

11.4.3.3 JavaClassID

11.4.3.3.1 Syntax

```
aligned(32) class JavaClassID {
   bit(16) length;
   bit(8 * length) ID;
}
```

11.4.3.3.2 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.

11.5 MPEG-J API List

11.5.1 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 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000255

Table 54 - 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.

11.5.2 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.

11.5.3 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.

11.5.3.1 Events

Events in the BIFS scene graph are identified by the two interface classes EventIn and EventOut. The EventOutListener class can monitor them.

11.5.3.1.1 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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000256

11.5.3.1.2 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.

11.5.3.1.3 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.

11.5.3.2 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.

11.5.3.3 Scene Management

The following interfaces are used to facilitate programmatic control over the MPEG-4 terminal's native scene.

11.5.3.3.1 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.

11.5.3.3.2 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.

11.5.3.3.2.1 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.

11.5.3.3.2.2 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).

11.5.3.3.2.3 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 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000257

- 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.

11.5.4 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.

11.5.4.1 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

11.5.4.1.1 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.

11.5.4.2 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

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000258

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.

11.5.4.3 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.

11.5.5 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000259

11.5.6 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 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.

11.5.7 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.

11.5.7.1 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.

11.5.7.2 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.

11.5.7.3 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.

11.5.7.4 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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000260

11.5.8 Detailed API Listing

The normative detailed API listing can be found in the Annexes to this document:

Annex U (normative): The HTML files (javadocs) of all the APIs defined in this document along with the necessary images in gif format.

Annex V (normative): Integrated API document (javadocs) of all the html files. This contains the API specification of all the APIs defined in this document.

12 Multiplexing of Elementary Streams

12.1 Introduction

Elementary stream data encapsulated in SL-packetized streams are sent/received through the DMIF Application Interface, as specified in clause 10. Multiplexing procedures and the architecture of the delivery protocol layers are outside the scope of ISO/IEC 14496-1. However, care has been taken to define the sync layer syntax and semantics such that SL-packetized streams can be easily embedded in various transport protocol stacks.

The analysis of existing transport protocol stacks has shown that, for stacks with fixed length packets (e.g., MPEG-2 Transport Stream) or with high multiplexing overhead (e.g., RTP/UDP/IP), it may be advantageous to have a generic, low complexity multiplexing tool that allows interleaving of data with low overhead and low delay. This is particularly important for low bit rate applications. Such a multiplex tool is specified in this clause. Its use is optional.

12.2 FlexMux Tool

12.2.1 Overview

The FlexMux tool is a flexible multiplexer that accommodates interleaving of SL-packetized streams with varying instantaneous bit rate. The basic data entity of the FlexMux is a FlexMux packet, which has a variable length. One or more SL packets are embedded in a FlexMux packet as specified in detail in the remainder of this clause. The FlexMux tool provides identification of SL packets originating from different elementary streams by means of FlexMux Channel numbers. Each SL-packetized stream is mapped into one FlexMux Channel. FlexMux packets with data from different SL-packetized streams can therefore be arbitrarily interleaved. The sequence of FlexMux packets that are interleaved into one stream are called a FlexMux Stream.

A FlexMux Stream retrieved from storage or transmission may be parsed as a single data stream. However, framing of FlexMux packets by the underlying layer is required for random access or error recovery. There is no requirement to frame each individual FlexMux packet. The FlexMux also requires reliable error detection by the underlying layer. This design has been chosen acknowledging the fact that framing and error detection mechanisms are in many cases provided by the transport protocol stack below the FlexMux.

Two different modes of operation of the FlexMux providing different features and complexity are defined. They are called Simple Mode and MuxCode Mode. A FlexMux Stream may contain an arbitrary mixture of FlexMux packets using either Simple Mode or MuxCode Mode. The syntax and semantics of both modes are specified below.

The delivery timing of the FlexMux Stream can be conveyed by means of FlexMux clock reference time stamps. This functionality may be used to establish a multiplex buffer model on the delivery layer. Both the time stamps and the MuxCode Mode require out-of-band configuration prior to usage.

12.2.2 Simple Mode

In the simple mode one SL packet is encapsulated in one FlexMux packet and tagged by an `index` which is equal to the FlexMux Channel number as indicated in Figure 32. This mode does not require any configuration or maintenance of state by the receiving terminal.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000261

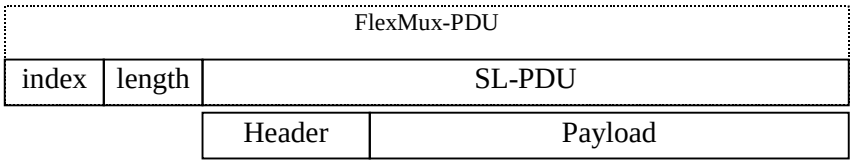


Figure 32 - Structure of FlexMux packet in simple mode

12.2.3 MuxCode mode

In the MuxCode mode one or more SL packets are encapsulated in one FlexMux packet as indicated in Figure 33. This mode requires configuration and maintenance of state by the receiving terminal. The configuration describes how FlexMux packets are shared between multiple SL packets. In this mode the `index` value is used to dereference configuration information that defines the allocation of the FlexMux packet payload to different FlexMux Channels.

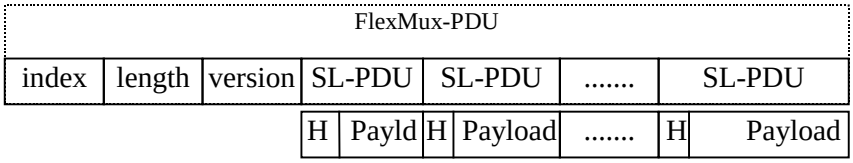


Figure 33 - Structure of FlexMux packet in MuxCode mode

12.2.4 FlexMux packet specification

12.2.4.1 Syntax

```
class FlexMuxPacket (MuxCodeTableEntry mct[], FlexMuxTimingDescriptor FM) {
   unsigned int(8) index;
   bit(8) length;
   if (index<238) {
      SL_Packet sPayload;
   } else if (index == 238) {
      bit(FM.FCR_Length) fmxClockReference;
      bit(FM.fmxRateLength) fmxRate;
   } else if (index == 239) {
      bit(8) stuffing[length];
   } else {
      bit(4) version;
      const bit(4) reserved=0b1111;
      multiple_SL_Packet mPayload(mct[index-240]);
   }
}
```

12.2.4.2 Semantics

The two modes of the FlexMux, Simple Mode and MuxCode Mode as well as special time stamp and stuffing packets are distinguished by the value of `index` as specified below.

`index` – if `index` is smaller than 238 then

FlexMux Channel = `index`

This range of values corresponds to the Simple Mode.

An `index` value of 238 indicates a FlexMux packet with clock reference information.

An `index` value of 239 indicates a FlexMux packet with stuffing.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000262

An index value in the range of 240 to 255 (inclusive) indicates that the MuxCode Mode is used and a MuxCode is referenced as

MuxCode = index - 240

MuxCode is used to associate the payload to FlexMux Channels as described in the Section 12.2.3.

NOTE — Although the number of FlexMux Channels is limited to 256, the use of multiple FlexMux streams allows virtually any number of elementary streams to be provided to the terminal.

length – the length of the FlexMux packet payload in bytes. This is equal to the length of the single encapsulated SL packet in Simple Mode and to the total length of the multiple encapsulated SL packets in MuxCode Mode.

version – indicates the current version of the MuxCodeTableEntry referenced by MuxCode. Version is used for error resilience purposes. If this version does not match the version of the referenced MuxCodeTableEntry that has most recently been received, the FlexMux packet cannot be parsed. The implementation is free to either wait until the required version of MuxCodeTableEntry becomes available or to discard the FlexMux packet.

sPayload – a single SL packet (Simple Mode).

mPayload – one or more SL packets that are identified using the specification of the associated MuxCodeTableEntry[index-240] (MuxCode Mode).

fmxClockReference – contains a Clock Reference time stamp for the FlexMux stream. The OTB time value t is reconstructed from this Clock Reference time stamp according to the following formula:

$$t = (\text{fmxClockReference}/\text{FM.FCRResolution}) + k*(2^{\text{FM.FCRLength}}/\text{FM.FCRResolution})$$

where k is the number of times that the fmxClockReference counter has wrapped around.

fmxRate - is the instant rate at which data from this FlexMux stream is delivered to the associated FlexMux buffers. The rate defined by fmxRate applies to all bytes in the FlexMux Clock Reference channel packet and each following FlexMux packet until the occurrence of the next FlexMux Clock Reference channel packet.

stuffing – one or more stuffing bytes that shall be discarded by the demultiplexer.

12.2.5 Configuration and usage of MuxCode Mode

12.2.5.1 Syntax

```
aligned(8) class MuxCodeTableEntry {
   int    i, k;
   bit(8) length;
   bit(4) MuxCode;
   bit(4) version;
   bit(8) substructureCount;
   for (i=0; i<substructureCount; i++) {
      bit(5) slotCount;
      bit(3) repetitionCount;
      for (k=0; k<slotCount; k++){
         bit(8) flexMuxChannel[[i]][[k]];
         bit(8) numberOfBytes[[i]][[k]];
      }
   }
}
```

12.2.5.2 Semantics

The configuration for MuxCode Mode is signaled by MuxCodeTableEntry messages. The transport of the MuxCodeTableEntry shall be defined during the design of the transport protocol stack that makes use of the

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000263

FlexMux tool. Part 6 of this Final Committee Draft of International Standard defines a method to convey this information using the DN_TransmuxConfig primitive.

The basic requirement for the transport of the configuration information is that data arrives reliably in a timely manner. However, no specific performance bounds are required for this control channel since version numbers allow to detect FlexMux packets that cannot currently be decoded and, hence, trigger suitable action in the receiving terminal.

`length` – the length in bytes of the remainder of the `MuxCodeTableEntry` following the `length` element.

`MuxCode` – the number through which this MuxCode table entry is referenced.

`version` – indicates the version of the `MuxCodeTableEntry`. Only the latest received version of a `MuxCodeTableEntry` is valid.

`substructureCount` – the number of substructures of this `MuxCodeTableEntry`.

`slotCount` – the number of slots with data from different FlexMux Channels that are described by this substructure.

`repetitionCount` – indicates how often this substructure is to be repeated. A `repetitionCount` zero indicates that this substructure is to be repeated infinitely. `repetitionCount` zero is only permitted in the last substructure of a MuxCodeTableEntry.

`flexMuxChannel[i][k]` – the FlexMux Channel to which the data in this slot belongs.

`numberOfBytes[i][k]` – the number of data bytes in this slot associated to `flexMuxChannel[i][k]`. This number of bytes corresponds to one SL packet.

12.2.5.3 Usage

The `MuxCodeTableEntry` describes how a FlexMux packet is partitioned into slots that carry data from different FlexMux Channels. This is used as a template for parsing FlexMux packets. If a FlexMux packet is longer than the template, parsing shall resume from the beginning of the template. If a FlexMux packet is shorter than the template, the remainder of the template is ignored.

Note that the usage of MuxCode mode may not be efficient if SL packets for a given elementary stream do not have a constant length. Given the overhead for an update of the associated MuxCodeTableEntry, usage of simple mode might be more efficient.

Note further that data for a single FlexMux channel may be conveyed through an arbitrary sequence of FlexMux packets with both simple mode and MuxCode mode.

EXAMPLE —

In this example we assume the presence of three substructures. Each one has a different slot count as well as repetition count. The exact parameters are as follows:

`substructureCount` = 3

`slotCount`[i] = 2, 3, 2 (for the corresponding substructure)

`repetitionCount`[i] = 3, 2, 1 (for the corresponding substructure)

We further assume that each slot configures channel number FMC*n* (`flexMuxChannel`) with a number of bytes Bytes*n* (`numberOfBytes`). This configuration would result in a splitting of the FlexMux packet payload to:

FMC1 (Bytes1), FMC2 (Bytes2) repeated 3 times, then

FMC3 (Bytes3), FMC4 (Bytes4), FMC5 (Bytes5) repeated 2 times, then

FMC6 (Bytes6), FMC7 (Bytes7) repeated once

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000264

The layout of the corresponding FlexMux packet would be as shown in Figure 34.

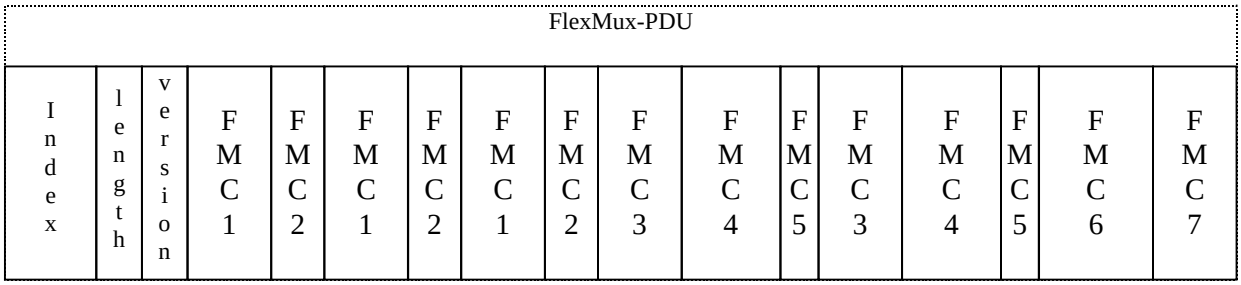


Figure 34 - Example for a FlexMux packet in MuxCode mode

12.2.6 Configuration and usage of FlexMux clock references

12.2.6.1 Syntax

```
aligned(8) class FlexMuxTimingDescriptor {
   bit(16) FCR_ES_ID;
   bit(32) FCRResolution;
   bit(8) FCRLength;
   bit(8) FmxRateLength;
}
```

12.2.6.2 Semantics

The sequence of `fmxClockReference` time stamps in a FlexMux stream constitutes a clock reference stream, albeit with a different syntax as specified in clause 10. Elementary streams may be associated to the time base established by this clock reference by referencing the `FCR_ES_ID` as their `OCR_ES_ID` in the `SLConfigDescriptor`. The transport of the `FlexMuxTimingDescriptor` shall be defined during the design of the transport protocol stack that makes use of the FlexMux tool.

`FCR_ES_ID` – is the ES_ID associated to this clock reference stream.

`FCRResolution` – is the resolution of the object time base in cycles per second.

`FCRLength` – is the length of the `fmxClockReference` field in FlexMux packets with `index` = 238. A length of zero shall indicate that no FlexMux packets with `index` = 238 are present in this FlexMux stream. `FCRlength` shall take values between zero and 64.

`FmxRateLength` - is the length of the `fmxRate` field in FlexMux packets with `index` = 238. `FmxRateLength` shall take values between 1 and 32.

12.2.6.3 Usage

The FlexMux clock reference time stamps may be used to establish and verify a multiplex buffer model. The `fmxClockReference` information determines the arrival time t(i) of individual bytes i of the FlexMux stream in the following way:

$$t(i) = \frac{FCR(i'')}{FCR\,\mathrm{Re}\,solution} + \frac{i - i''}{fmxRate(i)}$$

where:

i is the index of any byte in the FlexMux stream for i" < i < i'

i" is the index of the byte containing the last bit of the most recent `fmxClockReference` field in the FlexMux stream

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000265

FCR(i'') is the time encoded in the `fmxClockReference` in units of `FCRResolution`

fmxRate(i) indicates the rate specified by the fmxRate field for byte i

12.2.7 FlexMux buffer descriptor

12.2.7.1 Syntax

```
aligned(8) class FlexMuxBufferDescriptor {
   bit(8) flexMuxChannel;
   bit(24) FB_BufferSize;
}
```

12.2.7.2 Semantics

The size of multiplex buffers for each FlexMux channel is signaled by `FlexMuxBufferDescriptors`. One descriptor per FlexMux channel is required unless the `DefaultFlexMuxBufferDescriptor` is used. The transport of the `FlexMuxBufferDescriptors` shall be defined during the design of the transport protocol stack that makes use of the FlexMux tool.

`flexMuxChannel` - the number of a FlexMux channel.

`FB_BufferSize` - the size of the FlexMux buffer for this FlexMux channel in bytes.

12.2.8 Default FlexMux buffer descriptor

12.2.8.1 Syntax

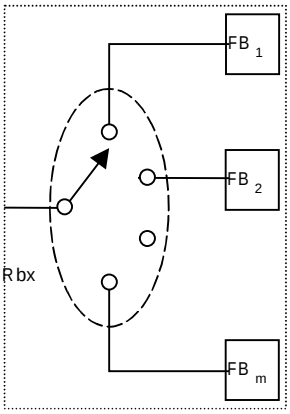
```
aligned(8) class DefaultFlexMuxBufferDescriptor {
   bit(24) FB_DefaultBufferSize;
}
```

12.2.8.2 Semantics

The default size of multiplex buffers for each individual channel in a FlexMux stream is signaled by the `DefaultFlexMuxBufferDescriptor`. FlexMux channels that use a different buffer size may signal this using the `FlexMuxBufferDescriptor`. The transport of the `DefaultFlexMuxBufferDescriptor` shall be defined during the design of the transport protocol stack that makes use of the FlexMux tool.

`FB_DefaultBufferSize` - the default size of FlexMux buffers for this FlexMux stream in bytes.

12.2.9 FlexMux buffer model



FB_n is the FlexMux buffer for the elementary stream in FlexMux channel n
Rbx is the rate at which data enters the FlexMux buffers.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000266

The FlexMux buffer model applies to FlexMux streams that utilize FlexMux Clock reference channel packets to define the delivery timing of the FlexMux stream. The FlexMux stream enters the FlexMux buffer model at the rate and timing as defined by the fmxClockReference and fmxRate fields. There may be some periods of time during which there are no bytes at the input of the FlexMux buffer model, but the bytes of all FlexMux packets that preceed the next FlexMux Clock reference channel packet shall be delivered to the FlexMux buffer model prior to the delivery of any byte of the next FlexMux Clock reference channel packet.

For each FlexMux channel i the FlexMux packet is stored in FlexMux Buffer FB_i. The bytes in buffer FBi are removed at a rate specified by the InstantRate field in the SL header of the contained SL-packetized stream. Upon removal each byte enters the elementary stream buffer DB_i. The FlexMux stream shall be constructed so that the following condition is met :

- Buffer FBi shall not overflow.

13 File Format

13.1 Introduction

The MP4 file format is designed to contain the media information of an ISO/IEC 14496 presentation in a flexible, extensible format that facilitates interchange, management, editing, and presentation of the media. This presentation may be 'local' to the system containing the presentation, or may be via a network or other stream delivery mechanism (a TransMux).

The file format is designed to be independent of any particular TransMux while enabling efficient support for TransMuxes in general.

13.1.1 Usage

The file format is intended to serve as a basis for a number of operations. In these various roles, it may be used in different ways, and different aspects of the overall design exercised.

13.1.1.1 Interchange

When used as an interchange format, the files would normally be self-contained (not referencing media in other files), contain only the media data actually used in the presentation, and not contain any information related to streaming over TransMuxes. This will result in a small, protocol-independent, self-contained file, which contains the core media data and the information needed to operate on it.

The following diagram gives an example of a simple interchange file, containing three streams.

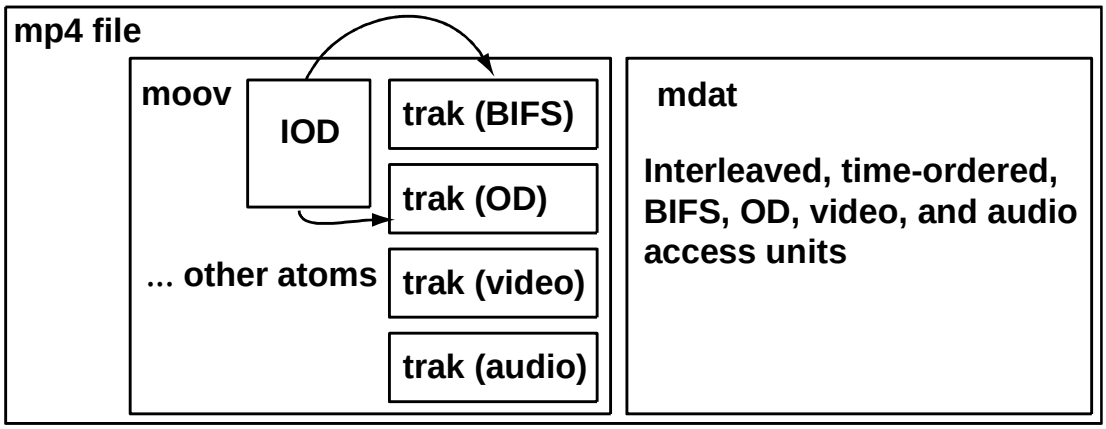


Figure 35 - Simple interchange file

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000267

13.1.1.2 Content Creation

During content creation, a number of areas of the format can be exercised to useful effect, particularly:

- the ability to store each elementary stream separately (not interleaved), possibly in separate files.
- the ability to work in a single presentation which contains MPEG-4 data and other streams (e.g. editing the audio track in the uncompressed format, to align with an already-prepared MPEG-4 video track).

These characteristics mean that presentations may be prepared, edits applied, and content developed and integrated without either iteratively re-writing the presentation on disc - which would be necessary if interleave was required and unused data had to be deleted; and also without iteratively decoding and re-encoding the data - which would be necessary if the data must be stored in an encoded state.

In the following diagram, a set of files being used in the process of content creation is shown.

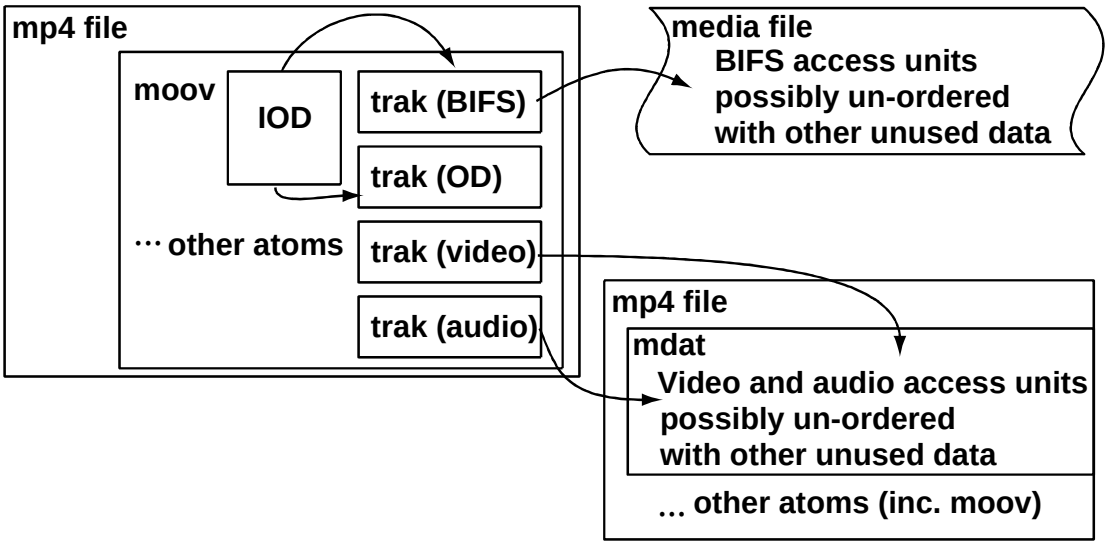


Figure 36 - Content Creation MP4 File

13.1.1.3 Preparation for streaming

When prepared for streaming, the file must contain information to direct the streaming server in the process of sending the information. In addition, it is helpful if these instructions and the media data are interleaved so that excessive seeking can be avoided when serving the presentation. It is also important that the original media data be retained unscathed, so that the files may be verified, or re-edited or otherwise re-used. Finally, it is helpful if a single file can be prepared for more than one protocol, so differing servers may use it over disparate protocols.

13.1.1.4 Local presentation

'Locally' viewing a presentation (i.e. directly from the file, not over a streamed interconnect) is an important application; it is used when a presentation is distributed (e.g. on CD or DVD ROM), during the process of development, and when verifying the content on streaming servers. Such local viewing must be supported, with full random access. If the presentation is on CD or DVD ROM, interleave is important as seeking may be slow.

13.1.1.5 Streamed presentation

When a server operates from the file to make a stream, the resulting stream must be conformant with the specifications for the protocol(s) used, and should contain no trace of the file-format information in the file itself. The server needs to be able to random access the presentation. It can be useful to re-use server content (e.g. to make excerpts) by referencing the same media data from multiple presentations; it can also assist streaming if the media data can be on read-only media (e.g. CD) and not copied, merely augmented, when prepared for streaming.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000268

The following diagram shows a presentation prepared for streaming over a multiplexing protocol, only one hint track is required.

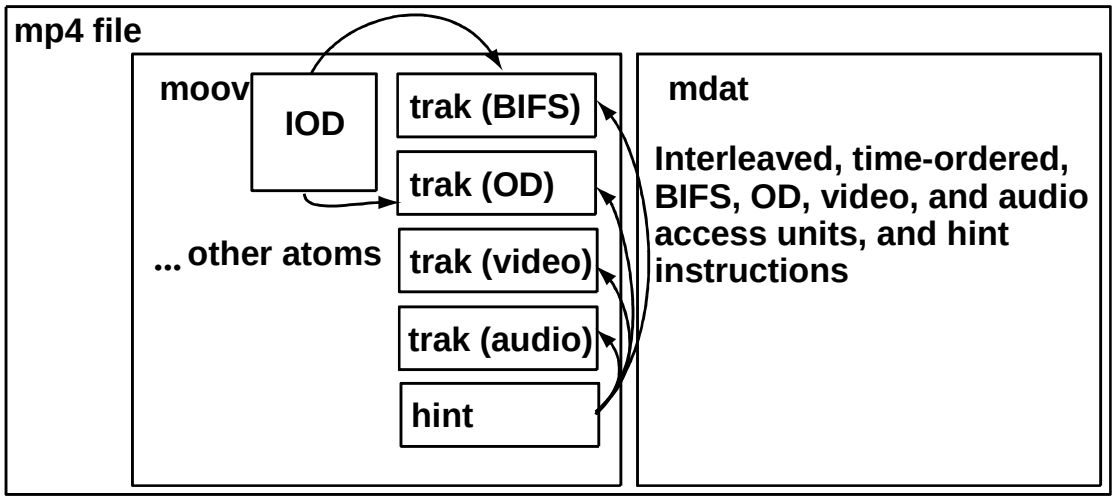


Figure 37 - Hinted Presentation for Streaming

13.1.2 Design principles

The file structure is object-oriented; a file can be decomposed into constituent objects very simply, and the structure of the objects inferred directly from their type.

Media-data is not 'framed' by the file format; the file format declarations which give the size, type and position of media data units is not physically contiguous with the media data. This makes it possible to subset the media-data, and to use it in its natural state, without requiring it to be copied to make space for framing. The meta-data is used to describe the media data by reference, not by inclusion.

Similarly the protocol information for a particular TransMux does not frame the media data; the protocol headers are not physically contiguous with the media data. Instead, the media data is included by reference. This makes it possible to represent media data in its natural state, not favoring any TransMux. It also makes it possible for the same set of media data to serve for local presentation, and for multiple TransMuxes.

The protocol information is built in such a way that the streaming servers need to know only about the protocol and the way it should be sent; the protocol information abstracts knowledge of the media so that the servers are, to a large extent, media-type agnostic. Similarly the media-data, stored as it is in a protocol-unaware fashion, enables the media tools to be protocol-agnostic.

The file format does not require that a single presentation be in a single file. This enables both sub-setting and re-use of content. When combined with the non-framing approach, it also makes it possible to include media data in files not formatted to this specification (e.g. 'raw' files containing only media data and no declarative information, or file formats already in use in the media or computer industries).

The file format is based on a common set of designs and a rich set of possible structures and usage. The same format serves all usages; translation is not required. However, when used in a particular way (e.g. for local presentation), profiles may be used to define the optimal structures and use of options for that usage. However, there is no provision for profiles or levels in the current specification.

13.1.3 Design overview

13.1.3.1 Storage of elementary streams

To maintain the goals of TransMux independence, the media data is stored in its most 'natural' format, and not fragmented. This enables easy local manipulation of the media data. Therefore media-data is stored as access units, a range of contiguous bytes for each access unit.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000269

13.1.3.2 Handling of elementary streams

The elementary streams in an MPEG-4 presentation are stored in the media tracks as access units (a single access unit is the definition of a 'sample' for an MPEG-4 media stream). This is true for all stream types in this draft, including such 'meta-information' streams as Object Descriptor and the Clock Reference. The consequences of this are, on the positive side, that the file format treats all streams equally; on the negative side, this means that there are 'internal' cross-links between the streams. This means that adding and removing streams from a presentation will involve more than adding or deleting the track and its associated media-data. Not only must the stream be placed in, or removed from, the scene, but also the object descriptor stream may need updating.

In a transmitted bit-stream, the access units in the SL Packets are transmitted on byte boundaries. This means that hint tracks will construct SL Packet headers using the information in the media tracks, and the hint tracks will reference the access units from the media track.

The SLConfigDescriptor for the media track shall be stored in the file using a default value (predefined = 2), except when the Elementary Stream Descriptor refers to a stream through a URL, i.e. the referred stream is outside the scope of the MP4 file. In that case the SLConfigDescriptor is not constrained to this predefined value.

13.1.3.3 Handling of FlexMux

An intermediate, optional, fragmentation and packetization step, called FlexMux, has been defined in this document. Some TransMuxes may carry a FlexMux stream rather than packetized elementary streams. Flexmux may be employed for a variety of purposes, including, but not limited to:

- reducing wasted network bandwidth caused by SL Packet header overhead when the payload is small;
- reducing required server resources when providing many streams, by reducing the number of disk reads or network writes.

The process of building FlexMux PDUs is necessarily aware of the characteristics of the TransMux into which the FlexMux must be placed. It is not therefore possible to design a TransMux-independent handling of FlexMux. Instead, in those TransMuxes where FlexMux is used, the hint tracks for that TransMux will encapsulate and include the formation of FlexMux packets. It is expected that the design of the hint tracks (as defined in Section 13.2.2.2) will, in this case, closely reflect the way that FlexMux is used. For example, a compact table resembling the MuxCode (a method used to associate the payload to FlexMux Channels) mode may be needed if the interleave offered by that mode is needed.

Note that in some cases, it may not be possible to create a static FlexMux multiplex via a hint track. Notably, if stream selection is dynamic (for example, based on application feedback) or the choice of muxcode modes or other aspects of Flexmux is dynamic, the FlexMux is therefore created dynamically. This is a necessary cost of run-time multiplexing. It may be difficult for a server to create such a multiplex dynamically at runtime, but with this cost comes added flexibility. A server that wished to provide such functionality could weigh the costs and benefits, and choose to perform the multiplexing without the aid of hint tracks.

Several ISO/IEC 14496 structures are intrinsically linked to FlexMux, and therefore must be addressed in the context of a FlexMux-aware hint track. For example, a stream map table must be supplied to the receiving terminal which maps FlexMux channel IDs to elementary stream IDs. Similarly, if the MuxCode mode of FlexMux is used, a MuxCode mode structure for each MuxCode index used must be defined and supplied to the terminal.

These mappings and definitions may change over time, and there is no normative way in ISO/IEC 14496 to supply these to the terminals; instead, some mechanism, associated with the overall system design or TransMux used, must be employed. The hinter must store the mappings and definitions. Because they are intimately associated with a particular time-segment of a particular hint track, it is recommended that they be placed in the sample description(s) for that hint track. This description would normally be in the form of:

- a table mapping FlexMux channels to elementary stream IDs.
- a set of MuxCode mode structure definitions.

It is recommended further that a format such as that in section 12.2.5, be used for the MuxCode mode definitions.

```
aligned(8) class MuxCodeTableEntry {
   int    i, k;
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000270

```
   bit(8) length;
   bit(4) MuxCode;
   bit(4) version;
   bit(8) substructureCount;
   for (i=0; i<substructureCount; i++) {
      bit(5) slotCount;
      bit(3) repetitionCount;
      for (k=0; k<slotCount; k++){
         bit(8) flexMuxChannel[[i]][[k]];
         bit(8) numberOfBytes[[i]][[k]];
      }
   }
}
```

Special attention must also be taken when pausing or seeking a stream that is being transported as part of a FlexMux stream. Pausing or seeking any component stream of a FlexMux must necessarily pause or seek all the streams. When seeking, care must be taken with random access points. These may not be aligned in time in the streams which form the FlexMux, which means that any seek operation cannot start them all at a random access point. Indeed, the random access points of the FlexMux itself are necessarily rather poorly defined under such circumstances.

It may be necessary for the server to:

- examine the track references to determine the base media tracks (elementary streams) which are formed into the FlexMux;
- find the latest time before the desired seek point such that there is a random access point for all the streams between that time and the seek point, by examining each stream separately;
- transmit the FlexMux stream from that time.

This will ensure that the terminal has received a random access point for all streams at or prior to the desired seek time. However, it may have to discard data for those streams which had data received before the random access points.

13.1.3.4 Handling of TransMuxes

The file format supports streaming of media data over a network as well as local playback. The process of sending protocol data units is time-based, just like the display of time-based data, and is therefore suitably described by a time-based format. A file or ‘movie’ that supports streaming includes information about the data units to stream. This information is included in additional tracks of the file called “hint” tracks.

Hint tracks contain instructions to assist a streaming server in the formation of packets for transmission. These instructions may contain immediate data for the server to send (e.g. header information) or reference segments of the media data. These instructions are encoded in the file in the same way that editing or presentation information is encoded in a file for local playback. Instead of editing or presentation information, information is provided which allows a server to packetize the media data in a manner suitable for streaming using a specific network transport or TransMux.

The same media data is used in a file that contains hints, whether it is for local playback, or streaming over a number of different TransMuxes. Separate ‘hint’ tracks for different TransMuxes may be included within the same file and the media will play over all such TransMuxes without making any additional copies of the media itself. In addition, existing media can be easily made streamable by the addition of appropriate hint tracks for specific TransMuxes. The media data itself need not be recast or reformatted in any way.

This approach to streaming is more space efficient than an approach that requires that the media information be partitioned into the actual data units which will be transmitted for a given transport and media format. Under such an approach, local playback requires either re-assembling the media from the packets, or having two copies of the media—one for local playback and one for streaming. Similarly, streaming such media over multiple TransMuxes using this approach requires multiple copies of the media data for each transport. This is inefficient with space, unless the media data has been heavily transformed for streaming (e.g., by the application of error-correcting coding techniques, or by encryption).

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000271

13.1.3.5 TransMux 'hint' tracks

Support for streaming is based upon the following three design parameters:

- The media data is represented as a set of network-independent standard tracks, which may be played, edited, and so on, as normal;
- There is a common declaration and base structure for server hint tracks; this common format is protocol independent, but contains the declarations of which protocol(s) are described in the server track(s);
- There is a specific design of the server hint tracks for each TransMux that may be transmitted; all these designs use the same basic structure. For example, there may be designs for RTP (for the Internet) and MPEG-2 transport (for broadcast), or for new standard or vendor-specific protocols.

The resulting streams, sent by the servers under the direction of the hint tracks, need contain no trace of file-specific information. This design does not require that the file structures or declaration style, be used either in the data on the wire or in the decoding station. For example, a file using H.261 video and DVI audio, streamed under RTP, results in a packet stream which is fully compliant with the IETF specifications for packing those codings into RTP.

The hint tracks are built and flagged so that when the presentation is viewed directly (not streamed), they may be ignored.

The specific design of the media data (hint samples), and sample descriptions for a particular TransMux is not defined by this part of ISO/IEC 14496. Instead, the designer of the system using that TransMux, or the body that owns and defines the TransMux, would define these tracks. Clearly there is an advantage in having standard hint track formats for standard TransMuxes, and their development and publication is encouraged.

13.2 File organization

13.2.1 Presentation structure

13.2.1.1 File Structure

A presentation may be contained in several files. One file contains the meta-data for the whole presentation, and is formatted to this specification. This file may also contain all the media data, whereupon the presentation is self-contained. The other files, if used, do not have to be formatted to this specification; they can contain used or unused media data, or other information. This specification concerns the structure of the presentation file only. The format of the media-data files is constrained by this specification only in that the media-data in the media files must be able to be described by the meta-data defined here.

If a MP4 file contains hint tracks, the media tracks which reference the media data from which the hints were built must remain in the file, even if the data within them is not directly referenced by the hint tracks.

13.2.1.2 Object Structure

The file is structured as a sequence of objects; some of these objects may contain other objects. The sequence of objects in the file must contain exactly one presentation meta-data wrapper (the movie atom). It is usually at the beginning or end of the file, to permit its easy location. The other objects that are found at this level may be free space, or media data atoms.

The fields in the objects are stored in network byte order (big-endian format).

13.2.1.3 Meta Data and Media Data

The meta-data is contained within the meta-data wrapper (the movie atom); the media data is contained either in the same file, within media-data atom(s), or in other files. The media data is composed of access units; the media data objects, or media data files, may contain other unreferenced information.

13.2.1.4 Track Identifiers

The track identifiers used in an MP4 file are unique within that file; no two tracks may use the same identifier.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000272

Each elementary stream in the file is stored as a media track. The lower two bytes are the elementary stream identifier (ES_ID). The upper two bytes are zero. Hint tracks may use track identifier values in the same range, if this number space is adequate (which it generally is). However, hint track identifiers may also use larger values of track identifier, as their identifiers are not mapped to elementary stream identifiers. Thus very large presentations can use the entire 16-bit number space for elementary stream identifiers.

The next track identifier value in the movie header generally contains a value one greater than the largest track identifier value found in the file. This enables easy generation of a track identifier under most circumstances. However, if this value is equal to or larger than 65535, and a new media track is to be added, then a search must be made in the file for a free track identifier. If the value is all 1s (32-bit maxint) then this search is needed for all additions.

If it is desired to add a track with a known track identifier (elementary stream identifier) then the file must be searched to ensure that there is no conflict. Note that hint tracks can be re-numbered fairly easily while more care should be taken with media tracks, as there may be references to their ES_ID (track ID) in other tracks.

Note that if it is desired to have hint tracks have track IDs outside the allowed range for elementary stream tracks, then next track ID will document the next available hint track ID. Since this is larger than 65535, a search will then always be needed to find a valid elementary stream track ID.

If two presentations are merged, then there may be conflict between their track IDs. In that case, one or more tracks will have to be re-numbered. There are two actions to be taken here:

- Changing the ID of the track itself, which is easy (track ID in the track header).
- Changing pointers to it.

The pointers may only occur in the file format structure itself. The file format uses track IDs only through track references, which are easily found and modified. Track IDs become ES_IDs in the MPEG-4 data, and ES_IDs occur within the OD Stream. Since all pointers to ES_IDs in the OD stream are replaced by means of track references there is no need to inspect the OD stream for cross-references within MPEG-4 streams.

Note that in ES_DescriptorRemove and IPI_DescrPointer it is a track reference *index* (using references of type mpod and ipir respectively – see subclause 13.2.3.7.2) that is stored in the file, and the tag values are specific to the file format (ES_DescrRemoveRefTag and IPI_DescrPointerRefTag). These reference indexes should be replaced with the ES_ID when hinting or serving, and the tag values adjusted.

13.2.1.5 Synchronization of streams

In the absence of explicit declarations to the contrary, tracks (streams) coming from the same file should be presented synchronized. This means that hinters and/or servers should either pick one of the streams to serve as the OCR source for the others or add an OCR stream to associate all the streams with it. Track references of type 'sync' can be used in the file to defeat the default behavior. In MPEG-4 the OCRStreamFlag and OCR_ES_ID fields in the ESDescriptor govern the synchronization relationships. The mapping of MP4 structures into those fields shall obey the following rules:

- The MPEG-4 ESDescriptor, as stored in the file, usually contains OCRStreamFlag set to FALSE, and no OCR_ES_ID. If an OCR_ES_ID is set, it should be used as is without interpretation or inspection; this case is normally used to synchronize to streams outside the file.
- If a track (stream) contains a track reference of type 'sync' whose value is 0, then the hinter or server shall set the OCRStreamFlag field in the MPEG-4 ESDescriptor to FALSE and shall not insert any OCR_ES_ID field. This means that this stream is not synchronized to another, but other streams can be synchronized to it.
- If a track (stream) contains a track reference of type 'sync' whose value is not 0, then the hinter or server shall set the OCRStreamFlag field in the MPEG-4 ESDescriptor to TRUE and shall insert an OCR_ES_ID field with the same value contained in the 'sync' track reference. This means that this stream is synchronized to the stream indicated in the OCR_ES_ID. Note that other streams may also be synchronized to the same stream, either explicitly or implicitly.
- If a track (stream) does not contain a track reference of type 'sync', then the default behavior applies. The hinter or server shall set the OCRStreamFlag field in the MPEG-4 ESDescriptor to TRUE and shall insert an

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000273

OCR_ES_ID field with a value selected based on the rules below. This means that this stream is synchronized to the stream indicated in the OCR_ES_ID. The rules for selecting the OCR_ES_ID are as follows:

- if no track (stream) in the file contains a track reference of type 'sync', then the hinter picks one trackId and uses that value for the OCR_ES_ID field of all ESDescriptors. There is one possible exception where the ESDescriptor of the stream which corresponds to that trackId, for which the OCRStreamFlag may be set to FALSE.

- if one or more tracks (streams) in the file contain a track reference of type 'sync', and all such track references indicate consistently a single trackId, then the hinter uses that trackId. Note that in a track reference of type 'sync' the value 0 is equivalent to the trackId of the track itself.

- if two or more tracks (streams) in the file contain a track reference of type 'sync', and such track references do not indicate a single trackId, then the hinter cannot make a deterministic selection and the behavior is undefined. Note again that in a track reference of type 'sync' the value 0 is equivalent to the trackId of the track itself.

13.2.2 Media Data Structure

13.2.2.1 Elementary Stream Tracks

In the file format, the media data is stored as access units; for each track the entire ES-descriptor is stored as the sample description or descriptions. The SLConfigDescriptor is stored according to a default value (predefined = 2) except when the ES-descriptor contains a URL to point to an elementary stream outside the scope of the MP4 file.

Note that the SL Packet header and payload are byte-aligned, so the placement of the header during hinting is possible without bit shifting, as each SL Packet and corresponding contained access unit will both start on byte boundaries.

Note also that an access unit must be stored as a contiguous set of bytes. This greatly facilitates the fragmentation process used in hint tracks. The file format can describe and use media data stored in other files, however this restriction still applies. Therefore if a file is to be used which contains ‘pre-fragmented’ media data (e.g. a FlexMux stream on disc), the media data will need to be copied to re-form the access units, in order to import the data into this file format.

The ESDescriptor for a stream within the scope of the MP4 file as described in this document is stored in the sample description and the fields and included structures are restricted as follows:

- `ES_ID` - set to 0 as stored; when built into a TransMux, the lower 16 bits of the trackID is used.
- `streamDependenceFlag` – set to 0 as stored; instead, track references of type ‘dpnd’ are used.
- `URLflag` – set to false, as the stream is in the file, not remote.
- `SLConfigDescriptor` - is predefined type 2.
- `OCRStreamFlag` – set to false in the file.

Note that the QoSDescriptor also may need re-writing for transmission as it contains information about PDU sizes etc.

13.2.2.1.1 Object Descriptors

The initial object descriptor and object descriptor streams are handled specially within the file format. Object descriptors contain ES descriptors, which in turn contain TransMux specific information. In addition, to facilitate editing, the information about a track is stored as an ESDescriptor in the sample description within that track. It must be taken from there, re-written as appropriate, and transmitted as part of the OD stream when the presentation is streamed.

As a consequence, ES descriptors are not stored within the OD track or initial object descriptor. Instead, the initial object descriptor has a descriptor used only in the file, containing solely the track ID of the elementary stream. When used, an appropriately re-written ESDescriptor from the referenced track replaces this descriptor. Likewise, OD tracks are linked to ES tracks by track references. Where an ES descriptor would be used within the OD track,

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000274

another descriptor is used, which again occurs only in the file. It contains the index into the set of mpod track references that this OD track owns. A suitably re-written ESDescriptor replaces it by the hinting of this track.

The ES_ID_Inc is used in the initial object descriptor atom:

```
class ES_ID_Inc extends BaseDescriptor : bit(8) tag=ES_IDIncTag {
   unsigned int(32)   Track_ID;    // ID of the track to use
}

ES_ID_IncTag = 0x0E is reserved for file format usage.
```

The ES_ID_Ref is used in the OD stream:

```
class ES_ID_Ref extends BaseDescriptor : bit(8) tag=ES_IDRefTag {
   bit(16)    ref_index; // track ref. index of the track to use
}

ES_ID_RefTag = 0x0F is reserved for file format usage.
MP4_IOD_Tag = 0x10 is reserved for file format usage.
MP4_OD_Tag = 0x11 is reserved for file format usage.
IPL_DescrPointerRefTag = 0x12 is reserved for file format usage.
ES_DescrRemoveRefTag = 0x07 is reserved for file format usage (command tag).
```

NOTE - The above tag values are defined in subclause 8.2.2.2 Table 1 and subclause 8.2.3.2 Table 2, and the actual values should be referenced from those tables.

A hinter may need to send more OD events than actually occur in the OD track: for example, if the ES_description changes at a time when there is no event in the OD track. In general, any OD events explicitly authored into the OD track should be sent along with those necessary to indicate other changes. The ES descriptor sent in the OD track would be taken from the description of the temporally next sample in the ES track (in decoding time).

13.2.2.2 Hint Tracks

Hint tracks are used to describe to a server how to serve the elementary stream data in the file over TransMuxes. Each TransMux has its own hint track format. The format of the hints is described by the sample description for the hint track. Most TransMuxes will need only one sample description format for each track.

Servers find their hint tracks by first finding all hint tracks, and then looking within that set for hint tracks using their protocol (sample description format). If there are choices at this point, then the server chooses on the basis of preferred protocol or by comparing features in the hint track header or other protocol-specific information in the sample descriptions.

Hint tracks construct TransMuxes by pulling data out of other tracks by reference. These other tracks may be hint tracks or elementary stream tracks. The exact form of these pointers is defined by the sample format for the protocol, but in general they consist of four pieces of information: a track reference index, a sample number, an offset, and a length. Some of these may be implicit for a particular protocol. Note that these 'pointers' always point to the actual source of the data. If a hint track is built 'on top' of another hint track, then the second hint track will have direct references to the media track(s) used by the first where data from those media tracks is placed in the TransMux.

All hint tracks use a common set of declarations and structures.

- Hint tracks are linked to the elementary stream tracks they carry, by track references of type 'hint'
- They use a handler-type of 'hint' in the handler reference atom
- They use a hint media header atom
- They use a hint sample entry in the sample description, with a name and format unique to the protocol they represent.

Hint tracks may be created by an authoring tool, or may be added to an existing presentation by a hinting tool. Such a tool serves as a 'bridge' between MPEG-4 and the protocol, since it intimately understands both. This permits authoring tools to understand MPEG-4, but not protocols, and for servers to understand protocols (and their hint tracks) but not the details of MPEG-4 data.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000275

13.2.3 Meta-data Structure (Objects)

The following represents the subset of the QuickTime file specification that is required to define an MP4 file. An object in QuickTime terminology is an Atom. Atoms not explicitly defined in this standard may be ignored.

Atoms start with a header that gives both size and type. The header permits compact or extended size (32 or 64 bits unsigned integer) giving the size of the object in bytes and compact or extended types (32 bits unsigned integer or full UUIDs). The standard MPEG-4 atoms all use compact types (32-bit) normally interpreted and presented as four printable characters, for ease of identification. Most atoms will use the compact (32-bit) size. Typically only the media data atom(s) may need the 64-bit size.

Note that the size is the entire size of the atom, including the size and type header, fields, and all contained atoms. This facilitates simplified parsing of the file. A zero size field, allowed only at the top-level atoms, indicates that the last atom in the file which extends to the end of the file. This is normally only used for media data (mdat) atoms. Note also that all indexes start with the value one rather than zero.

```
aligned(8) class Atom (unsigned int(32) atomtype,
        optional unsigned int(8)[16] extended-type) {
   unsigned int(32) size;
   unsigned int(32) type = atomtype;
   if (size==1) {
      unsigned int(64) largesize;
   }
   if (atomtype==‘uuid’) {
      unsigned int(8)[16] usertype = extended-type;
   }
}
```

The semantics of these two fields are:

`size` - is an integer that specifies the number of bytes in this atom including all its fields and contained atoms; if size is set to 1 then the actual size is given by the large size field.

`type` - identifies the atom type. Standard atoms use a compact type that is normally four printable characters to permit ease of identification, and this is shown in the atoms below. User extensions use an extended type. In this case the type field is set to ‘uuid’.

Type fields not defined here are reserved. Private extensions shall be achieved through the ‘uuid’ type. The following types are reserved and will either not be used or will be used only in their existing sense in future versions of this specification to avoid conflict with existing content using earlier pre-standard versions of this format:

```
clip, crgn, matt, kmat, pnot, ctab, load, imap; track reference types tmcd, chap,
scpt, ssrc.
```

Many objects also contain a version number and flags field:

```
aligned(8) class FullAtom(unsigned int(32) atomtype, unsigned int(8) v, bit(24) f)
   extends Atom(atomtype) {
   unsigned int(8)   version = v;
   bit(24)   flags = f;
 }
```

The semantics of these two fields are:

`version` - is an integer that specifies the version of this format of the atom.

`flags` - is a map of flags.

In a number of atoms in this specification, there are two variant forms: version 0 using 32-bit fields, and version 1 using 64-bit sizes for those same fields. In general, if a version 0 atom (32-bit field sizes) can be used, it should be; version 1 atoms should be used only when the 64-bit field sizes they permit, are required.

For convenience during content creation there are creation and modification times stored in the file. These can be 32-bit or 64-bit numbers, counting seconds since midnight, Jan. 1, 1904, which is a convenient date for leap-year calculations. 32 bits are sufficient until approximately year 2040.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000276

Fields shown as reserved in the atom descriptions should be initialized to the given value on atom creation, copied un-inspected when atoms are copied, and ignored on reading.

An overall view of the normal encapsulation structure is provided in the following table.

The table shows atoms that may occur at the top-level in the left-most column; indentation is used to show possible containment. Thus, for example, a track header (tkhd) is found in a track (trak), which is found in a movie (moov). Not all atoms need be used in all files; the mandatory atoms are marked with an asterisk (*). See the description of the individual atoms for a discussion of what must be assumed if the optional atoms are not present.

Note that user data objects may be found in moov or trak atoms, and objects using an extended type may be placed in a wide variety of containers, not just the top level.

Table 55 - Overview of Atom Encapsulation Structure

Moov						*	13.2.3.1	*container for all the meta-data*
	mvhd					*	13.2.3.3	*movie header, overall declarations*
	iods					*	13.2.3.4	*object descriptor*
	trak					*	13.2.3.4.2	*container for an individual track or stream*
		tkhd				*	13.2.3.6	*track header, overall information about the track*
		tref					13.2.3.7	*track reference container*
		edts					13.2.3.25	*edit list container*
			elst				13.2.3.26	*an edit list*
		mdia				*	13.2.3.8	*container for the media information in a track*
			mdhd			*	13.2.3.9	*media header, overall information about the media*
			hdlr				13.2.3.10	*handler, at this level, the media (handler) type*
			minf			*	13.2.3.11	*media information container*
				vmhd			13.2.3.12.1	*video media header, overall information (video track only)*
				smhd			13.2.3.12.2	*sound media header, overall information (sound track only)*
				hmhd			13.2.3.12.3	*hint media header, overall information (hint track only)*
				<mpeg>			13.2.3.12.4	*mpeg stream headers*
				dinf		*	13.2.3.13	*data information atom, container*
					dref	*	13.2.3.14	*data reference atom, declares source(s) of media in track*
				stbl		*	13.2.3.15	*sample table atom, container for the time/space map*
					stts	*	13.2.3.16.1	*(decoding) time-to-sample*
					ctts		13.2.3.16.2	*composition time-to-sample table*
					stss		13.2.3.21	*sync (key, I-frame) sample map*
					stsd	*	13.2.3.17	*sample descriptions (codec types, initialization etc.)*
					stsz	*	13.2.3.18	*sample sizes (framing)*
					stsc		13.2.3.19	*sample-to-chunk, partial data-offset information*
					stco		13.2.3.20	*chunk offset, partial data-offset information*
					stsh		13.2.3.22	*shadow sync*
					stdp		13.2.3.23	*degradation priority*
mdat							13.2.3.2	*Media data container*
free							13.2.3.24	*free space*
skip							13.2.3.24	*free space*
udta							13.2.3.27	*user-data, copyright etc.*

13.2.3.1 Movie Atom

Atom Type: 'moov'
Container: File
Mandatory: Yes
Quantity: Exactly one
The meta-data for a presentation is stored in the single Movie Atom that occurs at the top-level of a file. Normally this atom is the first or last in the sequence of atoms in a file, though this is not required.

13.2.3.1.1 Syntax

```
aligned(8) class MovieAtom extends Atom('moov'){
 }
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000277

13.2.3.2 Media Data Atom

Atom Type: 'mdat'
Container: File
Mandatory: No
Quantity: Any number

This atom contains the media data. In elementary stream tracks, this atom will contain MPEG-4 data as access units. A presentation may contain zero or more media data atoms. The actual media data follows the type field; its structure is described by the meta-data (see particularly the sample table).

In large presentations, it may be desirable to have more data in this atom than a 32-bit size would permit. In this case, the large variant of the size field, above, is used.

NOTE - there may be any number of these atoms in the file (including zero, if all the media data is in other files). The meta-data refers to media data by its absolute offset within the file (see the chunk offset atom); so mdat headers and free space may easily be skipped, and files without any atom structure may also be referenced and used.

13.2.3.2.1.1 Syntax

```
aligned(8) class MediaDataAtom extends Atom('mdat') {
   bit(8) data[];
 }
```

13.2.3.2.1.2 Semantics

`data` - is the contained media data

13.2.3.3 Movie Header Atom

Atom Type: 'mvhd'
Container: Movie Atom ('moov')
Mandatory: Yes
Quantity: Exactly one
This atom defines overall information that is media-independent, and relevant to the entire presentation considered as a whole.

13.2.3.3.1 Syntax

```
aligned(8) class MovieHeaderAtom (unsigned int(32) version) extends FullAtom('mvhd', version,
0) {
   if (version==1) {
      unsigned int(64)  creation-time;
      unsigned int(64)  modification-time;
      unsigned int(32)  timescale;
      unsigned int(64)  duration;
   } else { // version==0
      unsigned int(32)  creation-time;
      unsigned int(32)  modification-time;
      unsigned int(32)  timescale;
      unsigned int(32)  duration;
   }
   const bit(32)reserved = 0x00010000;
   const bit(16)reserved = 0x0100;
   const bit(16)reserved = 0;
   const unsigned int(32)[2] reserved = 0;
   const bit(32)[9]  reserved =
      { 0x00010000, 0, 0, 0, 0x00010000, 0, 0, 0, 0x40000000 };
   const bit(32)[6]  reserved = 0;
   unsigned int(32)  next-track-ID;
 }
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000278

13.2.3.3.2 Semantics

`version` - is an integer that specifies the version (0 or 1 in this draft).

`creation-time` - is an integer which declares the creation time of the presentation (in seconds since midnight, Jan. 1, 1904.

`modification-time` - is an integer which declares the most recent time the presentation was modified (in seconds since midnight, Jan. 1, 1904.

`timescale` - is an integer which specifies the time-scale for the entire presentation; this is the number of time units which pass in one second. A time coordinate system that measures time in sixtieths of a second, for example, has a time scale of 60.

`duration` - is an integer which declares length of the presentation (in the scale of the timescale). Note that this property is derived from the presentation's tracks. The value of this field corresponds to the duration of the longest track in the presentation.

`next-track-ID` - is an integer which indicates a value to use for the track ID of the next track to be added to this presentation. Note that 0 is not a valid track ID value. This must be larger than the largest track-ID in use. If this value is equal to or larger than 65535, and a new media track is to be added, then a search must be made in the file for a free track identifier that will fit into 16 bits. If the value is all 1s (32-bit maxint) then this search is needed for all additions.

13.2.3.4 Object Descriptor Atom

Atom Type: 'iods'
Container: Movie Atom ('moov')
Mandatory: Yes
Quantity: Exactly one

This object contains an Object Descriptor or an Initial Object Descriptor.

There are a number of possible file types based on usage, depending on the descriptor:

- Presentation, contains IOD which contains a BIFS stream (MP4 file)
- Sub-part of a presentation, contains an IOD without a BIFS stream (MP4 file)
- Sub-part of a presentation, contains an OD (MP4 file)
- Free-form file, Referenced by MP4 data references (free-format)

NOTE - The first three are MP4 files, the last file is not necessarily an MP4 file, as it is free-format.

13.2.3.4.1 Syntax

```
aligned(8) class ObjectDescriptorAtom
   extends FullAtom('iods', version = 0, 0) {
   ObjectDescriptor OD;
}
```

The syntax for ObjectDescriptor and InitialObjectDescriptor is described in 8.6.2 through 8.6.4.

13.2.3.4.2 Semantics

The semantics for ObjectDescriptor and InitialObjectDescriptor are described in 8.6.2 through 8.6.4.. The contents of this atom are formed by taking an object descriptor or initial object descriptor and:

- changing the tag to MP4_OD_Tag or MP4_IOD_Tag as appropriate for this object
- replacing the ES descriptors with ES_ID_Inc referencing the appropriate track.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000279

13.2.3.5 Track Atom

Atom Type: 'trak'
Container: Movie Atom ('moov')
Mandatory: Yes
Quantity: 1 or more

This is a container atom for a single track of a presentation. A presentation may consist of one or more tracks. Each track is independent of the other tracks in the presentation and carries its own temporal and spatial information. Each track will contain its associated media atom.

Tracks are used for two purposes: (a) to contain elementary media data (media tracks) and (b) to contain packetization information for streaming protocols (hint tracks).

There must be at least one media track within a MP4 file; and all the media tracks that contributed to the hint tracks present must remain in the file, even if the media data within them is not referenced by the hint tracks. After deleting all hint tracks, the entire un-hinted presentation must remain.

13.2.3.5.1 Syntax

```
aligned(8) class TrackAtom extends Atom('trak') {
 }
```

13.2.3.6 Track Header Atom

Atom Type: 'tkhd'
Container: Track Atom ('trak')
Mandatory: Yes
Quantity: Exactly one

The track header atom specifies the characteristics of a single track. Exactly one track header atom is contained in a track.

In the absence of an edit list, the presentation of a track starts immediately. An empty edit is used to offset the start time of a track.

13.2.3.6.1 Syntax

```
aligned(8) class TrackHeaderAtom
   extends FullAtom('tkhd', version, flags){
   if (version==1) {
      unsigned int(64)  creation-time;
      unsigned int(64)  modification-time;
      unsigned int(32)  track-ID;
      const unsigned int(32)  reserved = 0;
      unsigned int(64)  duration;
   } else { // version==0
      unsigned int(32)  creation-time;
      unsigned int(32)  modification-time;
      unsigned int(32)  track-ID;
      const unsigned int(32)  reserved = 0;
      unsigned int(32)  duration;
   }
   const unsigned int(32)[3] reserved = 0;
   const bit(16)reserved = { if track_is_audio 0x0100 else 0};
   const unsigned int(16)  reserved = 0;
   const bit(32)[9]   reserved =
      { 0x00010000, 0, 0, 0, 0x00010000, 0, 0, 0, 0x40000000 };
   const bit(32)reserved = {
      if track_is_visual 0x01400000 else 0 };
   const bit(32)reserved = {
      if track_is_visual 0x00F00000 else 0};
}
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000280

13.2.3.6.2 Semantics

`Version` - is an integer that specifies the version (0 or 1 in this draft).

`flags` - is a 24-bit integer with flags; the following values are defined.

`Track enabled` - Indicates that the track is enabled. Flag value is 0x000001. A disabled track (the low bit is zero) is treated as if it were not present.

`creation-time` - is an integer which declares the creation time of this track (in seconds since midnight, Jan. 1, 1904).

`modification-time` - is an integer which declares the most recent time the track was modified (in seconds since midnight, Jan. 1, 1904).

`track-ID` - is an integer which uniquely identifies this track over the entire lifetime of this presentation. Track Ids are never re-used and cannot be zero.

`duration` - is an integer that indicates the duration of this track (in the movie's time coordinate system). Note that this property is derived from the track's edits. The value of this field is equal to the sum of the durations of all of the track's edits. If there is no edit list, then the duration is the sum of the sample durations, converted into the movie time-scale.

13.2.3.7 Track reference atom

Atom Type: `tref'
Container: Track Atom ('trak')
Mandatory: No
Quantity: 0 or 1

The track reference atom provides a reference from the containing stream to another stream in the presentation. These references are typed. In particular, a 'hint' reference links from the containing hint track to the media data that it hints. Exactly one track reference atom can be contained within the track atom.

If this atom is not present, the track is not referencing any other track in any way. Note that the reference array is sized to fill the reference type atom.

Track references with a reference index of 0 are permitted. This indicates no reference, which can be useful to defeat the implied synchronization reference between tracks in the same file, when this implied behavior is not desired.

13.2.3.7.1 Syntax

```
aligned(8) class TrackReferenceAtom extends Atom('tref') {
 }
aligned(8) class TrackReferenceTypeAtom (unsigned int(32) reference-type) extends
Atom(reference-type) {
   unsigned int(32) track-IDs[];
}
```

13.2.3.7.2 Semantics

The track reference atom contains track reference type atoms. These are structured as track reference type atoms.

The `reference-type` must be set to one of the following values:

- `hint` - the referenced track(s) contain the original media for this hint track
- `dpnd` - the referencing track has an MPEG-4 dependency on the referenced track
- `ipir` - this track contains IPI declarations for the referenced track

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000281

- `mpod` - the referencing track is an OD track which uses the referenced track as an included elementary stream track
- `sync` - this track uses the referenced track as its synchronization source.

13.2.3.8 Media atom

Atom Type: 'mdia'
Container: Track Atom ('trak')
Mandatory: Yes
Quantity: Exactly one

The media declaration container contains all the objects that declare information about the media data within a stream.

13.2.3.8.1 Syntax

```
aligned(8) class MediaAtom extends Atom('mdia') {
 }
```

13.2.3.9 Media header atom

Atom Type: 'mdhd'
Container: Media Atom ('mdia')
Mandatory: Yes
Quantity: Exactly one

The media header declares the overall media-independent information relevant to the characteristics of the media in a stream.

13.2.3.9.1 Syntax

```
aligned(8) class MediaHeaderAtom extends FullAtom('mdhd', version, 0) {
   if (version==1) {
      unsigned int(64)  creation-time;
      unsigned int(64)  modification-time;
      unsigned int(32)  timescale;
      unsigned int(64)  duration;
   } else { // version==0
      unsigned int(32)  creation-time;
      unsigned int(32)  modification-time;
      unsigned int(32)  timescale;
      unsigned int(32)  duration;
   }
   bit(1)  pad = 0;
   unsigned int(5)[3] language;  // packed ISO-639-2/T language code
   const unsigned int(16)  reserved = 0;
 }
```

13.2.3.9.2 Semantics

`Version` - is an integer that specifies the version.

`creation-time` - is an integer which declares the creation time of the presentation (in seconds since midnight, Jan. 1, 1904).

`modification-time` - is an integer which declares the most recent time the presentation was modified (in seconds since midnight, Jan. 1, 1904).

`timescale` - is an integer which specifies the time-scale for this media; this is the number of time units which pass in one second. A time coordinate system that measures time in sixtieths of a second, for example, has a time scale of 60.

`duration` - is an integer which declares length of this media (in the scale of the timescale).

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000282

`language` - declares the language code for this media. See ISO 639-2/T for the set of three character codes. Each character is packed as the difference between its ASCII value and 0x60. The code is confined to being three lower-case letters, so these values are strictly positive. If the language code is unknown, it should be marked as `und' for undetermined. The value 0 should be interpreted as undetermined.

13.2.3.10 Handler reference atom

Atom Type: 'hdlr'
Container: Media Atom ('mdia') Atom
Mandatory: Yes
Quantity: 1 only

The handler atom within a Media Atom declares the process by which the media-data in the stream may be presented, and thus, the nature of the media in a stream. For example, a video handler would handle a video track.

13.2.3.10.1 Syntax

```
aligned(8) class HandlerAtom extends FullAtom('hdlr', version = 0, 0) {
   const unsigned int(32)  reserved = 0;
   unsigned int(32)   handler-type;
   const unsigned int(8)[12] reserved = 0;
   string  name;
 }
```

13.2.3.10.2 Semantics

`version` - is an integer that specifies the version.

`handler-type` - is an integer containing one of the following values:

`'odsm'`	ObjectDescriptorStream
`'crsm'`	ClockReferenceStream
`'sdsm'`	SceneDescriptionStream
`'vide'`	VisualStream
`'soun'`	AudioStream
`'m7sm'`	MPEG7Stream
`'ocsm'`	ObjectContentInfoStream
`'ipsm'`	IPMP Stream
`'mjsm'`	MPEG-J Stream
`'hint'`	Hint track

`name` - is a null-terminated string in UTF-8 characters which gives a human-readable name for the stream type (for debugging and inspection purposes).

13.2.3.11 Media information atom

Atom Type: 'minf'
Container: Media Atom ('mdia')
Mandatory: Yes
Quantity: Exactly one

The media information atom contains all the objects that declare characteristic information of the media in the stream.

13.2.3.11.1 Syntax

```
aligned(8) class MediaInformationAtom extends Atom('minf') {
 }
```

13.2.3.12 Media information header atoms

Atom Types: 'vmhd','smhd','hmhd'
Container: Media Information Atom ('minf')

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000283

Mandatory: exactly one media header must be present
Quantity: 1 only

There is a media information header for each track type (corresponding to the media handler type). This header is used for all tracks containing visual streams.

13.2.3.12.1 Video Media Header Atom

The video media header contains general presentation information, independent of the coding, for visual media.

13.2.3.12.1.1 Syntax

```
aligned(8) class VideoMediaHeaderAtom
   extends FullAtom(‘vmhd’, version = 0, 1) {
   const unsigned int(64)  reserved = 0;
 }
```

13.2.3.12.1.2 Semantics

`version` - is an integer that specifies the version.

13.2.3.12.2 Sound Media Header Atom

The sound media header contains general presentation information, independent of the coding, for audio media. This header is used for all tracks containing audio streams.

13.2.3.12.2.1 Syntax

```
aligned(8) class SoundMediaHeaderAtom
   extends FullAtom(‘smhd’, version = 0, 0) {
   const unsigned int(32)  reserved = 0;
 }
```

13.2.3.12.2.2 Semantics

`version` - is an integer that specifies the version.

13.2.3.12.3 Hint Media Header Atom

The hint media header contains general information, independent of the protocol, for hint tracks.

13.2.3.12.3.1 Syntax

```
aligned(8) class HintMediaHeaderAtom
   extends FullAtom(‘hmhd’, version = 0, 0) {
   unsigned int(16)  maxPDUsize;
   unsigned int(16)  avgPDUsize;
   unsigned int(32)  maxbitrate;
   unsigned int(32)  avgbitrate;
   unsigned int(32)  slidingavgbitrate;
 }
```

13.2.3.12.3.2 Semantics

`version` - is an integer that specifies the version.

`maxPDUsize` - gives the size in bytes of the largest PDU in this (hint) stream.

`avgPDUsize` - gives the average size of a PDU over the entire presentation.

`maxbitrate` - gives the maximum rate in bits/second over any window of one second.

`avgbitrate` - gives the average rate in bits/second over the entire presentation.

`slidingavgbitrate` - gives the maximum rate in bits/second over any one minute window (corresponding to the avgBitrate field in the DecoderConfigDescriptor).

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000284

13.2.3.12.4 MPEG-4 Media Header Atoms

ISO/IEC 14496 streams other than visual and audio currently use an empty MPEG-4 media header atom, as defined here. There is a set of reserved types for media headers specific to these ISO/IEC 14496 stream types.

13.2.3.12.4.1 Syntax

```
aligned(8) class Mpeg4MediaHeaderAtom
   extends FullAtom('nmhd', version = 0, flags) {
 }
```

13.2.3.12.4.2 Semantics

`version` - is an integer that specifies the version.

`flags` - is a 24-bit integer with flags (currently all zero).

The following types are reserved but currently unused:

```
ObjectDescriptorStream  'odhd'
ClockReferenceStream    'crhd'
SceneDescriptionStream  'sdhd'
MPEG7Stream             'm7hd'
ObjectContentInfoStream 'ochd'
IPMP Stream             'iphd'
MPEG-J Stream           'mjhd'
```

13.2.3.13 Data information atom

Atom Type: 'dinf'
Container: Media Information Atom ('minf')
Mandatory: Yes
Quantity: Exactly one

The data information atom contains objects that declare the location of the media information in a stream.

13.2.3.13.1 Syntax

```
aligned(8) class DataInformationAtom extends Atom('dinf') {
 }
```

13.2.3.14 Data reference atom

Atom Types: 'url ', 'urn ', 'dref'
Container: Data Information Atom ('dinf')
Mandatory: Yes
Quantity: Exactly one

The data reference object contains a table of data references (normally URLs) which declare the location(s) of the media data used within the presentation. The data reference index in the sample description ties entries in this table to samples. A track may be split over several sources in this way.

If the flag is set indicating that the data is in the same file as this atom, then no string should be supplied in the entry field, not even an empty one.

13.2.3.14.1 Syntax

```
aligned(8) class DataEntryUrlAtom(unsigned int(32) version, bit(24) flags)
   extends FullAtom('url ', version = 0, flags) {
   string  location;
}
aligned(8) class DataEntryUrnAtom(unsigned int(32) version, bit(24) flags)
   extends FullAtom('urn ', version = 0, flags) {
   string  name;
   string  location;
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000285

```
}
aligned(8) class DataReferenceAtom
   extends FullAtom('dref', version = 0, 0) {
   unsigned int(32)  entry-count;
   int i;
   for (i=0; i < entry-count; i++) {
      DataEntryAtom(entry-version, entry-flags) data-entry;
   }
}
```

13.2.3.14.2 Semantics

`version` - is an integer that specifies the version.

`entry-count` - is an integer that counts the actual entries.

`entry-version` - is an integer that specifies the version.

`entry-flags` - is a 24-bit integer with flags; one flag is defined (x000001) which means that the media data is in the same file as the movie atom.

`data-entry` - is a URL or URN entry. Name is a URN, and is required in a URN entry. Location is a URL, and is required in a URL entry and optional in a URN entry, where it gives a default location to find the resource with the given name. Each is a null-terminated string using UTF-8 characters. If the self-contained flag is set; the URL form is used and no string is present; the atom terminates with the entry-flags field. The URL type should be of a service that delivers a file (e.g. URLs of type file, http, ftp etc.), which ideally also permits random access. Relative URLs are permissible and are relative to the file containing this data reference.

13.2.3.15 Sample Table atom

Atom Type: 'stbl'
Container: Media Information Atom ('minf')
Mandatory: Yes
Quantity: Exactly one

The sample table contains all the time and data indexing of the media samples in a track. Using the tables here, it is possible to locate samples in time, determine their type (e.g. I-frame or not), and determine their size, container, and offset into that container.

If the track that contains the sample table atom references no data, then the sample table atom does not need to contain any sub-atoms (this is not a very useful media track).

If the track that the sample table atom is contained in does reference data, then the following sub-atoms are required: Sample Description, Sample Size, Sample to Chunk, and Chunk Offset. All of the sub-tables of the sample table use the same total sample count. Further, the Sample Description Atom must contain at least one entry. A Sample Description Atom is required because it contains the data reference index field that indicates which Data Reference atom to use to retrieve the media samples. Without the Sample Description, it is not possible to determine where the media samples are stored. The Sync Sample atom is optional. If the Sync Sample atom is not present, all samples are sync samples.

13.2.3.15.1 Syntax

```
aligned(8) class SampleTableAtom extends Atom('stbl') {
 }
```

13.2.3.16 Time to Sample Atoms

ISO/IEC 14496 composition time (CT) and decoding time (DT) are derived from the Time to Sample Atoms of which there are two types. The decoding time is derived in the Decoding Time to Sample Atom decoding time deltas between successive decoding times. The composition times are derived in the Composition Time to Sample Atom as composition time offsets from decoding time. If the composition times and decoding times are identical for every sample in the track, then only the Decoding Time to Sample Atom is required.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000286

Note that the time to sample atoms must give durations for all samples including the last one. Durations in the 'stts' atom are strictly positive (non-zero). If the duration of the last sample is indeterminate, use an arbitrary small value and a 'dwell' edit.

In the following example, there is a sequence of I, P, and B frames, each with a decoding time delta of 10. The samples are stored as follows, with the indicated values for their decoding time deltas and composition time offsets (the actual CT and DT are given for reference). The re-ordering occurs because the predicted P frames must be decoded before the bi-directionally predicted B frames. The value of DT for a sample is always the sum of the deltas of the preceding samples. Note that the total of the decoding deltas is the duration of the media in this track.

Table 56 - Closed GOP Example

GOP	/--	---	---	---	---	---	--\	/--	---	---	---	---	---	--\
	I1	P4	B2	B3	P7	B5	B6	I8	P11	B9	B10	P14	B12	B13
DT	0	10	20	30	40	50	60	70	80	90	100	110	120	130
CT	10	40	20	30	70	50	60	80	110	90	100	140	120	130
Decode delta	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Composition Offset	10	30	0	0	30	0	0	10	30	0	0	30	0	0

Table 57 - Open GOP Example

GOP	/--	--	--	--	--	--\	/-	--	--	--	---	--\
	I3	B1	B2	P6	B4	B5	I9	B7	B8	P12	B10	B11
DT	0	10	20	30	40	50	60	70	80	90	100	110
CT	30	10	20	60	40	50	90	70	80	120	100	110
Decode Delta	10	10	10	10	10	10	10	10	10	10	10	10
Composition offset	30	0	0	30	0	0	30	0	0	30	0	0

13.2.3.16.1 Decoding Time to Sample atom

Atom Type: 'stts'
Container: Sample Table Atom ('stbl')
Mandatory: Yes
Quantity: Exactly one

This atom contains a compact version of a table that allows indexing from decoding time to sample number. Other tables give sample sizes and pointers, from the sample number. Each entry in the table gives the number of consecutive samples with the same time delta, and the delta of those samples. By adding the deltas a complete time-to-sample map may be built.

The Decoding Time to Sample Atom contains decode time delta's: DT(n+1) = DT(n) + STTS(n) where STTS(n) is the (uncompressed) table entry for sample n.

The sample entries are ordered by decoding time stamps; therefore the deltas are all non-negative.

The DT axis has a zero origin; DT(i) = SUM(for j=0 to i-1 of delta(j)), and the sum of all deltas gives the length of the media in the track (not mapped to the overall timescale, and not considering any edit list).

The Edit List Atom provides the initial CT value if it is non-empty (non-zero).

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000287

13.2.3.16.1.1 Syntax

```
aligned(8) class TimeToSampleAtom
   extends FullAtom('stts', version = 0, 0) {
   unsigned int(32)  entry-count;
      int i;
   for (i=0; i < entry-count; i++) {
      unsigned int(32)  sample-count;
      int(32)   sample-delta;
   }
}
```

For example with table 1.2, the entry would be:

Sample count	Sample-delta
14	10

13.2.3.16.1.2 Semantics

`version` - is an integer that specifies the version.

`ttype` - is 'stts' (for decoding times).

`entry-count` - is an integer that gives the number of entries in the following table.

`sample-count` - is an integer that counts the number of consecutive samples that have the given duration.

`sample-delta` - is an integer that gives the delta of these samples in the time-scale of the media.

13.2.3.16.2 Composition Time to Sample atom

Atom Type: 'ctts'
Container: Sample Table Atom ('stbl')
Mandatory: No
Quantity: Exactly one

This atom provides the offset between decoding time and composition time. Since decoding time must be less than the composition time, the offsets are expressed as unsigned numbers such that CT(n) = DT(n) + CTTS(n) where CTTS(n) is the (uncompressed) table entry for sample n.

The composition time to sample table is optional and should only be present if DT and CT differ for any samples.

13.2.3.16.2.1 Syntax

```
aligned(8) class CompositionOffsetAtom
   extends FullAtom('ctts', version = 0, 0) {
   unsigned int(32)  entry-count;
      int i;
   for (i=0; i < entry-count; i++) {
      unsigned int(32)  sample-count;
      unsigned int(32)  sample-offset;
   }
}
```

For example in table 1.2

Sample count	offset
1	10
1	30
2	0
1	30
2	0
1	10
1	30
2	0
1	30
2	0

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000288

13.2.3.16.2.2 Semantics

`version` - is an integer that specifies the version, 0 in this draft.

`ttype` - is 'ctts' (for composition times).

`entry-count` - is an integer that gives the number of entries in the following table.

`sample-count` - is an integer that counts the number of consecutive samples that have the given offset.

`sample-offset` - is a non-negative integer that gives the offset between CT and DT, such that CT(n) = DT(n) + CTTS(n).

13.2.3.17 Sample description atom

Atom Types: 'mp4v', 'mp4a', 'mp4s'
Container: Sample Table Atom ('stbl')
Mandatory: Yes
Quantity: Exactly one

The sample description table gives detailed information about the coding type used, and any initialization information needed for that coding.

The information stored in the data array is stream-type specific, and may have variants within a stream type (e.g. different codings may use different specific information after some common fields, even within a visual stream).

For visual streams, a VisualSampleEntry is used; for audio streams, an AudioSampleEntry. For all other MPEG-4 streams, a MpegSampleEntry is used. Hint tracks use an entry format specific to their protocol, with an appropriate name.

For all the MPEG-4 streams, the data field stores an ES_Descriptor with all its contents. Note that this provides an SlConfigDescriptor which uses a pre-defined value solely for use within files.

For hint tracks, the sample description contains appropriate declarative data for the TransMux being used, and the format of the hint track. The definition of the sample description is specific to the TransMux. However, note the discussion of FlexMux above, and the need for a Stream Map table, and MuxCode mode format definitions.

Note that multiple descriptions may be used within a stream.

13.2.3.17.1 Syntax

```
aligned(8) class ESDAtom
   extends FullAtom('esds', version = 0, 0) {
   ES_DescriptorES;
}
aligned(8) abstract class SampleEntry (unsigned int(32) format)
  extends Atom(format){
  const unsigned int(8)[6] reserved = 0;
  unsigned int(16) data-reference-index;
  }
class HintSampleEntry() extends SampleEntry (protocol) {
unsigned int(8) data [];
  }
   // Visual Streams

class VisualSampleEntry() extends SampleEntry ('mp4v'){const unsigned int(32)[4] reserved = 0;
   const unsigned int(32)  reserved = 0x014000F0;
   const unsigned int(32)  reserved = 0x00480000;
   const unsigned int(32)  reserved = 0x00480000;
   const unsigned int(32)  reserved = 0;
   const unsigned int(16)  reserved = 1;
   const unsigned int(8)[32] reserved = 0;
   const unsigned int(16)  reserved = 24;
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000289

```
   const int(16)reserved = -1;
   ESDAtom ES;
}
   // Audio Streams

class AudioSampleEntry() extends SampleEntry ('mp4a'){
  const unsigned int(32)[2] reserved = 0;
  const unsigned int(16) reserved = 2 ;
  const unsigned int(16) reserved = 16 ;
  const unsigned int(32) reserved = 0 ;
  unsigned int(16) time-scale ; // copied from track
  const unsigned int(16) reserved = 0 ;
  ESDAtom ES;
  }
   // all other Mpeg stream types
class MpegSampleEntry() extends SampleEntry ('mp4s'){
  ESDAtom ES;
  }
aligned(8) class SampleDescriptionAtom (unsigned int(32) handler-type)
  extends FullAtom('stsd', 0, 0){
  int i ;
  unsigned int(32) entry-count;
  for (i = 0 ; i < entry-count ; i++){
    switch (handler-type){
      case ‘soun’: // AudioStream
        AudioSampleEntry();
        break;
      case ‘vide’: // VisualStream
        VisualSampleEntry();
        break;
      case ‘hint’: // Hint track
        HintSampleEntbry();
        break;
      default :
        MpegSampleEntry();
        break;
   }
 }
```

13.2.3.17.2 Semantics

`version` - is an integer that specifies the version.

`entry-count` - is an integer that gives the number of entries in the following table.

`SampleEntry` - is the appropriate sample entry.

`data-reference-index` - is integer that contains the index of the data reference to use to retrieve data associated with samples that use this sample description. Data references are stored in data reference atoms. The index ranges from 1 to the number of data references.

`data` - is information specific to the protocol.

ES - is the ES Descriptor for this stream.

13.2.3.18 Sample size atom

Atom Type: ‘stsz’
Container: Sample Table Atom (‘stbl’)
Mandatory: Yes
Quantity: Exactly one

The sample size atom contains the sample count and a table giving the size of each sample. This allows the media data itself to be unframed. The total number of samples in the media is always indicated in the sample count. If the default size is indicated, then no table follows.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000290

13.2.3.18.1 Syntax

```
aligned(8) class SampleSizeAtom extends FullAtom('stsz', version = 0, 0) {
   unsigned int(32)  sample-size;
   unsigned int(32)  sample-count;
   if (sample-size==0) {
      int i;
      for (i=0; i < sample-count; i++) {
         unsigned int(32)  entry-size;
      }
   }
 }
```

13.2.3.18.2 Semantics

`version` - is an integer that specifies the version.

`sample-size` - is integer specifying the default sample size. If all the samples are the same size, this field contains that size value. If this field is set to 0, then the samples have different sizes, and those sizes are stored in the sample size table.

`sample-count` - is an integer that gives the number of entries in the following table.

`entry-size` - is integer specifying the size of a sample, indexed by its number.

13.2.3.19 Sample to chunk atom

Atom Type: 'stsc'
Container: Sample Table Atom ('stbl')
Mandatory: Yes
Quantity: Exactly one

Samples within the media data are grouped into chunks. Chunks may be of different sizes, and the samples within a chunk may have different sizes. By using this table, you can find the chunk that contains a sample, its position, and the associated sample description.

The table is compactly coded. Each entry gives the index of the first chunk of a run of chunks with the same characteristics; by subtracting one entry here from the previous one, you can compute how many chunks are in this run. You can convert this to a sample count by multiplying by the appropriate samples-per-chunk.

13.2.3.19.1 Syntax

```
aligned(8) class SampleToChunkAtom
   extends FullAtom('stsc', version = 0, 0) {
   unsigned int(32)  entry-count;
   int i;
   for (i=0; i < entry-count; i++) {
      unsigned int(32)  first-chunk;
      unsigned int(32)  samples-per-chunk;
      unsigned int(32)  sample-description-index;
   }
 }
```

13.2.3.19.2 Semantics

`version` - is an integer that specifies the version.

`entry-count` - is an integer that gives the number of entries in the following table.

`first-chunk` - is an integer that gives the index of the first chunk in this run of chunks that share the same samples-per-chunk and sample-description-index.

`samples-per-chunk` - is an integer that gives the number of samples in each of these chunks.

`sample-description-index` - is an integer that gives the index of the sample entry that describes the samples in this chunk. The index ranges from 1 to the number of sample entries in the sample description atom.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000291

13.2.3.20 Chunk offset atom

Atom Type: 'stco'
Container: Sample Table Atom ('stbl')
Mandatory: Yes
Quantity: Exactly one

The chunk-offset table gives the index of each chunk into the containing file. There are two variants, permitting the use of 32-bit or 64-bit offsets. The latter is useful when managing very large presentations. At most one of these variants will occur in any single instance of a sample table.

Note that offsets are file offsets not the offset into any atom within the file (e.g. a mdat atom). This permits referring to media data in files without any atom structure. It does also mean that care must be taken when constructing a self-contained mp4 file with its meta-data (movie atom) at the front, as the size of the movie atom will affect the chunk offsets to the media data.

13.2.3.20.1 Syntax

```
aligned(8) class ChunkOffsetAtom
   extends FullAtom('stco', version = 0, 0) {
   unsigned int(32)  entry-count;
   int i;
   for (i=0; i < entry-count; i++) {
      unsigned int(32)  chunk-offset;
   }
 }

aligned(8) class ChunkLargeOffsetAtom
   extends FullAtom('co64', version = 0, 0) {
   unsigned int(32)  entry-count;
   int i;
   for (i=0; i < entry-count; i++) {
      unsigned int(64)  chunk-offset;
   }
 }
```

13.2.3.20.2 Semantics

`version` - is an integer that specifies the version.

`entry-count` - is an integer that gives the number of entries in the following table.

`chunk-offset` - is a 32 or 64 bit integer that gives the offset of the start of a chunk into its containing media stream (file).

13.2.3.21 Sync Sample Atom

Atom Type: 'stss'
Container: Sample Table Atom ('stbl')
Mandatory: No
Quantity: Exactly one

The sync sample atom provides a compact marking of the random access points within the stream. Precisely the samples named here would have the RandomAccessPoint flag set in their SL Packet headers. The table is arranged in strictly increasing order of sample number.

If this table is not present, every sample is a random access point.

13.2.3.21.1 Syntax

```
aligned(8) class SyncSampleAtom
   extends FullAtom('stss', version = 0, 0) {
   unsigned int(32)  entry-count;
   int i;
   for (i=0; i < entry-count; i++) {
      unsigned int(32)  sample-number;
   }
 }
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000292

13.2.3.21.2 Semantics

`version` - is an integer that specifies the version.

`entry-count` - is an integer that gives the number of entries in the following table.

`sample-number` - gives the numbers of the samples that are random access points in the stream.

13.2.3.22 Shadow Sync Sample Atom

Atom Type: 'stsh'
Container: Sample Table Atom ('stbl')
Mandatory: No
Quantity: Exactly one

The shadow sync table provides an optional set of sync samples that can be used when seeking or for similar purposes. In normal forward play they are ignored.

Each entry in the ShadowSyncTable consists of a pair of sample numbers. The first entry (shadowed-sample-number) indicates the number of the sample that a shadow sync will be defined for. This should always be a non-sync sample (e.g. a frame difference). The second sample number (sync-sample-number) indicates the sample number of the sync sample (i.e. key frame) that can be used when there is a random access at, or before, the shadowed-sample-number.

The entries in the ShadowSyncAtom must be sorted based on the shadowed-sample-number field.

The shadow sync samples are normally placed in an area of the track that is not presented during normal play (edited out by means of an edit list), though this is not a requirement. Note that shadow sync table can be ignored and the track will play (and seek) correctly if it is ignored (though perhaps not optimally).

The ShadowSyncSample replaces, not augments, the sample which it shadows (i.e. the next sample sent is shadowed-sample-number+1). The shadow sync sample is treated as if it occurred at the time of the sample it shadows, having the duration of the sample it shadows.

Hinting and transmission might become more complex if a shadow sample is used also as part of normal playback, or is used more than once as a shadow. In this case the hint track might need separate shadow syncs, all of which can get their media data from the one shadow sync in the media track, to allow for the different time-stamps etc. needed in their headers.

13.2.3.22.1 Syntax

```
aligned(8) class ShadowSyncSampleAtom
   extends FullAtom('stsh', version = 0, 0) {
   unsigned int(32)  entry-count;
   int i;
   for (i=0; i < entry-count; i++) {
      unsigned int(32)  shadowed-sample-number;
      unsigned int(32)  sync-sample-number;
   }
 }
```

13.2.3.22.2 Semantics

`version` - is an integer that specifies the version.

`entry-count` - is an integer that gives the number of entries in the following table.

`shadowed-sample-number` - gives the number of a sample for which there is an alternative sync sample.

`sync-sample-number` - gives the number of the alternative sync sample.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000293

13.2.3.23 Degradation Priority Atom ('stdp')

Atom Type: 'stdp'
Container: Sample Table Atom ('stbl').
Mandatory: No.
Quantity: Exactly one.

The degradation priority atom contains the MPEG-4 degradation priority of each sample. The values are stored in the table, one for each sample. The size of the table, `sample-count` is taken from the `sample-count` in the Sample Size Atom ('stsz').

The maximum size of a degradation priority in the SL header is 15 bits, A fixed 15-bit field is used here.

13.2.3.23.1 Syntax

```
aligned(8) class DegradationPriorityAtom
   extends FullAtom('stdp', version = 0, 0) {
   int i;
   for (i=0; i < sample-count; i++) {
      const bit(1) pad = 0; // must be zero
      unsigned int(15)  priority;
   }
}
```

13.2.3.23.2 Semantics

`version` - is an integer that specifies the version.

`priority` - is integer specifying the degradation priority for each sample.

13.2.3.24 Free space atom

Atom Types: 'free', 'skip'
Container: File
Mandatory: No
Quantity: Any number

The contents of a free-space atom are irrelevant and may be ignored, or the object deleted, without affecting the presentation. (Note that deleting the object may invalidate the offsets used in the sample table, unless this object is after all the media data).

13.2.3.24.1 Syntax

```
aligned(8) class FreeSpaceAtom extends Atom(free-type) {
   unsigned int(8) data[];
 }
```

13.2.3.24.2 Semantics

`free-type` - may be 'free' or 'skip'.

13.2.3.25 Edit Atom

Atom Type: 'edts'
Container: Track Atom ('trak')
Mandatory: No
Quantity: Exactly one

An edit atom maps the presentation time-line to the media time-line as it is stored in the file. The edit atom is a container for the edit lists.

Note that the Edit atom is optional. In the absence of this atom, there is an implicit one-to-one mapping of these time-lines.

In the absence of an edit list, the presentation of a track starts immediately. An empty edit is used to offset the start time of a track.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000294

13.2.3.25.1 Syntax

```
aligned(8) class EditAtom extends Atom(‘edts’) {
 }
```

13.2.3.26 Edit List Atom

Atom Type: ‘elst’
Container: Edit Atom (‘edts’)
Mandatory: No
Quantity: 1 only

The edit list atom contains an explicit timeline map. It is possible to represent ‘empty’ parts if the timeline, where no media is presented; a ‘dwell’, where a single time-point in the media is held for a period; and a normal mapping.

Edit lists provide a mapping from the relative time (the deltas in the sample table) into absolute time (the time line of the presentation), possibly introducing ‘silent’ intervals or repeating pieces of media.

Starting offsets for tracks (streams) are represented by an initial empty edit. For example, to play a track from its start for 30 seconds, but at 10 seconds into the presentation, we have the following edit list:

Entry-count = 2

Segment-duration = 10 seconds
Media-Time = -1
Media-Rate = 1

Segment-duration = 30 seconds (could be the length of the whole track)
Media-Time = 0 seconds
Media-Rate = 1

13.2.3.26.1 Syntax

```
aligned(8) class EditListAtom extends FullAtom(‘elst’, version, 0) {
   unsigned int(32)  entry-count;
   int i;
   for (i=0; i < entry-count; i++) {
      if (version==1) {
         unsigned int(64) segment-duration;
         int(64) media-time;
      } else { // version==0
         unsigned int(32)  segment-duration;
         int(32) media-time;
      }
      int(16)   media-rate;
   const int(16) reserved = 0;
   }
 }
```

13.2.3.26.2 Semantics

`version` - is an integer that specifies the version.

`entry-count` - is an integer that gives the number of entries in the following table.

`segment-duration` - is an integer that specifies the duration of this edit segment in units of the movie’s time scale.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000295

`media-time` - is an integer containing the starting time within the media of this edit segment (in media time scale units, in composition time). If this field is set to –1, it is an empty edit. The last edit in a track should never be an empty edit. Any difference between the movie's duration and the track's duration is expressed as an implicit empty edit.

`media-rate` - specifies the relative rate at which to play the media corresponding to this edit segment. If this value is 0, then the edit is specifying a 'dwell': the media at media-time is presented for the segment-duration. Otherwise this field must contain the value 1.

13.2.3.27 User-data atom

Atom Type: 'udta'
Container: Movie Atom ('moov') or Track Atom ('trak')
Mandatory: No
Quantity: Any quantity

The stream user-data atom contains objects that declare user information about the containing atom and its data (presentation or stream).

The user-data atom is a container atom for informative user-data. This user data is formatted as a set of atoms with more specific atom types, which declare more precisely their content.

Only a copyright notice is defined in this draft. There may be multiple copyright atoms using different language codes.

13.2.3.27.1 Syntax

```
aligned(8) class UserDataAtom extends Atom('udta') {
 }
aligned(8) class CopyrightAtom
   extends FullAtom('cprt', version = 0, 0) {
   const bit(1) pad = 0;
   unsigned int(5)[3] language;  // packed ISO-639-2/T language code
   string  notice;
 }
```

13.2.3.27.2 Semantics

`language` - declares the language code for the following text. See ISO 639-2/T for the set of three character codes. Each character is packed as the difference between its ASCII value and 0x60. The code is confined to being three lower-case letters, so these values are strictly positive.

`notice` - is a null-terminated string giving a copyright notice.

13.3 Extensibility

13.3.1 Objects

The normative objects defined in this specification are identified by a 32-bit value, which is normally a set of four printable characters from the ISO 8859-1 character set.

To permit user extension of the format, to store new object types, and to permit the inter-operation of the files formatted to this specification with certain distributed computing environments, there is a type mapping and a type extension mechanism which together form a pair.

Commonly used in distributed computing are UUIDs (universal unique identifiers), which are 16 bytes. Any MPEG-4 normative type specified here may be mapped directly into the UUID space by composing the four byte type value with the twelve byte MPEG reserved value, 0xxxxxxxx-0011-0010-8000-00AA00389B71. The four-character code replaces the XXXXXXXX in the preceding number. These types are identified to MPEG as the object types used in this specification.

User objects use the escape type 'uuid'. They are documented above. After the size and type fields, there is a full 16-byte UUID.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000296

Systems which wish to treat every object as having a UUID should employ the following algorithm:

```
size := read_uint32();
type := read_uint32();
if (type==‘uuid’)
   then uuid := read_uuid()
   else uuid := form_uuid(type, MPEG_12_bytes);
```

Similarly when linearizing a set of objects into files formatted to this specification, the following is applied:

```
write_uint32( object_size(object) );
uuid := object_uuid_type(object);
if (is_MPEG_uuid(uuid) )
   write_uint32( MPEG_type_of(uuid) )
   else { write_uint32(‘uuid’); write_uuid(uuid); }
```

A file containing MPEG-4 objects which have been written using the ‘uuid’ escape and the full UUID is not compliant; systems may choose to read objects using the uuid escape and an MPEG-4 uuid as equivalent to the MPEG-4 object of the same type as equivalent, or not.

13.3.2 Elementary streams

MPEG-4 streams may be combined into a presentation with other streams. Such streams and their declarations are beyond the scope of this specification.

13.3.3 TransMuxes (protocols)

Hint tracks may be defined for a number of different protocols depending on the desired delivery mechanism. Examples would include RTP and MPEG-2 Transport.

13.3.4 Storage formats

The main file containing the meta-data may use other files to contain media-data. These other files may contain header declarations from a variety of standards, including this one.

If such a secondary file has a meta-data declaration set in it, that meta-data is not part of the overall presentation. This allows small presentation files to be aggregated into a larger overall presentation by building new meta-data and referencing the media-data, rather than copying it.

The references into these other files need not use all the data in those files; in this way, a subset of the media-data may be used, or unwanted headers ignored.

14 Syntactic Description Language

14.1 Introduction

This subclause describes the mechanism with which bitstream syntax is documented in ISO/IEC 14496. This mechanism is based on a Syntactic Description Language (SDL), documented here in the form of syntactic description rules. It directly extends the C-like syntax used in ISO/IEC 11172:1993 and ISO/IEC 13818:1996 into a well-defined framework that lends itself to object-oriented data representations. In particular, SDL assumes an object-oriented underlying framework in which bitstream units consist of “classes.” This framework is based on the typing system of the C++ and Java programming languages. SDL extends the typing system by providing facilities for defining bitstream-level quantities, and how they should be parsed.

The elementary constructs are described first, followed by the composite syntactic constructs, and arithmetic and logical expressions. Finally, syntactic control flow and built-in functions are addressed. Syntactic flow control is needed to take into account context-sensitive data. Several examples are used to clarify the structure.

14.2 Elementary Data Types

The SDL uses the following elementary data types:

1. Constant-length direct representation bit fields or Fixed Length Codes — FLCs. These describe the encoded value exactly as it is to be used by the appropriate decoding process.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000297

2. Variable length direct representation bit fields, or parametric FLCs. These are FLCs for which the actual length is determined by the context of the bitstream (e.g., the value of another parameter).

3. Constant-length indirect representation bit fields. These require an extra lookup into an appropriate table or variable to obtain the desired value or set of values.

4. Variable-length indirect representation bit fields (e.g., Huffman codes).

These elementary data types are described in more detail in the clauses to follow immediately.

All quantities shall be represented in the bitstream with the most significant byte first, and also with the most significant bit first.

14.2.1 Constant-Length Direct Representation Bit Fields

Constant-length direct representation bit fields shall be represented as:

Rule E.1: Elementary Data Types
[**aligned**] ***type***[**(***length***)**] *element_name* [= *value*]; // C++-style comments allowed

The ***type*** may be any of the following: **int** for signed integer, **unsigned int** for unsigned integer, **double** for floating point, and **bit** for raw binary data. The *length* attribute indicates the length of the element in bits, as it is actually stored in the bitstream. Note that a data ***type*** equal to **double** shall only use 32 or 64 bit lengths. The *value* attribute shall be present only when the value is fixed (e.g., start codes or object IDs), and it may also indicate a range of values (i.e., '0x01..0xAF'). The ***type*** and the optional *length* attributes are always present, except if the data is non-parsable, i.e., it is not included in the bitstream. The keyword **aligned** indicates that the data is aligned on a byte boundary. As an example, a start code would be represented as:

```
aligned bit(32) picture_start_code=0x00000100;
```

An optional numeric modifier, as in **aligned(**32**)**, may be used to signify alignment on other than byte boundary. Allowed values are 8, 16, 32, 64, and 128. Any skipped bits due to alignment shall have the value '0'. An entity such as temporal reference would be represented as:

```
unsigned int(5) temporal_reference;
```

where **unsigned int(***5***)** indicates that the element shall be interpreted as a 5-bit unsigned integer. By default, data shall be represented with the most significant bit first, and the most significant byte first.

The value of parsable variables with declarations that fall outside the flow of declarations (see 14.6) shall be set to 0.

Constants shall be defined using the keyword **const**.

EXAMPLE —

```
const int SOME_VALUE=255; // non-parsable constant
const bit(3) BIT_PATTERN=1;  // this is equivalent to the bit string "001"
```

To designate binary values, the **0b** prefix shall be used, similar to the **0x** prefix for hexadecimal numbers. A period ('.') may be optionally placed after every four digits for readability. Hence 0x0F is equivalent to 0b0000.1111.

In several instances, it may be desirable to examine the immediately following bits in the bitstream, without actually consuming these bits. To support this behavior, a '*****' character shall be placed after the parse size parentheses to modify the parse size semantics.

Rule E.2: Look-ahead parsing
[**aligned**] ***type*** **(***length***)*** *element_name*;

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000298

For example, the value of next 32 bits in the bitstream can be checked to be an unsigned integer without advancing the current position in the bitstream using the following representation:

```
aligned unsigned int (32)* next_code;
```

14.2.2 Variable Length Direct Representation Bit Fields

This case is covered by Rule E.1, by allowing the *length* attribute to be a variable included in the bitstream, a non-parsable variable, or an expression involving such variables.

EXAMPLE —

```
unsigned int(3) precision;
int(precision) DC;
```

14.2.3 Constant-Length Indirect Representation Bit Fields

Indirect representation indicates that the actual value of the element at hand is indirectly specified by the bitstream through the use of a table or map. In other words, the value extracted from the bitstream is an index to a table from which the final desired value is extracted. This indirection may be expressed by defining the map itself:

Rule E.3: Maps

map *MapName* **(***output_type***) {**
index, **{***value_1*, … *value_M***}**,
…
}

These tables are used to translate or map bits from the bitstream into a set of one or more values. The input type of a **map** (the *index* specified in the first column) shall always be **bit**. The *output_type* entry shall be either a predefined type or a defined class (classes are defined in 14.3.1). The **map** is defined as a set of pairs of such indices and values. Keys are binary string constants while values are *output_type* constants. Values shall be specified as aggregates surrounded by curly braces, similar to C or C++ structures.

EXAMPLE —

```
class YUVblocks {// classes are fully defined later on
   int Yblocks;
   int Ublocks;
   int Vblocks;
}

// a table that relates the chroma format with the number of blocks
// per signal component
map blocks_per_component (YUVblocks) {
   0b00,{4, 1, 1}, // 4:2:0
   0b01,{4, 2, 2}, // 4:2:2
   0b10,{4, 4, 4}  // 4:4:4
}
```

The next rule describes the use of such a **map**.

Rule E.4: Mapped Data Types

type **(***MapName***)** *name*;

The ***type*** of the variable shall be identical to the ***type*** returned from the **map**.

EXAMPLE —

```
YUVblocks(blocks_per_component) chroma_format;
```

Using the above declaration, a particular value of the **map** may be accessed using the construct: `chroma_format.Ublocks`.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000299

14.2.4 Variable Length Indirect Representation Bit Fields

For a variable length element utilizing a Huffman or variable length code table, an identical specification to the fixed length case shall be used:

```
class val {
   unsigned int foo;
   int bar;
}

map sample_vlc_map (val)  {
   0b0000.001,     {0, 5},
   0b0000.0001, {1, -14}
}
```

The only difference is that the indices of the **map** are now of variable length. The variable-length codewords are (as before) binary strings, expressed by default in '0b' or '0x' format, optionally using the period ('.') every four digits for readability.

Very often, variable length code tables are partially defined. Due to the large number of possible entries, it may be inefficient to keep using variable length codewords for all possible values. This necessitates the use of escape codes, that signal the subsequent use of a fixed-length (or even variable length) representation. To allow for such exceptions, parsable type declarations are allowed for **map** values.

EXAMPLE — This example uses the class type 'val' as defined above.

```
map sample_map_with_esc (val)  {
   0b0000.001,     {0, 5},
   0b0000.0001, {1, -14},
   0b0000.0000.1,  {5, int(32)},
   0b0000.0000.0,  {0, -20}
}
```

When the codeword 0b0000.0000.1 is encountered in the bitstream, then the value '5' is assigned to the first element (`val.foo`). The following 32 bits are parsed and assigned as the value of the second element (`val.bar`). Note that, in case more than one element utilizes a parsable type declaration, the order is significant and is the order in which elements are parsed. In addition, the type within the **map** declaration shall match the type used in the class declaration associated with the **map**'s return type.

14.3 Composite Data Types

14.3.1 Classes

Classes are the mechanism with which definitions of composite types or objects is performed. Their definition is as follows.

Rule C.1: Classes

[**aligned**] [**abstract**] [**expandable**[**(***maxClassSize***)**]] **class** *object_name* [**extends** *parent_class*] [:
bit(*length***)** [*id_name*=] *object_id* | *id_range*] **{**
[*element*; …] // zero or more elements
}

The different elements within the curly braces are the definitions of the elementary bitstream components discussed in 12.2 or control flow elements that will be discussed in a subsequent subclause.

The optional keyword **extends** specifies that the **class** is "derived" from another **class**. Derivation implies that all information present in the base **class** is also present in the derived **class**, and that, in the bitstream, all such information *precedes* any additional bitstream syntax declarations specified in the new **class**.

The optional attribute *id_name* allows to assign an *object_id*, and, if present, is the key demultiplexing entity which allows differentiation between base and derived objects. It is also possible to have a range of possible values: the *id_range* is specified as *start_id .. end_id*, inclusive of both bounds.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000300

If the attribute id_name is used, a derived **class** may appear at any point in the bitstream where its base **class** is specified in the syntax. This allows to express polymorphism in the SDL syntax description. The actual **class** to be parsed is determined as follows:

- The base **class** declaration shall assign a constant value or range of values to *object_id*.
- Each derived **class** declaration shall assign a constant value or ranges of values to *object_id*. This value or set of values shall correspond to legal *object_id* value(s) for the base **class**.

NOTE 1 — Derivation of classes is possible even when object_ids are not used. However, in that case derived classes may not replace their base **class** in the bitstream.

NOTE 2 — Derived classes may use the same *object_id* value as the base **class**. In that case classes can only be discriminated through context information.

EXAMPLE —

```
class slice: aligned bit(32) slice_start_code=0x00000101 .. 0x000001AF {
   // here we get vertical_size_extension, if present
   if (scalable_mode==DATA_PARTITIONING) {
      unsigned int(7) priority_breakpoint;
   }
   …
}

class foo {
   int(3) a;
   ...
}

class bar extends foo {
   int(5) b; // this b is preceded by the 3 bits of a
   int(10) c;
   ...
}
```

The order of declaration of the bitstream components is important: it is the same order in which the elements appear in the bitstream. In the above examples, `bar.b` immediately precedes `bar.c` in the bitstream.

Objects may also be encapsulated within other objects. In this case, the *element* in Rule C.1 is an object itself.

14.3.2 Abstract Classes

When the **abstract** keyword is used in the **class** declaration, it indicates that only derived classes of this **class** shall be present in the bitstream. This implies that the derived classes may use the entire range of IDs available. The declaration of the abstract **class** requires a declaration of an ID, with the value 0.

EXAMPLE —

```
abstract class Foo : bit(1) id=0 { // the value 0 is not really used
   ...
}

// derived classes are free to use the entire range of IDs
class Foo0 extends Foo : bit(1) id=0 {
   ...
}

class Foo1 extends Foo : bit(1) id=1 {
   ...
}

class Example {
   Foo f;  // can only be Foo0 or Foo1, not Foo
}
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000301

14.3.3 Expandable classes

When the **expandable** keyword is used in the **class** declaration, it indicates that the **class** may contain implicit arrays or undefined trailing data, called the "expansion". In this case the **class** encodes its own size in bytes explicitly. This may be used for classes that require future compatible extension or that may include private data. A legacy device is able to decode an expandable **class** up to the last parsable variable that has been defined for a given revision of this **class**. Using the size information, the parser shall skip the **class** data following the last known syntax element. Anywhere in the syntax where a set of expandable classes with *object_id* is expected it is permissible to intersperse expandable classes with unknown *object_id* values. These classes shall be skipped, using the size information.

The size encoding precedes any parsable variables of the **class**. If the **class** has an *object_id*, the encoding of the *object_id* precedes the size encoding. The size information shall not include the number of bytes needed for the size and the *object_id* encoding. Instances of expandable classes shall always have a size corresponding to an integer number of bytes. The size information is accessible within the class as class instance variable `sizeOfInstance`.

If the **expandable** keyword has a *maxClassSize* attribute, then this indicates the maximum permissible size of this **class** in bytes, including any expansion.

The length encoding is itself defined in SDL as follows:

```
int sizeOfInstance = 0;
bit(1) nextByte;
bit(7) sizeOfInstance;
while(nextByte) {
   bit(1) nextByte;
   bit(7) sizeByte;
   sizeOfInstance = sizeOfInstance<<7 | sizeByte;
}
```

14.3.4 Parameter types

A parameter type defines a **class** with parameters. This is to address cases where the data structure of the **class** depends on variables of one or more other objects. Since SDL follows a declarative approach, references to other objects, in such cases, cannot be performed directly (none is instantiated). Parameter types provide placeholders for such references, in the same way as the arguments in a C function declaration. The syntax of a **class** definition with parameters is as follows.

Rule C.2: Class Parameter Types

[**aligned**] [**abstract**] **class** *object_name* [**(***parameter list***)**] [**extends** *parent_class*]
[: **bit(***length***)** [*id_name*=] *object_id* | *id_range*] **{**
[*element*; …] // zero or more elements
}

The parameter list is a list of ***type*** names and variable name pairs separated by commas. Any element of the bitstream, or value derived from the bitstream with a variable-length codeword, or a constant, can be passed as a parameter.

A **class** that uses parameter types is dependent on the objects in its parameter list, whether **class** objects or simple variables. When instantiating such a **class** into an object, the parameters have to be instantiated objects of their corresponding classes or types.

EXAMPLE —

```
class A {
   // class body
   ...
   unsigned int(4) format;
}
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000302

```
class B (A a, int i) {    // B uses parameter types
   unsigned int(i) bar;
   ...
   if( a.format == SOME_FORMAT ) {
      ...
   }
   ...
}

class C {
   int(2) i;
   A a;
   B foo( a, I); // instantiated parameters are required
}
```

14.3.5 Arrays

Arrays are defined in a similar way as in C/C++, i.e., using square brackets. Their length, however, can depend on run-time parameters such as other bitstream values or expressions that involve such values. The array declaration is applicable to both elementary as well as composite objects.

Rule A.1: Arrays

`typespec` *name* **`[`***length***`]`**;

`typespec` is a ***`type`*** specification (including bitstream representation information, e.g. '**`int(`**2**`)`**'). The attribute *name* is the name of the array, and *length* is its length.

EXAMPLE —

```
unsigned int(4) a[5];
int(10) b;
int(2) c[b];
```

Here 'a' is an array of 5 elements, each of which is represented using 4 bits in the bitstream and interpreted as an unsigned integer. In the case of 'c', its length depends on the actual value of 'b'. Multi-dimensional arrays are allowed as well. The parsing order from the bitstream corresponds to scanning the array by incrementing first the right-most index of the array, then the second, and so on.

14.3.6 Partial Arrays

In several situations, it is desirable to load the values of an array one by one, in order to check, for example, a terminating or other condition. For this purpose, an extended array declaration is allowed in which individual elements of the array may be accessed.

Rule A.2: Partial Arrays

`typespec` *name***`[[`***index***`]]`**;

Here *index* is the element of the array that is defined. Several such partial definitions may be given, but they shall all agree on the ***`type`*** specification. This notation is also valid for multidimensional arrays.

EXAMPLE —

```
int(4) a[[3]][[5]];
```

indicates the element a(5, 3) of the array (the element in the 6th row and the 4th column), while

```
int(4) a[3][[5]];
```

indicates the entire sixth column of the array, and

```
int(4) a[[3]][5];
```

indicates the entire fourth row of the array, with a length of 5 elements.

NOTE — **a[***5***]** means that the array has five elements, whereas **a[[***5***]]** implies that there are at least six.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000303

14.3.7 Implicit Arrays

When a series of polymorphic classes is present in the bitstream, it may be represented as an array of the same type as that of the base **class**. Let us assume that a set of polymorphic classes is defined, derived from the base **class** Foo (may or may not be abstract):

```
class Foo : int(16) id = 0 {
   ...
}
```

For an array of such objects, it is possible to implicitly determine the length by examining the validity of the **class** ID. Objects are inserted in the array as long as the ID can be properly resolved to one of the IDs defined in the base (if not abstract) or its derived classes. This behavior is indicated by an array declaration without a length specification.

EXAMPLE 1 —

```
class Example {
   Foo f[];   // length implicitly obtained via ID resolution
}
```

To limit the minimum and maximum length of the array, a range specification may be inserted in the specification of the length.

EXAMPLE 2 —

```
class Example {
   Foo f[1 .. 255];  // at least 1, at most 255 elements
}
```

In this example, ‘f’ may have at least 1 and at most 255 elements.

14.4 Arithmetic and Logical Expressions

All standard arithmetic and logical operators of C++ are allowed, including their precedence rules.

14.5 Non-Parsable Variables

In order to accommodate complex syntactic constructs, in which context information cannot be directly obtained from the bitstream but only as a result of a non-trivial computation, non-parsable variables are allowed. These are strictly of local scope to the **class** they are defined in. They may be used in expressions and conditions in the same way as bitstream-level variables. In the following example, the number of non-zero elements of an array is computed.

```
unsigned int(6) size;
int(4) array[size];
…
int i; // this is a temporary, non-parsable variable
for (i=0, n=0; i<size; i++) {
   if (array[[i]]!=0)
      n++;
}

int(3) coefficients[n];
// read as many coefficients as there are non-zero elements in array
```

14.6 Syntactic Flow Control

The syntactic flow control provides constructs that allow conditional parsing, depending on context, as well as repetitive parsing. The familiar C/C++ if-then-else construct is used for testing conditions. Similarly to C/C++, zero corresponds to false, and non-zero corresponds to true.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000304

Rule FC.1: Flow Control Using If-Then-Else

```
if (condition) {
   …
}[else if (condition) {
   …
}][else {
   …
}]
```

EXAMPLE 1 —

```
class conditional_object {
   unsigned int(3) foo;
   bit(1) bar_flag;
   if (bar_flag) {
      unsigned int(8) bar;
   }
   unsigned int(32) more_foo;
}
```

Here the presence of the entity ‘bar’ is determined by the ‘bar_flag’.

EXAMPLE 2 —

```
class conditional_object {
   unsigned int(3) foo;
   bit(1) bar_flag;
   if (bar_flag) {
      unsigned int(8) bar;
   } else {
      unsigned int(some_vlc_table) bar;
   }
   unsigned int(32) more_foo;
}
```

Here we allow two different representations for ‘bar’, depending on the value of ‘bar_flag’. We could equally well have another entity instead of the second version (the variable length one) of ‘bar’ (another object, or another variable). Note that the use of a flag necessitates its declaration before the conditional is encountered. Also, if a variable appears twice (as in the example above), the types shall be identical.

In order to facilitate cascades of if-then-else constructs, the ‘switch’ statement is also allowed.

Rule FC.2: Flow Control Using Switch

```
switch (condition) {
        [case label1: …]
        [default:]
}
```

The same category of context-sensitive objects also includes iterative definitions of objects. These simply imply the repetitive use of the same syntax to parse the bitstream, until some condition is met (it is the conditional repetition that implies context, but fixed repetitions are obviously treated the same way). The familiar structures of ‘for’, ‘while’, and ‘do’ loops can be used for this purpose.

Rule FC.3: Flow Control Using For

```
for (expression1; expression2; expression3) {
        …
}
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000305

expression1 is executed prior to starting the repetitions. Then *expression2* is evaluated, and if it is non-zero (true) the declarations within the braces are executed, followed by the execution of *expression3*. The process repeats until *expression2* evaluates to zero (false).

Note that it is not allowed to include a variable declaration in *expression1* (in contrast to C++).

Rule FC.4: Flow Control Using Do

```
do {
        …
} while (condition);
```

Here the block of statements is executed until *condition* evaluates to false. Note that the block will be executed at least once.

Rule FC.5: Flow Control Using While

```
while (condition) {
        …
}
```

The block is executed zero or more times, as long as *condition* evalutes to non-zero (true).

14.7 Built-In Operators

The following built-in operators are defined.

Rule O.1: lengthof() Operator

```
lengthof(variable)
```

This operator returns the length, in bits, of the quantity contained in parentheses. The length is the number of bits that was most recently used to parse the quantity at hand. A return value of 0 means that no bits were parsed for this variable.

14.8 Scoping Rules

All parsable variables have class scope, i.e., they are available as class member variables.

For non-parsable variables, the usual C++/Java scoping rules are followed (a new scope is introduced by curly braces: '{' and '}'). In particular, only variables declared in class scope are considered class member variables, and are thus available in objects of that particular type.

15 Profiles

15.1 Introduction

This clause 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 object descriptor profiles (OD profiles) specify the allowed configurations of the object descriptor tool and the sync layer tool. 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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000306

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.

15.2 OD Profile Definitions

15.2.1 Overview

The object descriptor profiles (OD profiles) specify the configurations of the object descriptor tool and the sync layer tool that are allowed. The object descriptor tool provides a structure for all descriptive information. The sync layer tool provides the syntax to convey, among others, timing information for elementary streams. object descriptor profiles are used, in particular, to reduce the amount of asynchronous operations as well as the amount of permanent storage.

15.2.2 OD Profiles Tools

The following tools are available to construct OD profiles:

- Object descriptor (OD) tool as defined in 8.5.
- Sync layer (SL) tool as defined in 10.2.
- Object content information (OCI) tool as defined in 8.4.
- Intellectual property management and protection (IPMP) tool as defined in 8.3.

15.2.3 OD Profiles

The OD profiles are defined in the following table. Currently, only one profile is defined, comprising all the tools. No additional profiles are foreseen at the moment, but the possibility of adding Profiles through amendments is left open.

Table 58 - OD Profiles

	OD Profiles
OD Tools	**Core**
SL	X
OD	X
OCI	X
IPMP	X

Decoders that claim compliance to a given profile shall implement all the tools with an 'X' entry for that profile.

15.2.4 OD Profiles@Levels

15.2.4.1 Levels for the Core Profile

No levels are defined yet for the OD Core profile. Future definition of Levels is anticipated; this will happen by means of an amendment to this part of the standard.

15.3 Scene Graph Profile Definitions

15.3.1 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000307

15.3.2 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 59.
- BIFS commands and BIFS animation as defined in 9.3.6 and 0, respectively.
- BIFS ROUTES as defined in 9.3.7.47.1.

15.3.3 Scene Graph Profiles

The following table defines the scene graph profiles:

Table 59 - Scene graph profiles

	Scene Graph Profiles			
Scene Graph Tools	**Audio**	**Simple 2D**	**Complete 2D**	**Complete**
Anchor			X	X
AudioBuffer	X		X	X
AudioDelay	X		X	X
AudioFX	X		X	X
AudioMix	X		X	X
AudioSwitch	X		X	X
Billboard				X
Collision				X
Composite2DTexture			X	X
Composite3DTexture				X
Form			X	X
Group	X	X	X	X
Inline			X	X
Layer2D			X	X
Layer3D				X
Layout			X	X
ListeningPoint			X	X
LOD				X
NavigationInfo				X
OrderedGroup		X	X	X
QuantizationParameter			X	X
Sound				X
Sound2D	X	X	X	X
Switch			X	X
Transform				X
Transform2D		X	X	X
Viewpoint				X
WorldInfo			X	X
Node Update			X	X
Route Update			X	X
Scene Update	X	X	X	X
AnimationStream			X	X
Script			?	X
ColorInterpolator			X	X
Conditional			X	X
CoordinateInterpolator2D			X	X
CoordinateInterpolator				X

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000308

CylinderSensor				X
DiscSensor			X	X
NormalInterpolator				X
OrientationInterpolator				X
PlaneSensor2D			X	X
PlaneSensor				X
PositionInterpolator				X
PositionInterpolator2D			X	X
ProximitySensor				X
ProximitySensor2D			X	X
ROUTE			X	X
ScalarInterpolator			X	X
SphereSensor				X
TermCap			X	X
TimeSensor			X	X
TouchSensor			X	X
VisibilitySensor				X
Valuator			X	X

Decoders that claim compliance to a given profile shall implement all the tools with an ‘X’ entry for that profile.

15.3.3.1 BIFS nodes for audio objects

The presence of AudioClip and AudioSource nodes in BIFS scene graph depends on the selected Audio profile. The following table describes what nodes are allowed in the BIFS scene graph depending on the Audio profile.

Table 60 - BIFS nodes for audio objects

Audio Profiles	Allowed Audio Object Nodes
Main	AudioClip, AudioSource
Scalable	AudioClip, AudioSource
Speech	AudioClip, AudioSource
Low Rate Synthesis	AudioClip, AudioSource

15.3.3.2 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 61 - 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
Simple Scalable	ImageTexture, Background2D, Background, MovieTexture
N-Bit	ImageTexture, Background2D, Background, MovieTexture
Hybrid	ImageTexture, Background2D, Background, MovieTexture, Face, Expression, FAP, FDP, FIT, FaceDefMesh, FaceDefTable, FaceDefTransform, Viseme

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000309

Basic Animated Texture	ImageTexture, Background2D, Background, Face, Expression, FAP, FDP, FIT, FaceDefMesh, FaceDefTable, FaceDefTransform, Viseme
Scaleable Texture	ImageTexture, Background2D, Background
Simple Face	Face, Expression, FAP, FDP, FIT, FaceDefMesh, FaceDefTable, FaceDefTransform, Viseme

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.

15.3.4 Scene Graph Profiles@Levels

15.3.4.1 Levels for the Audio Scene Graph Profile

15.3.4.1.1 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.

15.3.4.1.2 Levels

No levels are yet defined for the Audio scene graph profile. Future definition of Levels is anticipated; this will happen by means of an amendment to this part of the standard.

15.3.4.2 Levels for the Simple 2D Scene Graph Profile

15.3.4.2.1 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.

15.3.4.2.2 Level 1

The following restrictions apply for the Simple 2D scene graph profile at Level 1:

Table 62 - 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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000310

15.3.4.3 Levels for the Complete 2D Scene Graph Profile

15.3.4.3.1 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.

15.3.4.3.2 Levels

No levels are yet defined for the Complete 2D scene graph profile. Future definition of Levels is anticipated; this will happen by means of an amendment to this part of the standard.

15.3.4.4 Levels for the Complete Scene Graph Profile

15.3.4.4.1 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.

15.3.4.4.2 Levels

No levels are yet defined for the Complete scene graph profile. Future definition of Levels is anticipated; this will happen by means of an amendment to this part of the standard.

15.4 Graphics Profile Definitions

15.4.1 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.

15.4.2 Graphics Profiles Tools

The following tools are available to construct the graphics profiles:

- BIFS nodes related to graphics as defined in Table 63.

15.4.3 Graphics Profiles

The following table defines the graphics profiles:

Table 63 - Graphics profiles

	Graphics Profiles		
Graphics Tools	**Simple 2D**	**Complete 2D**	**Complete**
Appearance	X	X	X
Box			X
Bitmap	X	X	X
Background			X
Background2D		X	X
Circle		X	X
Color		X	X
Cone			X
Coordinate			X
Coordinate2D		X	X
Curve2D		X	X
Cylinder			X

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000311

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
IndexedFaceSet			X
IndexedFaceSet2D		X	X
IndexedLineSet			X
IndexedLineSet2D		X	X
LineProperties		X	X
Material			X
Material2D		X	X
Normal			X
PixelTexture		X	X
PointLight			X
PointSet			X
PointSet2D		X	X
Rectangle		X	X
Shape	X	X	X
Sphere			X
SpotLight			X
Text		X	X
TextureCoordinate		X	X
TextureTransform		X	X
Viseme			X

Decoders that claim compliance to a given profile shall implement all the tools with an ‘X’ entry for that profile.

15.4.4 Graphics Profiles@Levels

15.4.4.1 Levels for the Simple 2D Graphics Profile

15.4.4.1.1 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.

15.4.4.1.2 Levels

No levels are yet defined for the Simple 2D graphics profile. Future definition of Levels is anticipated; this will happen by means of an amendment to this part of the standard.

15.4.4.2 Levels for the Complete 2D Graphics Profile

15.4.4.2.1 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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000312

15.4.4.2.2 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.

15.4.4.3 Levels for the Complete Graphics Profile

15.4.4.3.1 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.

15.4.4.3.2 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.

15.5 MPEG-J Profile Definitions

15.5.1 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.

15.6 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 11.5.3.
- Resource APIs (package org.iso.mpeg.mpegj.resource) as defined in 11.5.4.
- Net APIs (package org.iso.mpeg.mpegj.net) as defined in 11.5.6.
- Decoder APIs (package org.iso.mpeg.mpegj.decoder) as defined in 11.5.5.
- Section Filtering and Service Information as defined in 11.5.7.
- Please note that the package org.iso.mpeg.mpegj is required in all terminals.

15.7 MPEG-J Profiles

The MPEG-J profiles are defined in Table 64. 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000313

Table 64 - 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).

15.8 MPEG-J Profiles@Levels

15.8.1 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.

15.8.2 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 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000314

Annex A
(informative)

Bibliography

[1] A. Eleftheriadis, “Flavor: A Language for Media Representation,” *Proceedings, ACM Multimedia ’97 Conference*, Seattle, Washington, November 1997, pp. 1–9.

[2] C. Herpel, “Elementary Stream Management in MPEG-4,” *IEEE Trans. on Circuits and Systems for Video Technology*, 1998 (to appear).

[3] Flavor Web Site, http://www.ee.columbia.edu/flavor.

[4] R. Koenen, F. Pereira, and L. Chiariglione, “MPEG-4: Context and Objectives,” *Signal Processing: Image Communication*, Special Issue on MPEG-4, Vol. 9, Nr. 4, May 1997.

[5] F. Pereira, and R. Koenen, “Very Low Bitrate Audio-Visual Applications,” *Signal Processing: Image Communication*, Vol. 9, Nr. 1, November 1996, pp. 55-77.

[6] A. Puri and A. Eleftheriadis, “MPEG-4: An Object-Based Multimedia Coding Standard Supporting Mobile Application,” *ACM Mobile Networks and Applications Journal*, 1998 (to appear).

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000315

Annex B
(informative)

Time Base Reconstruction

B.1 Time Base Reconstruction

The time stamps present in the sync layer are the means to synchronize events related to decoding, composition and overall buffer management. In particular, the clock references are the sole means of reconstructing the sending terminal's clock at the receiving terminal, when required (e.g., for broadcast applications). A normative method for this reconstruction is not specified. The following describes the process at a conceptual level.

B.1.1 Adjusting the Receiving Terminal's OTB

Each elementary stream may be generated by an encoder at the sending terminal with a different object time base (OTB). For each stream that conveys OCR information, it is possible for the receiving terminal to adjust a local OTB to the sending terminals' OTB. This is done by using well-known PLL techniques. The notion of time for each data stream can therefore be recovered at the receiving end.

B.1.2 Mapping Time Stamps to the STB

The OTBs of all data streams may run at a different speed than the STB of the receiving terminal. Therefore, a method is needed to map the value of time stamps expressed in any OTB to the STB of the receiving terminal. This step may be done jointly with the recovery of individual OTB's as described in the previous subclause.

Note that the receiving terminals' system time base need not be locked to any of the available object time bases.

The composition time t_{SCT} of a composition unit, expressed in terms of STB of the receiving terminal, can be calculated from the composition time stamp value t_{OCT}, expressed in terms of the OTB of the relevant sending terminal, by a linear transformation:

$$t_{SCT} = \frac{\Delta t_{STB}}{\Delta t_{OTB}} \cdot t_{OCT} - \frac{\Delta t_{STB}}{\Delta t_{OTB}} \cdot t_{OTB-START} + t_{STB-START}$$

with:

t_{SCT}	composition time of a composition unit measured in units of t_{STB}
t_{STB}	current time in the receiving terminal's STB
t_{OCT}	composition time of a composition unit measured in units of t_{OTB}
t_{OTB}	current time in the data stream's OTB, conveyed by an OCR
$t_{STB-START}$	value of receiving terminal's STB when the first byte of the OCR time stamp of the data stream is encountered
$t_{OTB-START}$	value of the first OCR time stamp of the data stream

$$\Delta t_{OTB} = t_{OTB} - t_{OTB-START}$$

$$\Delta t_{STB} = t_{STB} - t_{STB-START}$$

The quotient $\Delta t_{STB} / \Delta t_{OTB}$ is the instantaneous scaling factor between the two time bases. In cases where the clock speed and resolution of the sending terminal and of the receiving terminal are nominally identical, this quotient is very near 1. To avoid long term rounding errors, the quotient $\Delta t_{STB} / \Delta t_{OTB}$ should always be recalculated whenever the formula is applied to a newly received composition time stamp. The quotient can be updated each time an OCR time stamp is encountered.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000316

A similar formula can be derived for decoding times by replacing composition time stamps with decoding time stamps. If time stamps for some access units or composition units are only known implicitly, e.g., given by known update rates, these have to be mapped with the same mechanism.

With this procedure it is possible to synchronize the STB at a receiving terminal to several OTBs so that correct decoding and composition from several data streams is possible.

B.1.3 Adjusting the STB to an OTB

When all data streams in a presentation use the same OTB, it is possible to lock the STB at the receiving terminal to this OTB using well-known PLL techniques. In this case the mapping described in the previous subclause is not necessary and the following mapping may be used.

$$t_{STB-START} = t_{OTB-START}$$
$$\Delta t_{STB} = \Delta t_{OTB}$$
$$t_{SCT} = t_{OCT}$$

B.1.4 System Operation without Object Time Base

If a time base for an elementary stream is neither conveyed by OCR information nor derived from another elementary stream, time stamps can still be used by a receiving terminal but not in applications that require flow-control. For example, file-based playback may not require time base reconstruction: time stamps alone are sufficient for synchronization if a single time base is assumed for all the data streams.

In the absence of time stamps, the receiving terminal may only operate under the assumption that each access unit is to be decoded and presented as soon as it is received. In this case the systems decoder model does not apply and cannot be used as a model for the terminal's behavior.

In the case that a universal clock is available which can be shared between peer terminals, it may be used as a common time base. It is then possible to use the systems decoder model without explicit OCR transmission. The procedures for doing so are application-dependent and are not defined in ISO/IEC 14496-1.

B.2 Temporal aliasing and audio resampling

A receiving terminal compliant with ISO/IEC 14496 is not required to synchronize decoding of AUs and composition of CUs. In other words, its STB does not have to be identical to any of the OTBs of received data streams. The number of decoded and actually presented (displayed/played back) units per second may therefore differ. Temporal aliasing may then manifest itself as composition units being either presented multiple times or skipped.

If audio signals are encoded on a system with an OTB different from the STB of the receiving terminal, even nominally identical sampling rates of the audio samples may not match exactly, so that audio samples may be dropped or repeated.

Proper re-sampling techniques may of course in both cases be applied at the receiving terminal.

B.3 Reconstruction of a Synchronised Audio-visual Scene: A Walkthrough

The different steps to reconstruct a synchronized scene are as follows:

1. The time base for each data stream is recovered either from the OCR conveyed with the SL-packetized elementary stream of this data stream or from another data stream present in the presentation.

2. Object time stamps are mapped to the STB of the receiving terminal according to a suitable algorithm (e.g., the one detailed above).

3. Received access units are placed in the decoding buffer.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000317

4. Each access unit is instantaneously decoded by the decoder at instants of time (in terms of the receiver terminal's STB) corresponding to its implicit or explicit DTS and the resulting one or more composition units are placed in the composition memory.

5. The compositor may access each CU at time instants between the one corresponding its CTS and the one corresponding to the CTS of the subsequent CU.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000318

Annex C
(normative)

View Dependent Object Scalability

C.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.

C.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.

C.3.1 specifies the bitstream syntax for a View Dependent Object which initializes the session at the upstream data decoder. C.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.

C.2.1 View Dependent Object

```
class ViewDependentObject {
   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;
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000319

```
   }
}

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;
}
```

C.3 Bitstream Semantics

C.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.

C.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.

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).

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000320

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 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000321

Annex D
(informative)

Registration procedure

D.1 Procedure for the request of a Registration ID (RID)

Requesters of a RID shall apply to the Registration Authority. Registration forms shall be available from the Registration Authority. The requester shall provide the information specified in D.3. Companies and organizations are eligible to apply.

D.2 Responsibilities of the Registration Authority

The primary responsibilities of the Registration Authority administrating the registration of either the private data format identifiers or the IPMP system type values are outlined in this annex; certain other responsibilities may be found in the JTC 1 Directives. The Registration Authority shall:

a) implement a registration procedure for application for a unique RID in accordance with the JTC 1 Directives;

b) receive and process the applications for allocation of an identifier from application providers;

c) ascertain which applications received are in accordance with this registration procedure, and to inform the requester within 30 days of receipt of the application of their assigned RID;

d) inform application providers whose request is denied in writing with 30 days of receipt of the application, and to consider resubmissions of the application in a timely manner;

e) maintain an accurate register of the allocated identifiers. Revisions to format specifications shall be accepted and maintained by the Registration Authority;

f) make the contents of this register available upon request to National Bodies of JTC 1 that are members of ISO or IEC, to liaison organizations of ISO or IEC and to any interested party;

g) maintain a data base of RID request forms, granted and denied. Parties seeking technical information on the format of private data which has a RID shall have access to such information which is part of the data base maintained by the Registration Authority.;

h) report its activities annually to JTC 1, the ITTF, and the SC 29 Secretariat, or their respective designees; and

i) accommodate the use of existing RIDs whenever possible.

D.3 Contact information for the Registration Authority

To Be Determined

D.4 Responsibilities of Parties Requesting a RID

The party requesting a format identifier or an IPMP system type identifier shall:

a) apply using the Form and procedures supplied by the Registration Authority;

b) include a description of the purpose of the registered bitstream, and the required technical details as specified in the application form;

c) provide contact information describing how a complete description can be obtained on a non-discriminatory basis;

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000322

d) agree to institute the intended use of the granted RID within a reasonable time frame; and

e) to maintain a permanent record of the application form and the notification received from the Registration Authority of a granted RID.

D.5 Appeal Procedure for Denied Applications

The Registration Management Group is formed to have jurisdiction over appeals to denied request for a RID. The RMG shall have a membership who is nominated by P- and L-members of the ISO technical committee responsible for ISO/IEC 14496. It shall have a convenor and secretariat nominated from its members. The Registration Authority is entitled to nominate one non-voting observing member.

The responsibilities of the RMG shall be:

a) to review and act on all appeals within a reasonable time frame;

b) to inform, in writing, organizations which make an appeal for reconsideration of its petition of the RMGs disposition of the matter;

c) to review the annual report of the Registration Authorities summary of activities; and

d) to supply Member Bodies of ISO and National Committees of IEC with information concerning the scope of operation of the Registration Authority.

D.6 Registration Application Form

D.6.1 Contact Information of organization requesting a RID

Organization Name:
Address:

Telephone:
Fax:
E-mail:
Telex:

D.6.2 Request for a specific RID

NOTE — If the system has already been implemented and is in use, fill in this item and item D.6.3 and skip to D.6.5, otherwise leave this space blank and skip to D.6.3)

D.6.3 Short description of RID that is in use and date system was implemented

D.6.4 Statement of an intention to apply the assigned RID

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000323

D.6.5 Date of intended implementation of the RID

D.6.6 Authorized representative

Name:
Title:
Address:
Email:

Signature _____

D.6.7 For official use of the Registration Authority

Registration Rejected _____

Reason for rejection of the application:

Registration Granted _____ Registration Value _____

Attachment 1 — Attachment of technical details of the registered data format.
Attachment 2 — Attachment of notification of appeal procedure for rejected applications.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000324

Annex E
(informative)

The QoS Management Model for ISO/IEC 14496 Content

The Quality of Service (QoS) aspects deserve particular attention in ISO/IEC 14496: the ability of the standard to adapt to different service scenarios is affected by its ability to consistently manage QoS requirements. Current techniques on error resilience are already effective, but are not and will not be able to satisfy every possible requirement.

In general terms, the end-user acceptance of a particular service varies depending on the kind of service. As an example, person to person communication is severely affected by the audio quality, while it can tolerate variations in the video quality. However, a television broadcast with higher video and lower audio quality may be acceptable depending on the program being transmitted. The acceptability of a particular service thus depends very much on the service itself. It is not possible to define universal Quality of Service levels that may be suitable for all circumstances. Thus the most suitable solution is to let the content creator decide what QoS the end-user should obtain for every particular elementary stream: the author has the best knowledge of the service.

The QoS so defined represents the QoS that should be offered to the end-user, i.e., the QoS at the output of the receiving terminal. This may be the output of the decoder, but may also take into account the compositor and renderer if they significantly impact the QoS of the presentation as seen by the end-user, and if a capacity for processing a specific stream can be quantified. Note that the QoS information is not mandatory. In the absence of QoS requirements, a best effort approach should be pursued. This QoS concept is defined as *total QoS*.

In ISO/IEC 14496-1 the information concerning the total QoS of a particular elementary stream is carried in a QoS Descriptor as part of its elementary stream descriptor (ES_Descriptor). The receiving terminal, upon reception of the ES_Descriptor, is therefore aware of the characteristics of the elementary stream and of the total QoS to be offered to the end-user. Moreover the receiving terminal knows about its own performance capabilities. It is therefore the only possible entity able to compute the Quality of Service to be requested to the delivery layer in order to fit the user requirements. Note that this computation could also ignore/override the total QoS parameters.

The QoS that is requested to the delivery layer is named *media QoS*, since it is expressed with a semantic which is media oriented. The delivery layer will process the requests, determine whether to bundle multiple elementary streams into a single network connection (TransMux) and compute the QoS for the network connection, using the QoS parameters as defined by the network infrastructure. This QoS concept is named *network QoS*, since it is specific for a particular network technology.

The above categorization of the various QoS concepts managed in ISO/IEC 14496 may suggest that this issue is only relevant when operating in a network environment. However the concepts are of general value, and are applicable to systems operating on local files as well, when taking into account the overall capacity of the system.

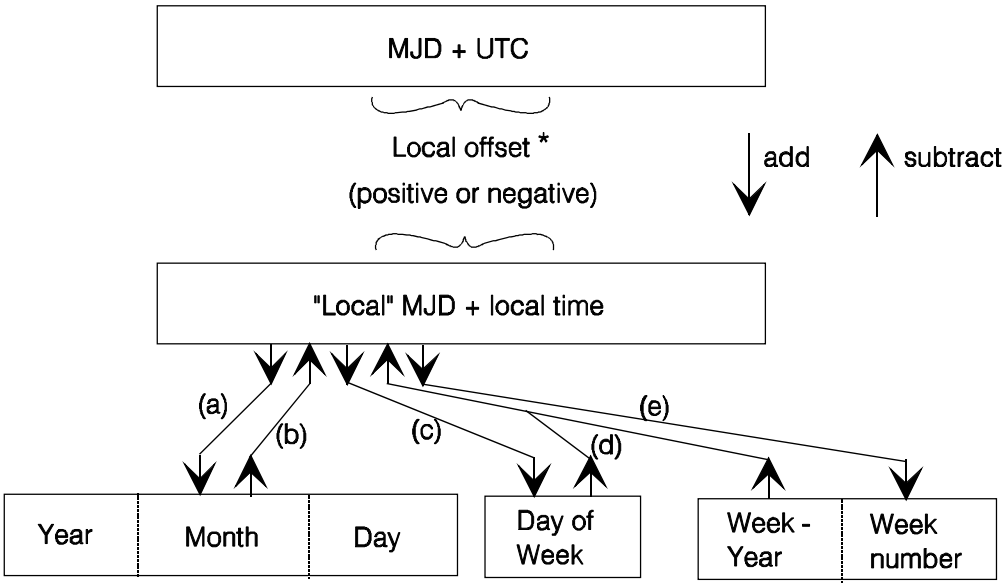
© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000325

Annex F
(informative)

Conversion Between Time and Date Conventions

The types of conversions that may be required are summarized in the diagram below.



* Offsets are positive for Longitudes East of Greenwich and negative for longitudes West of Greenwich.

Figure 38 - Conversion routes between Modified Julian Date (MJD) and Coordinated Universal Time (UTC)

The conversion between MJD + UTC and the “local” MJD + local time is simply a matter of adding or subtracting the local offset. This process may, of course, involve a “carry” or “borrow” from the UTC affecting the MJD. The other five conversion routes shown on the diagram are detailed in the formulas below.

Symbols used:

MJD:	Modified Julian Day
UTC:	Co-ordinated Universal Time
Y:	Year from 1900 (e.g. for 2003, Y = 103)
M:	Month from January (= 1) to December (= 12)
D:	Day of month from 1 to 31
WY:	"Week number" Year from 1900
MN:	Week number according to ISO 2015
WD:	Day of week from Monday (= 1) to Sunday (= 7)
K, L ,M' , W, Y':	Intermediate variables
$\times$:	Multiplication
int:	Integer part, ignoring remainder
mod 7:	Remainder (0-6) after dividing integer by 7

a) To find Y, M, D from MJD

Y' = int [(MJD - 15 078,2) / 365,25]
M' = int { [MJD - 14 956,1 - int (Y' $\times$ 365,25)] / 30,6001 }
D = MJD - 14 956 - int (Y' $\times$ 365,25) - int (M' $\times$ 30,6001)
If M' = 14 or M' = 15, then K = 1; else K = 0

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000326

$Y = Y' + K$
$M = M' - 1 - K \times 12$

b) To find MJD from Y, M, D

If M = 1 or M = 2, then L = 1; else L = 0
$MJD = 14\,956 + D + \text{int}\,[\,(Y - L) \times 365{,}25] + \text{int}\,[\,(M + 1 + L \times 12) \times 30{,}6001\,]$

c) To find WD from WJD

$WD = [\,(MJD + 2) \bmod 7\,] + 1$

d) To find MJD from WY, WN, WD

$MJD = 15\,012 + WD + 7 \times \{\,WN + \text{int}\,[\,(WY \times 1\,461 / 28) + 0{,}41]\,\}$

e) To find WY, WN from MJD

$W = \text{int}\,[\,(MJD / 7) - 2\,144{,}64\,]$
$WY = \text{int}\,[\,(W \times 28 / 1\,461) - 0{,}0079]$
$WN = W - \text{int}\,[\,(WY \times 1\,461 / 28) + 0{,}41]$

EXAMPLE —

MJD	=	45 218	W	=	4 315
Y	=	(19)82	WY	=	(19)82
M	=	9 (September)	WN	=	36
D	=	6	WD	=	1 (Monday)

NOTE — These formulas are applicable between the inclusive dates 1 900 March 1 to 2 100 February 28.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000327

Annex G
(normative)

Adaptive Arithmetic Decoder for BIFS-Anim

The follwing procedures, in C code, describe the adaptative arithmetic deoder 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.

First, the following integers are defined :

```
static long bottom=0, q1=2^14, q2=2^15, q3=3*2^14, top=2^16;
```

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);
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000328

```
}

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
        zerorun = !bit;   // if 0, start counting again at zerorun = 1
      }
   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.

```
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--] ++;

}
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000329

Annex H
(normative)

Node coding tables

Nodes: Anchor AnimationStream Appearance AudioBuffer AudioClip AudioDelay AudioFX AudioMix AudioSource AudioSwitch Background Background2D Billboard Bitmap Box Circle Collision Color ColorInterpolator CompositeTexture2D CompositeTexture3D Conditional Cone Coordinate Coordinate2D CoordinateInterpolator CoordinateInterpolator2D Curve2D Cylinder CylinderSensor DirectionalLight DiscSensor ElevationGrid Expression Extrusion Face FaceDefMesh FaceDefTables FaceDefTransform FAP FDP FIT Fog FontStyle Form Group ImageTexture IndexedFaceSet IndexedFaceSet2D IndexedLineSet IndexedLineSet2D Inline LOD Layer2D Layer3D Layout LineProperties ListeningPoint Material Material2D MovieTexture NavigationInfo Normal NormalInterpolator OrderedGroup OrientationInterpolator PixelTexture PlaneSensor PlaneSensor2D PointLight PointSet PointSet2D PositionInterpolator PositionInterpolator2D ProximitySensor2D ProximitySensor QuantizationParameter Rectangle ScalarInterpolator Script Shape Sound Sound2D Sphere SphereSensor SpotLight Switch TermCap Text TextureCoordinate TextureTransform TimeSensor TouchSensor Transform Transform2D Valuator Viewpoint VisibilitySensor Viseme WorldInfo

Node Data Types: SF2DNode SF3DNode SFAppearanceNode SFAudioNode SFBackground2DNode SFBackground3DNode SFColorNode SFCoordinate2DNode SFCoordinateNode SFExpressionNode SFFAPNode SFFDPNode SFFITNode SFFaceDefMeshNode SFFaceDefTablesNode SFFaceDefTransformNode SFFogNode SFFontStyleNode SFGeometryNode SFLinePropertiesNode SFMaterialNode SFNavigationInfoNode SFNormalNode SFStreamingNode SFTextureCoordinateNode SFTextureNode SFTextureTransformNode SFTopNode SFViewpointNode SFVisemeNode SFWorldNode

Extended Nodes: **:** AcousticMaterial AcousticScene ApplicationWindow BAP BDP Body BodyDefTable BodySegmentConnectionHint DirectiveSound Hierarchical3DMesh MaterialKey PerceptualParameters

Extended Node Data Types: **:** SF2DNode SF3DNode SFBAPNode SFBDPNode SFBodyDefTableNode SFBodySegmentConnectionHintNode SFMaterialNode SFPerceptualParameterNode SFWorldNode

H.1 Node Tables

Legend:

Node Name	Node Data Type list					nodeType/NDT		
Field name	*Field type*	*DEF id*	*IN id*	*OUT id*	*DYN id*	*[min, max]*	*Quantizer id*	*Animation method*

H.1.1 Anchor

Anchor	SFWorldNode SF3DNode SF2DNode					0000001 000001 00001		
Field name	*Field type*	*DEF id*	*IN id*	*OUT id*	*DYN id*	*[m, M]*	*Q*	*A*
addChildren	MF3DNode		000					
removeChildren	MF3DNode		001					
children	MF3DNode	00	010	00				
description	SFString	01	011	01				
parameter	MFString	10	100	10				
url	MFURL	11	101	11				

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000330

H.1.2 AnimationStream

AnimationStream	SFWorldNode SF3DNode SF2DNode SFStreamingNode					0000010 000010 00010 001		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
loop	SFBool	000	000	000				
speed	SFFloat	001	001	001		[-I, +I]	0	7
startTime	SFTime	010	010	010		[-I, +I]		
stopTime	SFTime	011	011	011		[-I, +I]		
url	MFURL	100	100	100				
duration_changed	SFTime			101				
isActive	SFBool			110				

H.1.3 Appearance

Appearance	SFWorldNode SFAppearanceNode					0000011 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
material	SFMaterialNode	00	00	00				
texture	SFTextureNode	01	01	01				
textureTransform	SFTextureTransform Node	10	10	10				

H.1.4 AudioBuffer

AudioBuffer	SFWorldNode SFAudioNode					0000100 001		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
loop	SFBool	000	000	0000				
pitch	SFFloat	001	001	0001		[0, +I]	0	7
startTime	SFTime	010	010	0010		[0, +I]	0	
stopTime	SFTime	011	011	0011		[0, +I]	0	
children	MFAudioNode	100	100	0100				
numChan	SFInt32	101	101	0101		[0, 255]	13,8	
phaseGroup	MFInt32	110	110	0110				
length	SFFloat	111				[0, +I]	0	
duration_changed	SFTime			0111				
isActive	SFBool			1000				

H.1.5 AudioClip

AudioClip	SFWorldNode SFAudioNode SFStreamingNode					0000101 010 010		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
description	SFString	000	000	000				
loop	SFBool	001	001	001				
pitch	SFFloat	010	010	010		[0, +I]	0	7
startTime	SFTime	011	011	011		[-I, +I]		
stopTime	SFTime	100	100	100		[-I, +I]		
url	MFURL	101	101	101				
duration_changed	SFTime			110				
isActive	SFBool			111				

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000331

H.1.6 AudioDelay

AudioDelay	SFWorldNode SFAudioNode					0000110 011		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren	MFAudioNode		00					
removeChildren	MFAudioNode		01					
children	MFAudioNode	00	10	0				
delay	SFTime	01	11	1		[0, +I]		
numChan	SFInt32	10				[0, 255]	13,8	
phaseGroup	MFInt32	11				[0, 255]	13,8	

H.1.7 AudioFX

AudioFX	SFWorldNode SFAudioNode					0000111 100		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren	MFAudioNode		000					
removeChildren	MFAudioNode		001					
children	MFAudioNode	000	010	00				
orch	SFString	001	011	01				
score	SFString	010	100	10				
params	MFFloat	011	101	11		[-I, +I]	0	7
numChan	SFInt32	100				[0, 255]	13,8	
phaseGroup	MFInt32	101				[0, 255]	13,8	

H.1.8 AudioMix

AudioMix	SFWorldNode SFAudioNode					0001000 101		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren	MFAudioNode		000					
removeChildren	MFAudioNode		001					
children	MFAudioNode	000	010	00				
numInputs	SFInt32	001	011	01		[1, 255]	13,8	
matrix	MFFloat	010	100	10		[0, 1]	0	7
numChan	SFInt32	011				[0, 255]	13,8	
phaseGroup	MFInt32	100				[0, 255]	13,8	

H.1.9 AudioSource

AudioSource	SFWorldNode SFAudioNode SFStreamingNode					0001001 110 011		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren	MFAudioNode		000					
removeChildren	MFAudioNode		001					
children	MFAudioNode	000	010	000				
url	MFURL	001	011	001				
pitch	SFFloat	010	100	010	0	[0, +I]	0	7
speed	SFFloat	011	101	011	1	[0, +I]	0	7
startTime	SFTime	100	110	100				
stopTime	SFTime	101	111	101				
numChan	SFInt32	110				[0, 255]	13,8	
phaseGroup	MFInt32	111				[0, 255]	13,8	

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000332

H.1.10 AudioSwitch

AudioSwitch	SFWorldNode SFAudioNode					0001010 111		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren	MFAudioNode		00					
removeChildren	MFAudioNode		01					
children	MFAudioNode	00	10	0				
whichChoice	MFInt32	01	11	1		[0, 1]	13,1	
numChan	SFInt32	10				[0, 255]	13,8	
phaseGroup	MFInt32	11				[0, 255]	13,8	

H.1.11 Background

Background	SFWorldNode SF3DNode SFBackground3DNode					0001011 000011 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_bind	SFBool		0000					
groundAngle	MFFloat	0000	0001	0000	00	[0, 1.5707963]	6	8
groundColor	MFColor	0001	0010	0001	01	[0, 1]	4	4
backUrl	MFURL	0010	0011	0010				
bottomUrl	MFURL	0011	0100	0011				
frontUrl	MFURL	0100	0101	0100				
leftUrl	MFURL	0101	0110	0101				
rightUrl	MFURL	0110	0111	0110				
topUrl	MFURL	0111	1000	0111				
skyAngle	MFFloat	1000	1001	1000	10	[0, 3.14159265]	6	8
skyColor	MFColor	1001	1010	1001	11	[0, 1]	4	4
isBound	SFBool			1010				

H.1.12 Background2D

Background2D	SFWorldNode SF2DNode SF3DNode SFBackground2DNode					0001100 00011 000100 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_bind	SFBool		00					
backColor	SFColor	0	01	00		[0, 1]	4	4
url	MFURL	1	10	01				
isBound	SFBool			10				

H.1.13 Billboard

Billboard	SFWorldNode SF3DNode					0001101 000101		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren	MF3DNode		00					
removeChildren	MF3DNode		01					
children	MF3DNode	0	10	0				
axisOfRotation	SFVec3f	1	11	1			9	9

H.1.14 Bitmap

Bitmap	SFWorldNode SFGeometryNode					0001110 00001		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
scale	SFVec2f					[-1, +I]	12	12

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000333

H.1.15 Box

Box	SFWorldNode SFGeometryNode					0001111 00010		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
size	SFVec3f					[0, +I]	11	

H.1.16 Circle

Circle	SFWorldNode SFGeometryNode					0010000 00011		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
radius	SFFloat					[0, +I]	12	7

H.1.17 Collision

Collision	SFWorldNode SF3DNode					0010001 000110		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren	MF3DNode		00					
removeChildren	MF3DNode		01					
children	MF3DNode	00	10	00				
collide	SFBool	01	11	01				
proxy	SF3DNode	10						
collideTime	SFTime			10				

H.1.18 Color

Color	SFWorldNode SFColorNode					0010010 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
color	MFColor					[0, 1]	4	4

H.1.19 ColorInterpolator

ColorInterpolator	SFWorldNode SF3DNode SF2DNode					0010011 000111 00100		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_fraction	SFFloat		00					
key	MFFloat	0	01	00		[0, 1]	8	
keyValue	MFColor	1	10	01		[0, 1]	4	
value_changed	SFColor			10				

H.1.20 CompositeTexture2D

CompositeTexture2D	SFWorldNode SFTextureNode					0010100 001		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren	MF2DNode		000					
removeChildren	MF2DNode		001					
children	MF2DNode	000	010	000				
pixelWidth	SFInt32	001	011	001		[0, 65535]	13,16	
pixelHeight	SFInt32	010	100	010		[0, 65535]	13,16	
background	SFBackground2DNode	011	101	011				
viewport	SFViewportNode	100	110	100				

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000334

H.1.21 CompositeTexture3D

CompositeTexture3D	SFWorldNode SFTextureNode					0010101 010		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren	MF3DNode		0000					
removeChildren	MF3DNode		0001					
children	MF3DNode	000	0010	000				
pixelWidth	SFInt32	001	0011	001		[0, 65535]	13,16	
pixelHeight	SFInt32	010	0100	010		[0, 65535]	13,16	
background	SFBackground3DNode	011	0101	011				
fog	SFFogNode	100	0110	100				
navigationInfo	SFNavigationInfoNode	101	0111	101				
viewpoint	SFViewpointNode	110	1000	110				

H.1.22 Conditional

Conditional	SFWorldNode SF3DNode SF2DNode					0010110 001000 00101		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
activate	SFBool		00					
reverseActivate	SFBool		01					
buffer	SFCommandBuffer		10	0				
isActive	SFBool			1				

H.1.23 Cone

Cone	SFWorldNode SFGeometryNode					0010111 00100		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
bottomRadius	SFFloat	00				[0, +I]	11	
height	SFFloat	01				[0, +I]	11	
side	SFBool	10						
bottom	SFBool	11						

H.1.24 Coordinate

Coordinate	SFWorldNode SFCoordinateNode					0011000 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
point	MFVec3f					[-I, +I]	1	1

H.1.25 Coordinate2D

Coordinate2D	SFWorldNode SFCoordinate2DNode					0011001 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
point	MFVec2f					[-I, +I]	2	2

H.1.26 CoordinateInterpolator

CoordinateInterpolator	SFWorldNode SF3DNode					0011010 001001		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_fraction	SFFloat		00					
key	MFFloat	0	01	00		[0, 1]	8	
keyValue	MFVec3f	1	10	01		[-I, +I]	1	
value_changed	MFVec3f			10				

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000335

H.1.27 CoordinateInterpolator2D

CoordinateInterpolator2D	SFWorldNode SF2DNode SF3DNode					0011011 00110 001010		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_fraction	SFFloat		00					
key	MFFloat	0	01	00		[0, 1]	8	
keyValue	MFVec2f	1	10	01		[-I, +I]	2	
value_changed	MFVec2f			10				

H.1.28 Curve2D

Curve2D	SFWorldNode SFGeometryNode					0011100 00101		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
point	SFCoordinate2DNode	00	00	00				
fineness	SFFloat	01	01	01		[0, 1]	0	7
type	MFInt32	10	10	10		[0, 3]	13,2	

H.1.29 Cylinder

Cylinder	SFWorldNode SFGeometryNode					0011101 00110		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
bottom	SFBool	000						
height	SFFloat	001				[0, +I]	11	
radius	SFFloat	010				[0, +I]	11	
side	SFBool	011						
top	SFBool	100						

H.1.30 CylinderSensor

CylinderSensor	SFWorldNode SF3DNode					0011110 001011		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
autoOffset	SFBool	000	000	0000				
diskAngle	SFFloat	001	001	0001		[0, 6.2831853]	6	
enabled	SFBool	010	010	0010				
maxAngle	SFFloat	011	011	0011		[0, 6.2831853]	6	
minAngle	SFFloat	100	100	0100		[0, 6.2831853]	6	
offset	SFFloat	101	101	0101		[0, 6.2831853]	6	
isActive	SFBool			0110				
rotation_changed	SFRotation			0111				
trackPoint_changed	SFVec3f			1000				

H.1.31 DirectionalLight

DirectionalLight	SFWorldNode SF3DNode					0011111 001100		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
ambientIntensity	SFFloat	000	000	000	00	[0, 1]	4	8
color	SFColor	001	001	001	01	[0, 1]	4	4
direction	SFVec3f	010	010	010	10		9	9
intensity	SFFloat	011	011	011	11	[0, 1]	4	8
on	SFBool	100	100	100				

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000336

H.1.32 DiscSensor

DiscSensor	SFWorldNode SF2DNode SF3DNode					0100000 00111 001101		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
autoOffset	SFBool	000	000	000				
enabled	SFBool	001	001	001				
maxAngle	SFFloat	010	010	010		[-6.2831853, 6.2831853]	6	
minAngle	SFFloat	011	011	011		[-6.2831853, 6.2831853]	6	
offset	SFFloat	100	100	100		[0, 6.2831853]	6	
isActive	SFBool			101				
rotation_changed	SFFloat			110				
trackPoint_changed	SFVec2f			111				

H.1.33 ElevationGrid

ElevationGrid	SFWorldNode SFGeometryNode					0100001 00111		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_height	MFFloat		00					
color	SFColorNode	0000	01	00				
normal	SFNormalNode	0001	10	01				
texCoord	SFTextureCoordinate Node	0010	11	10				
height	MFFloat	0011				[-I, +I]	11	7
ccw	SFBool	0100						
colorPerVertex	SFBool	0101						
creaseAngle	SFFloat	0110				[0, 6.2831853]	6	
normalPerVertex	SFBool	0111						
solid	SFBool	1000						
xDimension	SFInt32	1001				[0, +I]	11	
xSpacing	SFFloat	1010				[0, +I]	11	
zDimension	SFInt32	1011				[0, +I]	11	
zSpacing	SFFloat	1100				[0, +I]	11	

H.1.34 Expression

Expression	SFWorldNode SFExpressionNode					0100010 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
expression_select1	SFInt32	000	000	000		[0, 31]	13,5	
expression_intensity1	SFInt32	001	001	001		[0, 63]	13,6	
expression_select2	SFInt32	010	010	010		[0, 31]	13,5	
expression_intensity2	SFInt32	011	011	011		[0, 63]	13,6	
init_face	SFBool	100	100	100				
expression_def	SFBool	101	101	101				

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000337

H.1.35 Extrusion

Extrusion	SFWorldNode SFGeometryNode					0100011 01000		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_crossSection	MFVec2f		00					
set_orientation	MFRotation		01					
set_scale	MFVec2f		10					
set_spine	MFVec3f		11					
beginCap	SFBool	0000						
ccw	SFBool	0001						
convex	SFBool	0010						
creaseAngle	SFFloat	0011				[0, 6.2831853]	6	
crossSection	MFVec2f	0100				[-I, +I]	2	
endCap	SFBool	0101						
orientation	MFRotation	0110				[-I, +I]	10	
scale	MFVec2f	0111				[0, +I]	7	
solid	SFBool	1000						
spine	MFVec3f	1001				[-I, +I]	1	

H.1.36 Face

Face	SFWorldNode SF3DNode SF2DNode					0100100 001110 01000		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
fap	SFFAPNode	000	000	000				
fdp	SFFDPNode	001	001	001				
fit	SFFITNode	010	010	010				
ttsSource	SFAudioNode	011	011	011				
renderedFace	MF3DNode	100	100	100				

H.1.37 FaceDefMesh

FaceDefMesh	SFWorldNode SFFaceDefMeshNode					0100101 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
faceSceneGraphNode	SF3DNode	00						
intervalBorders	MFInt32	01					0	
coordIndex	MFInt32	10					0	
displacements	MFVec3f	11					0	

H.1.38 FaceDefTables

FaceDefTables	SFWorldNode SFFaceDefTablesNode					0100110 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
fapID	SFInt32	00				[1, 68]	13,7	
highLevelSelect	SFInt32	01				[1, 64]	13,6	
faceDefMesh	MFFaceDefMeshNode	10	0	0				
faceDefTransform	MFFaceDefTransformNode	11	1	1				

H.1.39 FaceDefTransform

FaceDefTransform	SFWorldNode SFFaceDefTransformNode					0100111 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
faceSceneGraphNode	SF3DNode	000						

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000338

fieldId	SFInt32	001						
rotationDef	SFRotation	010				[-I, +I]	10	
scaleDef	SFVec3f	011					7	
translationDef	SFVec3f	100					1	

H.1.40 FAP

FAP	SFWorldNode SFFAPNode					0101000 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
viseme	SFVisemeNode	0000000	0000000	0000000				
expression	SFExpressionNode	0000001	0000001	0000001				
open_jaw	SFInt32	0000010	0000010	0000010		[0, +I]	0	
lower_t_midlip	SFInt32	0000011	0000011	0000011		[-I, +I]	0	
raise_b_midlip	SFInt32	0000100	0000100	0000100		[-I, +I]	0	
stretch_l_corner	SFInt32	0000101	0000101	0000101		[-I, +I]	0	
stretch_r_corner	SFInt32	0000110	0000110	0000110		[-I, +I]	0	
lower_t_lip_lm	SFInt32	0000111	0000111	0000111		[-I, +I]	0	
lower_t_lip_rm	SFInt32	0001000	0001000	0001000		[-I, +I]	0	
lower_b_lip_lm	SFInt32	0001001	0001001	0001001		[-I, +I]	0	
lower_b_lip_rm	SFInt32	0001010	0001010	0001010		[-I, +I]	0	
raise_l_cornerlip	SFInt32	0001011	0001011	0001011		[-I, +I]	0	
raise_r_cornerlip	SFInt32	0001100	0001100	0001100		[-I, +I]	0	
thrust_jaw	SFInt32	0001101	0001101	0001101		[0, +I]	0	
shift_jaw	SFInt32	0001110	0001110	0001110		[-I, +I]	0	
push_b_lip	SFInt32	0001111	0001111	0001111		[-I, +I]	0	
push_t_lip	SFInt32	0010000	0010000	0010000		[-I, +I]	0	
depress_chin	SFInt32	0010001	0010001	0010001		[-I, +I]	0	
close_t_l_eyelid	SFInt32	0010010	0010010	0010010		[-I, +I]	0	
close_t_r_eyelid	SFInt32	0010011	0010011	0010011		[-I, +I]	0	
close_b_l_eyelid	SFInt32	0010100	0010100	0010100		[-I, +I]	0	
close_b_r_eyelid	SFInt32	0010101	0010101	0010101		[-I, +I]	0	
yaw_l_eyeball	SFInt32	0010110	0010110	0010110		[-I, +I]	0	
yaw_r_eyeball	SFInt32	0010111	0010111	0010111		[-I, +I]	0	
pitch_l_eyeball	SFInt32	0011000	0011000	0011000		[-I, +I]	0	
pitch_r_eyeball	SFInt32	0011001	0011001	0011001		[-I, +I]	0	
thrust_l_eyeball	SFInt32	0011010	0011010	0011010		[-I, +I]	0	
thrust_r_eyeball	SFInt32	0011011	0011011	0011011		[-I, +I]	0	
dilate_l_pupil	SFInt32	0011100	0011100	0011100		[-I, +I]	0	
dilate_r_pupil	SFInt32	0011101	0011101	0011101		[-I, +I]	0	
raise_l_i_eyebrow	SFInt32	0011110	0011110	0011110		[-I, +I]	0	
raise_r_i_eyebrow	SFInt32	0011111	0011111	0011111		[-I, +I]	0	
raise_l_m_eyebrow	SFInt32	0100000	0100000	0100000		[-I, +I]	0	
raise_r_m_eyebrow	SFInt32	0100001	0100001	0100001		[-I, +I]	0	
raise_l_o_eyebrow	SFInt32	0100010	0100010	0100010		[-I, +I]	0	
raise_r_o_eyebrow	SFInt32	0100011	0100011	0100011		[-I, +I]	0	
squeeze_l_eyebrow	SFInt32	0100100	0100100	0100100		[-I, +I]	0	
squeeze_r_eyebrow	SFInt32	0100101	0100101	0100101		[-I, +I]	0	
puff_l_cheek	SFInt32	0100110	0100110	0100110		[-I, +I]	0	
puff_r_cheek	SFInt32	0100111	0100111	0100111		[-I, +I]	0	
lift_l_cheek	SFInt32	0101000	0101000	0101000		[0, +I]	0	
lift_r_cheek	SFInt32	0101001	0101001	0101001		[0, +I]	0	
shift_tongue_tip	SFInt32	0101010	0101010	0101010		[-I, +I]	0	
raise_tongue_tip	SFInt32	0101011	0101011	0101011		[-I, +I]	0	
thrust_tongue_tip	SFInt32	0101100	0101100	0101100		[-I, +I]	0	
raise_tongue	SFInt32	0101101	0101101	0101101		[-I, +I]	0	
tongue_roll	SFInt32	0101110	0101110	0101110		[0, +I]	0	
head_pitch	SFInt32	0101111	0101111	0101111		[-I, +I]	0	
head_yaw	SFInt32	0110000	0110000	0110000		[-I, +I]	0	
head_roll	SFInt32	0110001	0110001	0110001		[-I, +I]	0	
lower_t_midlip_o	SFInt32	0110010	0110010	0110010		[-I, +I]	0	

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000339

raise_b_midlip_o	SFInt32	0110011	0110011	0110011		[-I, +I]	0	
stretch_l_cornerlip	SFInt32	0110100	0110100	0110100		[-I, +I]	0	
stretch_r_cornerlip	SFInt32	0110101	0110101	0110101		[-I, +I]	0	
lower_t_lip_lm_o	SFInt32	0110110	0110110	0110110		[-I, +I]	0	
lower_t_lip_rm_o	SFInt32	0110111	0110111	0110111		[-I, +I]	0	
raise_b_lip_lm_o	SFInt32	0111000	0111000	0111000		[-I, +I]	0	
raise_b_lip_rm_o	SFInt32	0111001	0111001	0111001		[-I, +I]	0	
raise_l_cornerlip_o	SFInt32	0111010	0111010	0111010		[-I, +I]	0	
raise_r_cornerlip_o	SFInt32	0111011	0111011	0111011		[-I, +I]	0	
stretch_l_nose	SFInt32	0111100	0111100	0111100		[-I, +I]	0	
stretch_r_nose	SFInt32	0111101	0111101	0111101		[-I, +I]	0	
raise_nose	SFInt32	0111110	0111110	0111110		[-I, +I]	0	
bend_nose	SFInt32	0111111	0111111	0111111		[-I, +I]	0	
raise_l_ear	SFInt32	1000000	1000000	1000000		[-I, +I]	0	
raise_r_ear	SFInt32	1000001	1000001	1000001		[-I, +I]	0	
pull_l_ear	SFInt32	1000010	1000010	1000010		[-I, +I]	0	
pull_r_ear	SFInt32	1000011	1000011	1000011		[-I, +I]	0	

H.1.41 FDP

FDP	SFWorldNode SFFDPNode					0101001 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
featurePointsCoord	SFCoordinateNode	000	00	00				
textureCoord	SFTextureCoordinate Node	001	01	01				
faceDefTables	MFFaceDefTablesNo de	010	10	10				
faceSceneGraph	MF3DNode	011	11	11				
useOrthoTexture	SFBool	100						

H.1.42 FIT

FIT	SFWorldNode SFFITNode					0101010 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
FAPs	MFInt32	0000	0000	0000		[-1, 68]	13,7	
Graph	MFInt32	0001	0001	0001		[0, 68]	13,7	
numeratorExp	MFInt32	0010	0010	0010		[0, 15]	13,4	
denominatorExp	MFInt32	0011	0011	0011		[0, 15]	13,4	
numeratorImpulse	MFInt32	0100	0100	0100		[0, 1023]	13,10	
numeratorTerms	MFInt32	0101	0101	0101		[0, 10]	13,4	
denominatorTerms	MFInt32	0110	0110	0110		[0, 10]	13,4	
numeratorCoefs	MFFloat	0111	0111	0111		[-I, +I]		
denominatorCoefs	MFFloat	1000	1000	1000		[-I, +I]		

H.1.43 Fog

Fog	SFWorldNode SF3DNode SFFogNode					0101011 001111 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
color	SFColor	00	00	00	0	[0, 1]	4	4
fogType	SFString	01	01	01				
visibilityRange	SFFloat	10	10	10	1	[0, +I]	11	7
set_bind	SFBool		11					
isBound	SFBool			11				

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000340

H.1.44 FontStyle

FontStyle	SFWorldNode SFFontStyleNode					0101100 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
family	MFString	0000						
horizontal	SFBool	0001						
justify	MFString	0010						
language	SFString	0011						
leftToRight	SFBool	0100						
size	SFFloat	0101				[0, +I]	11	
spacing	SFFloat	0110				[0, +I]	11	
style	SFString	0111						
topToBottom	SFBool	1000						

H.1.45 Form

Form	SFWorldNode SF2DNode SF3DNode					0101101 01001 010000		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren	MF2DNode		000					
removeChildren	MF2DNode		001					
children	MF2DNode	000	010	000				
size	SFVec2f	001	011	001		[0, +I]	12	12
groups	MFInt32	010	100	010		[-1, 1022]	13,10	
constraints	MFString	011	101	011				
groupsIndex	MFInt32	100	110	100		[-1, 1022]	13,10	

H.1.46 Group

Group	SFWorldNode SFTopNode SF3DNode SF2DNode					0101110 001 010001 01010		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren	MF3DNode		00					
removeChildren	MF3DNode		01					
children	MF3DNode		10					

H.1.47 ImageTexture

ImageTexture	SFWorldNode SFTextureNode					0101111 011		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
url	MFURL	00						
repeatS	SFBool	01						
repeatT	SFBool	10						

H.1.48 IndexedFaceSet

IndexedFaceSet	SFWorldNode SFGeometryNode					0110000 01001		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_colorIndex	MFInt32		000					
set_coordIndex	MFInt32		001					
set_normalIndex	MFInt32		010					
set_texCoordIndex	MFInt32		011					
color	SFColorNode	0000	100	00				
coord	SFCoordinateNode	0001	101	01				
normal	SFNormalNode	0010	110	10				
texCoord	SFTextureCoordinate Node	0011	111	11				

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000341

ccw	SFBool	0100						
colorIndex	MFInt32	0101				[-1, +I]	14	
colorPerVertex	SFBool	0110						
convex	SFBool	0111						
coordIndex	MFInt32	1000				[-1, +I]	14	
creaseAngle	SFFloat	1001				[0, 6.2831853]	6	
normalIndex	MFInt32	1010				[-1, +I]	14	
normalPerVertex	SFBool	1011						
solid	SFBool	1100						
texCoordIndex	MFInt32	1101				[-1, +I]	14	

H.1.49 IndexedFaceSet2D

IndexedFaceSet2D	SFWorldNode SFGeometryNode					0110001 01010		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_colorIndex	MFInt32		000					
set_coordIndex	MFInt32		001					
set_texCoordIndex	MFInt32		010					
color	SFColorNode	000	011	00				
coord	SFCoordinate2DNode	001	100	01				
texCoord	SFTextureCoordinate Node	010	101	10				
colorIndex	MFInt32	011				[0, +I]	14	
colorPerVertex	SFBool	100						
convex	SFBool	101						
coordIndex	MFInt32	110				[0, +I]	14	
texCoordIndex	MFInt32	111				[0, +I]	14	

H.1.50 IndexedLineSet

IndexedLineSet	SFWorldNode SFGeometryNode					0110010 01011		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_colorIndex	MFInt32		00					
set_coordIndex	MFInt32		01					
color	SFColorNode	000	10	0				
coord	SFCoordinateNode	001	11	1				
colorIndex	MFInt32	010				[-1, +I]	14	
colorPerVertex	SFBool	011						
coordIndex	MFInt32	100				[-1, +I]	14	

H.1.51 IndexedLineSet2D

IndexedLineSet2D	SFWorldNode SFGeometryNode					0110011 01100		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_colorIndex	MFInt32		00					
set_coordIndex	MFInt32		01					
color	SFColorNode	000	10	0				
coord	SFCoordinate2DNode	001	11	1				
colorIndex	MFInt32	010				[0, +I]	14	
colorPerVertex	SFBool	011						
coordIndex	MFInt32	100				[0, +I]	14	

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000342

H.1.52 Inline

Inline	SFWorldNode SF3DNode SFStreamingNode SF2DNode					0110100 010010 100 01011		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
url	MFURL							

H.1.53 LOD

LOD	SFWorldNode SF3DNode SF2DNode					0110101 010011 01100		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
level	MF3DNode	00						
center	SFVec3f	01				[-I, +I]	1	
range	MFFloat	10				[0, +I]	11	

H.1.54 Layer2D

Layer2D	SFWorldNode SFTopNode SF2DNode SF3DNode					0110110 010 01101 010100		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren	MF2DNode		000					
removeChildren	MF2DNode		001					
children	MF2DNode	00	010	00				
size	SFVec2f	01	011	01		[-I, +I]	12	12
background	SFBackground2DNode	10	100	10				
viewport	SFViewportNode	11	101	11				

H.1.55 Layer3D

Layer3D	SFWorldNode SFTopNode SF2DNode SF3DNode					0110111 011 01110 010101		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren	MF3DNode		000					
removeChildren	MF3DNode		001					
children	MF3DNode	000	010	000				
size	SFVec2f	001	011	001		[-I, +I]	12	12
background	SFBackground3DNode	010	100	010				
fog	SFFogNode	011	101	011				
navigationInfo	SFNavigationInfoNode	100	110	100				
viewpoint	SFViewpointNode	101	111	101				

H.1.56 Layout

Layout	SFWorldNode SF2DNode SF3DNode					0111000 01111 010100		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren	MF2DNode		0000					
removeChildren	MF2DNode		0001					
children	MF2DNode	0000	0010	0000				
wrap	SFBool	0001	0011	0001				
size	SFVec2f	0010	0100	0010	00	[0, +I]	12	12
horizontal	SFBool	0011	0101	0011				
justify	MFString	0100	0110	0100				
leftToRight	SFBool	0101	0111	0101				

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000343

topToBottom	SFBool	0110	1000	0110				
spacing	SFFloat	0111	1001	0111	01	[0, +I]	0	7
smoothScroll	SFBool	1000	1010	1000				
loop	SFBool	1001	1011	1001				
scrollVertical	SFBool	1010	1100	1010				
scrollRate	SFFloat	1011	1101	1011	10	[-I, +I]	0	7

H.1.57 LineProperties

LineProperties	SFWorldNode SFLinePropertiesNode					0111001 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
lineColor	SFColor	00	00	00	0	[0, 1]	4	4
lineStyle	SFInt32	01	01	01		[0, 5]	13,3	
width	SFFloat	10	10	10	1	[0, +I]	12	7

H.1.58 ListeningPoint

ListeningPoint	SFWorldNode SF3DNode					0111010 010111		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_bind	SFBool		00					
jump	SFBool	00	01	000				
orientation	SFRotation	01	10	001	0		10	10
position	SFVec3f	10	11	010	1	[-I, +I]	1	1
description	SFString	11						
bindTime	SFTime			011				
isBound	SFBool			100				

H.1.59 Material

Material	SFWorldNode SFMaterialNode					0111011 01		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
ambientIntensity	SFFloat	000	000	000	000	[0, 1]	4	8
diffuseColor	SFColor	001	001	001	001	[0, 1]	4	4
emissiveColor	SFColor	010	010	010	010	[0, 1]	4	4
shininess	SFFloat	011	011	011	011	[0, 1]	4	8
specularColor	SFColor	100	100	100	100	[0, 1]	4	4
transparency	SFFloat	101	101	101	101	[0, 1]	4	8

H.1.60 Material2D

Material2D	SFWorldNode SFMaterialNode					0111100 10		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
emissiveColor	SFColor	00	00	00	0	[0, 1]	4	4
filled	SFBool	01	01	01				
lineProps	SFLinePropertiesNode	10	10	10				
transparency	SFFloat	11	11	11	1	[0, 1]	4	8

H.1.61 MovieTexture

MovieTexture	SFWorldNode SFTextureNode SFStreamingNode					0111101 100 101		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
loop	SFBool	000	000	000				
speed	SFFloat	001	001	001		[-I, +I]	0	7

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000344

startTime	SFTime	010	010	010		[-I, +I]		
stopTime	SFTime	011	011	011		[-I, +I]		
url	MFURL	100	100	100				
repeatS	SFBool	101						
repeatT	SFBool	110						
duration_changed	SFTime			101				
isActive	SFBool			110				

H.1.62 NavigationInfo

NavigationInfo	SFWorldNode SF3DNode SFNavigationInfoNode					0111110 011000 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_bind	SFBool		000					
avatarSize	MFFloat	000	001	000		[0, +I]	11	
headlight	SFBool	001	010	001				
speed	SFFloat	010	011	010		[0, +I]	0	
type	MFString	011	100	011				
visibilityLimit	SFFloat	100	101	100		[0, +I]	11	7
isBound	SFBool			101				

H.1.63 Normal

Normal	SFWorldNode SFNormalNode					0111111 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
vector	MFVec3f						9	9

H.1.64 NormalInterpolator

NormalInterpolator	SFWorldNode SF3DNode					1000000 011001		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_fraction	SFFloat		00					
key	MFFloat	0	01	00		[0, 1]	8	
keyValue	MFVec3f	1	10	01		[-I, +I]	9	
value_changed	MFVec3f			10				

H.1.65 OrderedGroup

OrderedGroup	SFWorldNode SF3DNode SF2DNode SFTopNode					1000001 011010 10000 100		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren	MF3DNode		00					
removeChildren	MF3DNode		01					
children	MF3DNode	0	10	0				
order	MFFloat	1	11	1		[0, +I]	3	

H.1.66 OrientationInterpolator

OrientationInterpolator	SFWorldNode SF3DNode					1000010 011011		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_fraction	SFFloat		00					
key	MFFloat	0	01	00		[0, 1]	8	
keyValue	MFRotation	1	10	01		[-I, +I]	10	
value_changed	SFRotation			10				

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000345

H.1.67 PixelTexture

PixelTexture	SFWorldNode SFTextureNode					1000011 101		
Field name	*Field type*	*DEF id*	*IN id*	*OUT id*	*DYN id*	*[m, M]*	*Q*	*A*
image	SFImage	00					0	
repeatS	SFBool	01						
repeatT	SFBool	10						

H.1.68 PlaneSensor

PlaneSensor	SFWorldNode SF3DNode					1000100 011100		
Field name	*Field type*	*DEF id*	*IN id*	*OUT id*	*DYN id*	*[m, M]*	*Q*	*A*
autoOffset	SFBool	000	000	000				
enabled	SFBool	001	001	001				
maxPosition	SFVec2f	010	010	010		[-I, +I]	2	
minPosition	SFVec2f	011	011	011		[-I, +I]	2	
offset	SFVec3f	100	100	100		[-I, +I]	1	
isActive	SFBool			101				
trackPoint_changed	SFVec3f			110				
translation_changed	SFVec3f			111				

H.1.69 PlaneSensor2D

PlaneSensor2D	SFWorldNode SF2DNode SF3DNode					1000101 10001 011101		
Field name	*Field type*	*DEF id*	*IN id*	*OUT id*	*DYN id*	*[m, M]*	*Q*	*A*
autoOffset	SFBool	000	000	000				
enabled	SFBool	001	001	001				
maxPosition	SFVec2f	010	010	010		[-I, +I]	2	
minPosition	SFVec2f	011	011	011		[-I, +I]	2	
offset	SFVec2f	100	100	100		[-I, +I]	12	
isActive	SFBool			101				
trackPoint_changed	SFVec2f			110				
translation_changed	SFVec2f			111				

H.1.70 PointLight

PointLight	SFWorldNode SF3DNode					1000110 011110		
Field name	*Field type*	*DEF id*	*IN id*	*OUT id*	*DYN id*	*[m, M]*	*Q*	*A*
ambientIntensity	SFFloat	000	000	000	000	[0, 1]	4	8
attenuation	SFVec3f	001	001	001	001	[0, +I]	11	1
color	SFColor	010	010	010	010	[0, 1]	4	4
intensity	SFFloat	011	011	011	011	[0, 1]	4	8
location	SFVec3f	100	100	100	100	[-I, +I]	1	1
on	SFBool	101	101	101				
radius	SFFloat	110	110	110	101	[0, +I]	11	7

H.1.71 PointSet

PointSet	SFWorldNode SFGeometryNode					1000111 01101		
Field name	*Field type*	*DEF id*	*IN id*	*OUT id*	*DYN id*	*[m, M]*	*Q*	*A*
color	SFColorNode	0	0	0				
coord	SFCoordinateNode	1	1	1				

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000346

H.1.72 PointSet2D

PointSet2D	SFWorldNode SFGeometryNode					1001000 01110		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
color	SFColorNode	0	0	0				
coord	SFCoordinate2DNode	1	1	1				

H.1.73 PositionInterpolator

PositionInterpolator	SFWorldNode SF3DNode					1001001 011111		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_fraction	SFFloat		00					
key	MFFloat	0	01	00		[0, 1]	8	
keyValue	MFVec3f	1	10	01		[-I, +I]	1	
value_changed	SFVec3f			10				

H.1.74 PositionInterpolator2D

PositionInterpolator 2D	SFWorldNode SF2DNode SF3DNode					1001010 10010 100000		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_fraction	SFFloat		00					
key	MFFloat	0	01	00		[0, 1]	8	
keyValue	MFVec2f	1	10	01		[-I, +I]	2	
value_changed	SFVec2f			10				

H.1.75 ProximitySensor2D

ProximitySensor2D	SFWorldNode SF2DNode SF3DNode					1001011 10011 100001		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
center	SFVec2f	00	00	000		[-1, +I]	2	
size	SFVec2f	01	01	001		[0, +I]	12	
enabled	SFBool	10	10	010				
isActive	SFBool			011				
position_changed	SFVec2f			100				
orientation_changed	SFFloat			101				
enterTime	SFTime			110				
exitTime	SFTime			111				

H.1.76 ProximitySensor

ProximitySensor	SFWorldNode SF3DNode					1001100 100010		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
center	SFVec3f	00	00	000		[-I, +I]	1	
size	SFVec3f	01	01	001		[0, +I]	11	
enabled	SFBool	10	10	010				
isActive	SFBool			011				
position_changed	SFVec3f			100				
orientation_changed	SFRotation			101				
enterTime	SFTime			110				
exitTime	SFTime			111				

H.1.77 QuantizationParameter

QuantizationParameter	SFWorldNode SF2DNode SF3DNode					1001101 10100 100011		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
isLocal	SFBool	000000						
position3DQuant	SFBool	000001						

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000347

position3DMin	SFVec3f	000010				[-I, +I]	0	
position3DMax	SFVec3f	000011				[-I, +I]	0	
position3DNbBits	SFInt32	000100				[0, 31]	13,5	
position2DQuant	SFBool	000101						
position2DMin	SFVec2f	000110				[-I, +I]	0	
position2DMax	SFVec2f	000111				[-I, +I]	0	
position2DNbBits	SFInt32	001000				[0, 31]	13,5	
drawOrderQuant	SFBool	001001						
drawOrderMin	SFFloat	001010				[-I, +I]	0	
drawOrderMax	SFFloat	001011				[-I, +I]	0	
drawOrderNbBits	SFInt32	001100				[0, 31]	13,5	
colorQuant	SFBool	001101						
colorMin	SFFloat	001110				[0, 1]	0	
colorMax	SFFloat	001111				[0, 1]	0	
colorNbBits	SFInt32	010000				[0, 31]	13,5	
textureCoordinateQuant	SFBool	010001						
textureCoordinateMin	SFFloat	010010				[0, 1]	0	
textureCoordinateMax	SFFloat	010011				[0, 1]	0	
textureCoordinateNbBits	SFInt32	010100				[0, 31]	13,5	
angleQuant	SFBool	010101						
angleMin	SFFloat	010110				[0, 6.2831853]	0	
angleMax	SFFloat	010111				[0, 6.2831853]	0	
angleNbBits	SFInt32	011000				[0, 31]	13,5	
scaleQuant	SFBool	011001						
scaleMin	SFFloat	011010				[0, +I]	0	
scaleMax	SFFloat	011011				[0, +I]	0	
scaleNbBits	SFInt32	011100				[0, 31]	13,5	
keyQuant	SFBool	011101						
keyMin	SFFloat	011110				[-I, +I]	0	
keyMax	SFFloat	011111				[-I, +I]	0	
keyNbBits	SFInt32	100000				[0, 31]	13,5	
normalQuant	SFBool	100001						
normalNbBits	SFInt32	100010				[0, 31]	13,5	
sizeQuant	SFBool	100011						
sizeMin	SFFloat	100100				[-I, +I]	0	
sizeMax	SFFloat	100101				[-I, +I]	0	
sizeNbBits	SFInt32	100110				[0, 31]	13,5	
useEfficientCoding	SFBool	100111						

H.1.78 Rectangle

Rectangle	SFWorldNode SFGeometryNode					1001110 01111		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
size	SFVec2f					[0, +I]	12	2

H.1.79 ScalarInterpolator

ScalarInterpolator	SFWorldNode SF3DNode SF2DNode					1001111 100100 10101		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_fraction	SFFloat		00					
key	MFFloat	0	01	00		[0, 1]	8	
keyValue	MFFloat	1	10	01		[-I, +I]	0	
value_changed	SFFloat			10				

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000348

H.1.80 Script

Script	SFWorldNode SF3DNode SF2DNode					1010000 100101 10110		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
url	MFScript	00						
directOutput	SFBool	01						
mustEvaluate	SFBool	10						

H.1.81 Shape

Shape	SFWorldNode SF3DNode SF2DNode					1010001 100110 10111		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
appearance	SFAppearanceNode	0	0	0				
geometry	SFGeometryNode	1	1	1				

H.1.82 Sound

Sound	SFWorldNode SF3DNode					1010010 100111		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
direction	SFVec3f	0000	0000	0000		[-I, +I]	9	
intensity	SFFloat	0001	0001	0001	000	[0, 1]	4	7
location	SFVec3f	0010	0010	0010	001	[-I, +I]	1	1
maxBack	SFFloat	0011	0011	0011	010	[0, +I]	11	7
maxFront	SFFloat	0100	0100	0100	011	[0, +I]	11	7
minBack	SFFloat	0101	0101	0101	100	[0, +I]	11	7
minFront	SFFloat	0110	0110	0110	101	[0, +I]	11	7
priority	SFFloat	0111	0111	0111		[0, 1]	4	
source	SFAudioNode	1000	1000	1000				
spatialize	SFBool	1001						

H.1.83 Sound2D

Sound2D	SFWorldNode SF2DNode SF3DNode					1010011 11000 101000		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
intensity	SFFloat	00	00	00	0	[0, 1]	4	7
location	SFVec2f	01	01	01	1	[-I, +I]	2	2
source	SFAudioNode	10	10	10				
spatialize	SFBool	11						

H.1.84 Sphere

Sphere	SFWorldNode SFGeometryNode					1010100 10000		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
radius	SFFloat					[0, +I]	11	

H.1.85 SphereSensor

SphereSensor	SFWorldNode SF3DNode					1010101 101001		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
autoOffset	SFBool	00	00	000				
enabled	SFBool	01	01	001				
offset	SFRotation	10	10	010		[-I, +I]	10	
isActive	SFBool			011				
rotation_changed	SFRotation			100				
trackPoint_changed	SFVec3f			101				

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000349

H.1.86 SpotLight

SpotLight	SFWorldNode SF3DNode					1010110 101010		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
ambientIntensity	SFFloat	0000	0000	0000	0000	[0, 1]	4	8
attenuation	SFVec3f	0001	0001	0001	0001	[0, +I]	11	1
beamWidth	SFFloat	0010	0010	0010	0010	[0, 1.5707963]	6	8
color	SFColor	0011	0011	0011	0011	[0, 1]	4	4
cutOffAngle	SFFloat	0100	0100	0100	0100	[0, 1.5707963]	6	8
direction	SFVec3f	0101	0101	0101	0101	[-I, +I]	9	9
intensity	SFFloat	0110	0110	0110	0110	[0, 1]	4	8
location	SFVec3f	0111	0111	0111	0111	[-I, +I]	1	1
on	SFBool	1000	1000	1000				
radius	SFFloat	1001	1001	1001	1000	[0, +I]	11	7

H.1.87 Switch

Switch	SFWorldNode SF3DNode SF2DNode					1010111 101011 11001		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
choice	MF3DNode	0	0	0				
whichChoice	SFInt32	1	1	1		[-1, 1022]	13,10	

H.1.88 TermCap

TermCap	SFWorldNode SF3DNode SF2DNode					1011000 11010 101100		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
evaluate	SFTime		0					
capability	SFInt32		1	0		[0, 127]	13,7	
value	SFInt32			1		[0, 7]	13,3	

H.1.89 Text

Text	SFWorldNode SFGeometryNode					1011001 10001		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
string	MFString	00	00	00				
length	MFFloat	01	01	01	0	[0, +I]	11	7
fontStyle	SFFontStyleNode	10	10	10				
maxExtent	SFFloat	11	11	11	1	[0, +I]	11	7

H.1.90 TextureCoordinate

TextureCoordinate	SFWorldNode SFTextureCoordinateNode					1011010 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
point	MFVec2f					[-I, +I]	5	2

H.1.91 TextureTransform

TextureTransform	SFWorldNode SFTextureTransformNode					1011011 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
center	SFVec2f	00	00	00	00	[-I, +I]	2	2
rotation	SFFloat	01	01	01	01	[0, 6.2831853]	6	6
scale	SFVec2f	10	10	10	10	[-I, +I]	7	12
translation	SFVec2f	11	11	11	11	[-I, +I]	2	2

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000350

H.1.92 TimeSensor

TimeSensor	SFWorldNode SF3DNode SF2DNode					1011100 101101 11011		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
cycleInterval	SFTime	000	000	0000		[0, +I]		
enabled	SFBool	001	001	0001				
loop	SFBool	010	010	0010				
startTime	SFTime	011	011	0011		[-I, +I]		
stopTime	SFTime	100	100	0100		[-I, +I]		
cycleTime	SFTime			0101				
fraction_changed	SFFloat			0110				
isActive	SFBool			0111				
time	SFTime			1000				

H.1.93 TouchSensor

TouchSensor	SFWorldNode SF2DNode SF3DNode					1011101 11100 101110		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
enabled	SFBool			000				
hitNormal_changed	SFVec3f			001				
hitPoint_changed	SFVec3f			010				
hitTexCoord_changed	SFVec2f			011				
isActive	SFBool			100				
isOver	SFBool			101				
touchTime	SFTime			110				

H.1.94 Transform

Transform	SFWorldNode SF3DNode					1011110 101111		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren	MF3DNode		000					
removeChildren	MF3DNode		001					
center	SFVec3f	000	010	000	000	[-I, +I]	1	1
children	MF3DNode	001	011	001				
rotation	SFRotation	010	100	010	001		10	10
scale	SFVec3f	011	101	011	010	[0, +I]	7	11
scaleOrientation	SFRotation	100	110	100	011		10	10
translation	SFVec3f	101	111	101	100	[-I, +I]	1	1

H.1.95 Transform2D

Transform2D	SFWorldNode SF2DNode SF3DNode					1011111 11101 110000		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren	MF2DNode		000					
removeChildren	MF2DNode		001					
children	MF2DNode	000	010	000				
center	SFVec2f	001	011	001	000	[-I, +I]	2	2
rotationAngle	SFFloat	010	100	010	001	[0, 6.2831853]	6	6
scale	SFVec2f	011	101	011	010	[0, +I]	7	12
scaleOrientation	SFFloat	100	110	100	011	[0, 6.2831853]	6	6
translation	SFVec2f	101	111	101	100	[-I, +I]	2	2

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000351

H.1.96 Valuator

Valuator	SFWorldNode SF3DNode SF2DNode					1100000 110001 11110		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
inSFBool	SFBool		00000					
inSFColor	SFColor		00001					
inMFColor	MFColor		00010					
inSFFloat	SFFloat		00011					
inMFFloat	MFFloat		00100					
inSFInt32	SFInt32		00101					
inMFInt32	MFInt32		00110					
inSFRotation	SFRotation		00111					
inMFRotation	MFRotation		01000					
inSFString	SFString		01001					
inMFString	MFString		01010					
inSFTime	SFTime		01011					
inSFVec2f	SFVec2f		01100					
inMFVec2f	MFVec2f		01101					
inSFVec3f	SFVec3f		01110					
inMFVec3f	MFVec3f		01111					
outSFBool	SFBool			00000				
outSFColor	SFColor			00001				
outMFColor	MFColor			00010				
outSFFloat	SFFloat			00011				
outMFFloat	MFFloat			00100				
outSFInt32	SFInt32			00101				
outMFInt32	MFInt32			00110				
outSFRotation	SFRotation			00111				
outMFRotation	MFRotation			01000				
outSFString	SFString			01001				
outMFString	MFString			01010				
outSFTime	SFTime			01011				
outSFVec2f	SFVec2f			01100				
outMFVec2f	MFVec2f			01101				
outSFVec3f	SFVec3f			01110				
outMFVec3f	MFVec3f			01111				
Factor1	SFFloat	0000	10000	10000		[-I, +I]	0	
Factor2	SFFloat	0001	10001	10001		[-I, +I]	0	
Factor3	SFFloat	0010	10010	10010		[-I, +I]	0	
Factor4	SFFloat	0011	10011	10011		[-I, +I]	0	
Offset1	SFFloat	0100	10100	10100		[-I, +I]	0	
Offset2	SFFloat	0101	10101	10101		[-I, +I]	0	
Offset3	SFFloat	0110	10110	10110		[-I, +I]	0	
Offset4	SFFloat	0111	10111	10111		[-I, +I]	0	
Sum	SFBool	1000	11000	11000				

H.1.97 Viewpoint

Viewpoint	SFWorldNode SF3DNode SFViewpointNode					1100001 110010 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_bind	SFBool		000					
fieldOfView	SFFloat	000	001	000	00	[0, 3.1415927]	6	8
jump	SFBool	001	010	001				
orientation	SFRotation	010	011	010	01		10	10
position	SFVec3f	011	100	011	10	[-I, +I]	1	1
description	SFString	100						
bindTime	SFTime			100				
isBound	SFBool			101				

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000352

H.1.98 VisibilitySensor

VisibilitySensor	SFWorldNode SF3DNode					1100010 110011		
Field name	*Field type*	*DEF id*	*IN id*	*OUT id*	*DYN id*	*[m, M]*	*Q*	*A*
center	SFVec3f	00	00	000	0		1	1
enabled	SFBool	01	01	001				
size	SFVec3f	10	10	010	1		11	11
enterTime	SFTime			011				
exitTime	SFTime			100				
isActive	SFBool			101				

H.1.99 Viseme

Viseme	SFWorldNode SFVisemeNode					1100011 1		
Field name	*Field type*	*DEF id*	*IN id*	*OUT id*	*DYN id*	*[m, M]*	*Q*	*A*
viseme_select1	SFInt32	00	00	00		[0, 31]	13,5	
viseme_select2	SFInt32	01	01	01		[0, 31]	13,5	
viseme_blend	SFInt32	10	10	10		[0, 63]	13,6	
viseme_def	SFBool	11	11	11				

H.1.100 WorldInfo

WorldInfo	SFWorldNode SF2DNode SF3DNode					1100100 11111 110100		
Field name	*Field type*	*DEF id*	*IN id*	*OUT id*	*DYN id*	*[m, M]*	*Q*	*A*
info	MFString	0						
title	SFString	1						

H.2 Node Definition Type Tables

Legend:

Node Definition Type		Number of nodes			
Node name	nodeType	DEF	IN	OUT	DYN

H.2.1 SF2DNode

SF2DNode		31 Nodes			
reserved	00000				
Anchor	00001	2 DEF bits	3 IN bits	2 OUT bits	0 DYN bits
AnimationStream	00010	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Background2D	00011	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
ColorInterpolator	00100	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
Conditional	00101	0 DEF bits	2 IN bits	1 OUT bits	0 DYN bits
CoordinateInterpolator2D	00110	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
DiscSensor	00111	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Face	01000	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Form	01001	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Group	01010	0 DEF bits	2 IN bits	0 OUT bits	0 DYN bits
Inline	01011	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
LOD	01100	2 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Layer2D	01101	2 DEF bits	3 IN bits	2 OUT bits	0 DYN bits
Layer3D	01110	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Layout	01111	4 DEF bits	4 IN bits	4 OUT bits	2 DYN bits
OrderedGroup	10000	1 DEF bits	2 IN bits	1 OUT bits	0 DYN bits
PlaneSensor2D	10001	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
PositionInterpolator2D	10010	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
ProximitySensor2D	10011	2 DEF bits	2 IN bits	3 OUT bits	0 DYN bits
QuantizationParameter	10100	6 DEF bits	0 IN bits	0 OUT bits	0 DYN bits

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000353

ScalarInterpolator	10101	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
Script	10110	2 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Shape	10111	1 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
Sound2D	11000	2 DEF bits	2 IN bits	2 OUT bits	1 DYN bits
Switch	11001	1 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
TermCap	11010	0 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
TimeSensor	11011	3 DEF bits	3 IN bits	4 OUT bits	0 DYN bits
TouchSensor	11100	0 DEF bits	0 IN bits	3 OUT bits	0 DYN bits
Transform2D	11101	3 DEF bits	3 IN bits	3 OUT bits	3 DYN bits
Valuator	11110	4 DEF bits	5 IN bits	5 OUT bits	0 DYN bits
WorldInfo	11111	1 DEF bits	0 IN bits	0 OUT bits	0 DYN bits

H.2.2 SF3DNode

SF3DNode		**52 Nodes**			
reserved	000000				
Anchor	000001	2 DEF bits	3 IN bits	2 OUT bits	0 DYN bits
AnimationStream	000010	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Background	000011	4 DEF bits	4 IN bits	4 OUT bits	2 DYN bits
Background2D	000100	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
Billboard	000101	1 DEF bits	2 IN bits	1 OUT bits	0 DYN bits
Collision	000110	2 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
ColorInterpolator	000111	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
Conditional	001000	0 DEF bits	2 IN bits	1 OUT bits	0 DYN bits
CoordinateInterpolator	001001	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
CoordinateInterpolator2D	001010	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
CylinderSensor	001011	3 DEF bits	3 IN bits	4 OUT bits	0 DYN bits
DirectionalLight	001100	3 DEF bits	3 IN bits	3 OUT bits	2 DYN bits
DiscSensor	001101	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Face	001110	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Fog	001111	2 DEF bits	2 IN bits	2 OUT bits	1 DYN bits
Form	010000	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Group	010001	0 DEF bits	2 IN bits	0 OUT bits	0 DYN bits
Inline	010010	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
LOD	010011	2 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Layer2D	010100	2 DEF bits	3 IN bits	2 OUT bits	0 DYN bits
Layer3D	010101	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Layout	010110	4 DEF bits	4 IN bits	4 OUT bits	2 DYN bits
ListeningPoint	010111	2 DEF bits	2 IN bits	3 OUT bits	1 DYN bits
NavigationInfo	011000	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
NormalInterpolator	011001	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
OrderedGroup	011010	1 DEF bits	2 IN bits	1 OUT bits	0 DYN bits
OrientationInterpolator	011011	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
PlaneSensor	011100	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
PlaneSensor2D	011101	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
PointLight	011110	3 DEF bits	3 IN bits	3 OUT bits	3 DYN bits
PositionInterpolator	011111	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
PositionInterpolator2D	100000	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
ProximitySensor2D	100001	2 DEF bits	2 IN bits	3 OUT bits	0 DYN bits
ProximitySensor	100010	2 DEF bits	2 IN bits	3 OUT bits	0 DYN bits
QuantizationParameter	100011	6 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
ScalarInterpolator	100100	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
Script	100101	2 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Shape	100110	1 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
Sound	100111	4 DEF bits	4 IN bits	4 OUT bits	3 DYN bits
Sound2D	101000	2 DEF bits	2 IN bits	2 OUT bits	1 DYN bits
SphereSensor	101001	2 DEF bits	2 IN bits	3 OUT bits	0 DYN bits
SpotLight	101010	4 DEF bits	4 IN bits	4 OUT bits	4 DYN bits
Switch	101011	1 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
TermCap	101100	0 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
TimeSensor	101101	3 DEF bits	3 IN bits	4 OUT bits	0 DYN bits
TouchSensor	101110	0 DEF bits	0 IN bits	3 OUT bits	0 DYN bits

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000354

Transform	101111	3 DEF bits	3 IN bits	3 OUT bits	3 DYN bits
Transform2D	110000	3 DEF bits	3 IN bits	3 OUT bits	3 DYN bits
Valuator	110001	4 DEF bits	5 IN bits	5 OUT bits	0 DYN bits
Viewpoint	110010	3 DEF bits	3 IN bits	3 OUT bits	2 DYN bits
VisibilitySensor	110011	2 DEF bits	2 IN bits	3 OUT bits	1 DYN bits
WorldInfo	110100	1 DEF bits	0 IN bits	0 OUT bits	0 DYN bits

H.2.3 SFAppearanceNode

SFAppearanceNode	1 Node				
reserved	0				
Appearance	1	2 DEF bits	2 IN bits	2 OUT bits	0 DYN bits

H.2.4 SFAudioNode

SFAudioNode	7 Nodes				
reserved	000				
AudioBuffer	001	3 DEF bits	3 IN bits	4 OUT bits	0 DYN bits
AudioClip	010	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
AudioDelay	011	2 DEF bits	2 IN bits	1 OUT bits	0 DYN bits
AudioFX	100	3 DEF bits	3 IN bits	2 OUT bits	0 DYN bits
AudioMix	101	3 DEF bits	3 IN bits	2 OUT bits	0 DYN bits
AudioSource	110	3 DEF bits	3 IN bits	3 OUT bits	1 DYN bits
AudioSwitch	111	2 DEF bits	2 IN bits	1 OUT bits	0 DYN bits

H.2.5 SFBackground2DNode

SFBackground2DNode	1 Node				
reserved	0				
Background2D	1	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits

H.2.6 SFBackground3DNode

SFBackground3DNode	1 Node				
reserved	0				
Background	1	4 DEF bits	4 IN bits	4 OUT bits	2 DYN bits

H.2.7 SFColorNode

SFColorNode	1 Node				
reserved	0				
Color	1	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits

H.2.8 SFCoordinate2DNode

SFCoordinate2DNode	1 Node				
reserved	0				
Coordinate2D	1	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits

H.2.9 SFCoordinateNode

SFCoordinateNode	1 Node				
reserved	0				
Coordinate	1	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits

H.2.10 SFExpressionNode

SFExpressionNode	1 Node				
reserved	0				
Expression	1	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000355

H.2.11 SFFAPNode

SFFAPNode		1 Node			
reserved	0				
FAP	1	7 DEF bits	7 IN bits	7 OUT bits	0 DYN bits

H.2.12 SFFDPNode

SFFDPNode		1 Node			
reserved	0				
FDP	1	3 DEF bits	2 IN bits	2 OUT bits	0 DYN bits

H.2.13 SFFITNode

SFFITNode		1 Node			
reserved	0				
FIT	1	4 DEF bits	4 IN bits	4 OUT bits	0 DYN bits

H.2.14 SFFaceDefMeshNode

SFFaceDefMeshNode		1 Node			
reserved	0				
FaceDefMesh	1	2 DEF bits	0 IN bits	0 OUT bits	0 DYN bits

H.2.15 SFFaceDefTablesNode

SFFaceDefTablesNode		1 Node			
reserved	0				
FaceDefTables	1	2 DEF bits	1 IN bits	1 OUT bits	0 DYN bits

H.2.16 SFFaceDefTransformNode

SFFaceDefTransformNode		1 Node			
reserved	0				
FaceDefTransform	1	3 DEF bits	0 IN bits	0 OUT bits	0 DYN bits

H.2.17 SFFogNode

SFFogNode		1 Node			
reserved	0				
Fog	1	2 DEF bits	2 IN bits	2 OUT bits	1 DYN bits

H.2.18 SFFontStyleNode

SFFontStyleNode		1 Node			
reserved	0				
FontStyle	1	4 DEF bits	0 IN bits	0 OUT bits	0 DYN bits

H.2.19 SFGeometryNode

SFGeometryNode		17 Nodes			
reserved	00000				
Bitmap	00001	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Box	00010	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Circle	00011	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Cone	00100	2 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Curve2D	00101	2 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
Cylinder	00110	3 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
ElevationGrid	00111	4 DEF bits	2 IN bits	2 OUT bits	0 DYN bits

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000356

Extrusion	01000	4 DEF bits	2 IN bits	0 OUT bits	0 DYN bits
IndexedFaceSet	01001	4 DEF bits	3 IN bits	2 OUT bits	0 DYN bits
IndexedFaceSet2D	01010	3 DEF bits	3 IN bits	2 OUT bits	0 DYN bits
IndexedLineSet	01011	3 DEF bits	2 IN bits	1 OUT bits	0 DYN bits
IndexedLineSet2D	01100	3 DEF bits	2 IN bits	1 OUT bits	0 DYN bits
PointSet	01101	1 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
PointSet2D	01110	1 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
Rectangle	01111	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Sphere	10000	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Text	10001	2 DEF bits	2 IN bits	2 OUT bits	1 DYN bits

H.2.20 SFLinePropertiesNode

SFLinePropertiesNode		**1 Node**			
reserved	0				
LineProperties	1	2 DEF bits	2 IN bits	2 OUT bits	1 DYN bits

H.2.21 SFMaterialNode

SFMaterialNode		**2 Nodes**			
reserved	00				
Material	01	3 DEF bits	3 IN bits	3 OUT bits	3 DYN bits
Material2D	10	2 DEF bits	2 IN bits	2 OUT bits	1 DYN bits

H.2.22 SFNavigationInfoNode

SFNavigationInfoNode		**1 Node**			
reserved	0				
NavigationInfo	1	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits

H.2.23 SFNormalNode

SFNormalNode		1 Node			
reserved	0				
Normal	1	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits

H.2.24 SFStreamingNode

SFStreamingNode		**5 Nodes**			
reserved	000				
AnimationStream	001	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
AudioClip	010	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
AudioSource	011	3 DEF bits	3 IN bits	3 OUT bits	1 DYN bits
Inline	100	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
MovieTexture	101	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits

H.2.25 SFTextureCoordinateNode

SFTextureCoordinateNode		**1 Node**			
reserved	0				
TextureCoordinate	1	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits

H.2.26 SFTextureNode

SFTextureNode		**5 Nodes**			
reserved	000				
CompositeTexture2D	001	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
CompositeTexture3D	010	3 DEF bits	4 IN bits	3 OUT bits	0 DYN bits
ImageTexture	011	2 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
MovieTexture	100	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
PixelTexture	101	2 DEF bits	0 IN bits	0 OUT bits	0 DYN bits

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000357

H.2.27 SFTextureTransformNode

SFTextureTransformNode		1 Node			
reserved	0				
TextureTransform	1	2 DEF bits	2 IN bits	2 OUT bits	2 DYN bits

H.2.28 SFTopNode

SFTopNode		4 Nodes			
reserved	000				
Group	001	0 DEF bits	2 IN bits	0 OUT bits	0 DYN bits
Layer2D	010	2 DEF bits	3 IN bits	2 OUT bits	0 DYN bits
Layer3D	011	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
OrderedGroup	100	1 DEF bits	2 IN bits	1 OUT bits	0 DYN bits

H.2.29 SFViewpointNode

SFViewpointNode		1 Node			
reserved	0				
Viewpoint	1	3 DEF bits	3 IN bits	3 OUT bits	2 DYN bits

H.2.30 SFViewportNode

SFViewportNode		0 Nodes			
Reserved	0				

H.2.31 SFVisemeNode

SFVisemeNode		1 Node			
reserved	0				
Viseme	1	2 DEF bits	2 IN bits	2 OUT bits	0 DYN bits

H.2.32 SFWorldNode

SFWorldNode		100 Nodes			
reserved	0000000				
Anchor	0000001	2 DEF bits	3 IN bits	2 OUT bits	0 DYN bits
AnimationStream	0000010	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Appearance	0000011	2 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
AudioBuffer	0000100	3 DEF bits	3 IN bits	4 OUT bits	0 DYN bits
AudioClip	0000101	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
AudioDelay	0000110	2 DEF bits	2 IN bits	1 OUT bits	0 DYN bits
AudioFX	0000111	3 DEF bits	3 IN bits	2 OUT bits	0 DYN bits
AudioMix	0001000	3 DEF bits	3 IN bits	2 OUT bits	0 DYN bits
AudioSource	0001001	3 DEF bits	3 IN bits	3 OUT bits	1 DYN bits
AudioSwitch	0001010	2 DEF bits	2 IN bits	1 OUT bits	0 DYN bits
Background	0001011	4 DEF bits	4 IN bits	4 OUT bits	2 DYN bits
Background2D	0001100	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
Billboard	0001101	1 DEF bits	2 IN bits	1 OUT bits	0 DYN bits
Bitmap	0001110	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Box	0001111	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Circle	0010000	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Collision	0010001	2 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
Color	0010010	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
ColorInterpolator	0010011	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
CompositeTexture2D	0010100	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
CompositeTexture3D	0010101	3 DEF bits	4 IN bits	3 OUT bits	0 DYN bits
Conditional	0010110	0 DEF bits	2 IN bits	1 OUT bits	0 DYN bits
Cone	0010111	2 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Coordinate	0011000	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000358

Coordinate2D	0011001	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
CoordinateInterpolator	0011010	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
CoordinateInterpolator2D	0011011	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
Curve2D	0011100	2 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
Cylinder	0011101	3 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
CylinderSensor	0011110	3 DEF bits	3 IN bits	4 OUT bits	0 DYN bits
DirectionalLight	0011111	3 DEF bits	3 IN bits	3 OUT bits	2 DYN bits
DiscSensor	0100000	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
ElevationGrid	0100001	4 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
Expression	0100010	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Extrusion	0100011	4 DEF bits	2 IN bits	0 OUT bits	0 DYN bits
Face	0100100	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
FaceDefMesh	0100101	2 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
FaceDefTables	0100110	2 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
FaceDefTransform	0100111	3 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
FAP	0101000	7 DEF bits	7 IN bits	7 OUT bits	0 DYN bits
FDP	0101001	3 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
FIT	0101010	4 DEF bits	4 IN bits	4 OUT bits	0 DYN bits
Fog	0101011	2 DEF bits	2 IN bits	2 OUT bits	1 DYN bits
FontStyle	0101100	4 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Form	0101101	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Group	0101110	0 DEF bits	2 IN bits	0 OUT bits	0 DYN bits
ImageTexture	0101111	2 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
IndexedFaceSet	0110000	4 DEF bits	3 IN bits	2 OUT bits	0 DYN bits
IndexedFaceSet2D	0110001	3 DEF bits	3 IN bits	2 OUT bits	0 DYN bits
IndexedLineSet	0110010	3 DEF bits	2 IN bits	1 OUT bits	0 DYN bits
IndexedLineSet2D	0110011	3 DEF bits	2 IN bits	1 OUT bits	0 DYN bits
Inline	0110100	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
LOD	0110101	2 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Layer2D	0110110	2 DEF bits	3 IN bits	2 OUT bits	0 DYN bits
Layer3D	0110111	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Layout	0111000	4 DEF bits	4 IN bits	4 OUT bits	2 DYN bits
LineProperties	0111001	2 DEF bits	2 IN bits	2 OUT bits	1 DYN bits
ListeningPoint	0111010	2 DEF bits	2 IN bits	3 OUT bits	1 DYN bits
Material	0111011	3 DEF bits	3 IN bits	3 OUT bits	3 DYN bits
Material2D	0111100	2 DEF bits	2 IN bits	2 OUT bits	1 DYN bits
MovieTexture	0111101	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
NavigationInfo	0111110	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Normal	0111111	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
NormalInterpolator	1000000	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
OrderedGroup	1000001	1 DEF bits	2 IN bits	1 OUT bits	0 DYN bits
OrientationInterpolator	1000010	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
PixelTexture	1000011	2 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
PlaneSensor	1000100	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
PlaneSensor2D	1000101	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
PointLight	1000110	3 DEF bits	3 IN bits	3 OUT bits	3 DYN bits
PointSet	1000111	1 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
PointSet2D	1001000	1 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
PositionInterpolator	1001001	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
PositionInterpolator2D	1001010	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
ProximitySensor2D	1001011	2 DEF bits	2 IN bits	3 OUT bits	0 DYN bits
ProximitySensor	1001100	2 DEF bits	2 IN bits	3 OUT bits	0 DYN bits
QuantizationParameter	1001101	6 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Rectangle	1001110	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
ScalarInterpolator	1001111	1 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
Script	1010000	2 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
Shape	1010001	1 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
Sound	1010010	4 DEF bits	4 IN bits	4 OUT bits	3 DYN bits
Sound2D	1010011	2 DEF bits	2 IN bits	2 OUT bits	1 DYN bits
Sphere	1010100	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
SphereSensor	1010101	2 DEF bits	2 IN bits	3 OUT bits	0 DYN bits
SpotLight	1010110	4 DEF bits	4 IN bits	4 OUT bits	4 DYN bits

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000359

Switch	1010111	1 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
TermCap	1011000	0 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
Text	1011001	2 DEF bits	2 IN bits	2 OUT bits	1 DYN bits
TextureCoordinate	1011010	0 DEF bits	0 IN bits	0 OUT bits	0 DYN bits
TextureTransform	1011011	2 DEF bits	2 IN bits	2 OUT bits	2 DYN bits
TimeSensor	1011100	3 DEF bits	3 IN bits	4 OUT bits	0 DYN bits
TouchSensor	1011101	0 DEF bits	0 IN bits	3 OUT bits	0 DYN bits
Transform	1011110	3 DEF bits	3 IN bits	3 OUT bits	3 DYN bits
Transform2D	1011111	3 DEF bits	3 IN bits	3 OUT bits	3 DYN bits
Valuator	1100000	4 DEF bits	5 IN bits	5 OUT bits	0 DYN bits
Viewpoint	1100001	3 DEF bits	3 IN bits	3 OUT bits	2 DYN bits
VisibilitySensor	1100010	2 DEF bits	2 IN bits	3 OUT bits	1 DYN bits
Viseme	1100011	2 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
WorldInfo	1100100	1 DEF bits	0 IN bits	0 OUT bits	0 DYN bits

H.3 Node Tables for Extended Nodes

Node Name	Node Data Type list					nodeType/NDT		
Field name	*Field type*	*DEF id*	*IN id*	*OUT id*	*DYN id*	*[min, max]*	*Quantizer id*	*Animation method*

H.3.1 AcousticMaterial

AcousticMaterial	SFMaterialNode SFWorldNode					10 0010		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
ambientIntensity	SFFloat	0000	000	000	000	[0, 1]	4	8
diffuseColor	SFColor	0001	001	001	001	[0, 1]	4	8
emissiveColor	SFColor	0010	010	010	010	[0, 1]	4	8
shininess	SFFloat	0011	011	011	011	[0, 1]	4	8
specularColor	SFColor	0100	100	100	100	[0, 1]	4	8
transparency	SFFloat	0101	101	101	101	[0, 1]	4	8
reffunc	MFFloat	0110				[-I, +I]	0	
transfunc	MFFloat	0111				[-I, +I]	0	
refFrequency	MFFloat	1000				[0, +I]	0	
transFrequency	MFFloat	1001				[0, +I]	0	

H.3.2 AcousticScene

AcousticScene	SFWorldNode SF3DNode					0011 010		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
center	SFVec3f	000				[-I, +I]	1	
Size	SFVec3f	001				[-I, +I]	11	
reverbTime	MFTime	010				[0, +I]	0	
reverbFreq	MFFloat	011				[0, +I]	0	
reverbLevel	SFFloat	100	0	0	0	[0, +I]	0	7
reverbDelay	SFTime	101	1	1	1	[0, +I]	0	0

H.3.3 ApplicationWindow

ApplicationWindow	SFWorldNode SF2DNode					0100 10		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
isActive	SFBool	000	000	000				
startTime	SFTime	001	001	001		[-I, +I]	0	
stopTime	SFTime	010	010	010		[-I, +I]	0	
description	SFString	011	011	011				
parameter	MFString	100	100	100				
url	MFURL	101	101	101				
size		SFVec2f	110	110	110		[-I, +I]	12

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000360

H.3.4 BAP

BAP	SFWorldNode SFBAPNode					0101 10		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
sacroiliac_tilt	SFInt32	000000000	000000000	000000000		[-I, +I]	0	
sacroiliac_torsion	SFInt32	000000001	000000001	000000001		[-I, +I]	0	
sacroiliac_roll	SFInt32	000000010	000000010	000000010		[-I, +I]	0	
l_hip_flexion	SFInt32	000000011	000000011	000000011		[-I, +I]	0	
r_hip_flexion	SFInt32	000000100	000000100	000000100		[-I, +I]	0	
l_hip_abduct	SFInt32	000000101	000000101	000000101		[-I, +I]	0	
r_hip_abduct	SFInt32	000000110	000000110	000000110		[-I, +I]	0	
l_hip_twisting	SFInt32	000000111	000000111	000000111		[-I, +I]	0	
r_hip_twisting	SFInt32	000001000	000001000	000001000		[-I, +I]	0	
l_knee_flexion	SFInt32	000001001	000001001	000001001		[-I, +I]	0	
r_knee_flexion	SFInt32	000001010	000001010	000001010		[-I, +I]	0	
l_knee_twisting	SFInt32	000001011	000001011	000001011		[-I, +I]	0	
r_knee_twisting	SFInt32	000001100	000001100	000001100		[-I, +I]	0	
l_ankle_flexion	SFInt32	000001101	000001101	000001101		[-I, +I]	0	
r_ankle_flexion	SFInt32	000001110	000001110	000001110		[-I, +I]	0	
l_ankle_twisting	SFInt32	000001111	000001111	000001111		[-I, +I]	0	
r_ankle_twisting	SFInt32	000010000	000010000	000010000		[-I, +I]	0	
l_subtalar_flexion	SFInt32	000010001	000010001	000010001		[-I, +I]	0	
r_subtalar_flexion	SFInt32	000010010	000010010	000010010		[-I, +I]	0	
l_midtarsal_flexion	SFInt32	000010011	000010011	000010011		[-I, +I]	0	
r_midtarsal_flexion	SFInt32	000010100	000010100	000010100		[-I, +I]	0	
l_metatarsal_flexion	SFInt32	000010101	000010101	000010101		[-I, +I]	0	
r_metatarsal_flexion	SFInt32	000010110	000010110	000010110		[-I, +I]	0	
l_sternoclavicular_abduct	SFInt32	000010111	000010111	000010111		[-I, +I]	0	
r_sternoclavicular_abduct	SFInt32	000011000	000011000	000011000		[-I, +I]	0	
l_sternoclavicular_rotate	SFInt32	000011001	000011001	000011001		[-I, +I]	0	
r_sternoclavicular_rotate	SFInt32	000011010	000011010	000011010		[-I, +I]	0	
l_acromioclavicular_abduct	SFInt32	000011011	000011011	000011011		[-I, +I]	0	
r_acromioclavicular_abduct	SFInt32	000011100	000011100	000011100		[-I, +I]	0	
l_acromioclavicular_rotate	SFInt32	000011101	000011101	000011101		[-I, +I]	0	
r_acromioclavicular_rotate	SFInt32	000011110	000011110	000011110		[-I, +I]	0	
l_shoulder_flexion	SFInt32	000011111	000011111	000011111		[-I, +I]	0	
r_shoulder_flexion	SFInt32	000100000	000100000	000100000		[-I, +I]	0	
l_shoulder_abduct	SFInt32	000100001	000100001	000100001		[-I, +I]	0	
r_shoulder_abduct	SFInt32	000100010	000100010	000100010		[-I, +I]	0	
l_shoulder_twisting	SFInt32	000100011	000100011	000100011		[-I, +I]	0	
r_shoulder_twisting	SFInt32	000100100	000100100	000100100		[-I, +I]	0	
l_elbow_flexion	SFInt32	000100101	000100101	000100101		[-I, +I]	0	
r_elbow_flexion	SFInt32	000100110	000100110	000100110		[-I, +I]	0	
l_elbow_twisting	SFInt32	000100111	000100111	000100111		[-I, +I]	0	
r_elbow_twisting	SFInt32	000101000	000101000	000101000		[-I, +I]	0	
l_wrist_flexion	SFInt32	000101001	000101001	000101001		[-I, +I]	0	
r_wrist_flexion	SFInt32	000101010	000101010	000101010		[-I, +I]	0	
l_wrist_pivot	SFInt32	000101011	000101011	000101011		[-I, +I]	0	
r_wrist_pivot	SFInt32	000101100	000101100	000101100		[-I, +I]	0	
l_wrist_twisting	SFInt32	000101101	000101101	000101101		[-I, +I]	0	
r_wrist_twisting	SFInt32	000101110	000101110	000101110		[-I, +I]	0	
Skullbase_roll	SFInt32	000101111	000101111	000101111		[-I, +I]	0	
Skullbase_torsion	SFInt32	000110000	000110000	000110000		[-I, +I]	0	
Skullbase_tilt	SFInt32	000110001	000110001	000110001		[-I, +I]	0	
vc1roll	SFInt32	000110010	000110010	000110010		[-I, +I]	0	
vc1torsion	SFInt32	000110011	000110011	000110011		[-I, +I]	0	
vc1tilt	SFInt32	000110100	000110100	000110100		[-I, +I]	0	
vc2roll	SFInt32	000110101	000110101	000110101		[-I, +I]	0	
vc2torsion	SFInt32	000110110	000110110	000110110		[-I, +I]	0	
vc2tilt	SFInt32	000110111	000110111	000110111		[-I, +I]	0	
vc3roll	SFInt32	000111000	000111000	000111000		[-I, +I]	0	

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000361

vc3torsion	SFInt32	000111001	000111001	000111001		[-I, +I]	0	
vc3tilt	SFInt32	000111010	000111010	000111010		[-I, +I]	0	
vc4roll	SFInt32	000111011	000111011	000111011		[-I, +I]	0	
vc4torsion	SFInt32	000111100	000111100	000111100		[-I, +I]	0	
vc4tilt	SFInt32	000111101	000111101	000111101		[-I, +I]	0	
vc5roll	SFInt32	000111110	000111110	000111110		[-I, +I]	0	
vc5torsion	SFInt32	000111111	000111111	000111111		[-I, +I]	0	
vc5tilt	SFInt32	001000000	001000000	001000000		[-I, +I]	0	
vc6roll	SFInt32	001000001	001000001	001000001		[-I, +I]	0	
vc6torsion	SFInt32	001000010	001000010	001000010		[-I, +I]	0	
vc6tilt	SFInt32	001000011	001000011	001000011		[-I, +I]	0	
vc7roll	SFInt32	001000100	001000100	001000100		[-I, +I]	0	
vc7torsion	SFInt32	001000101	001000101	001000101		[-I, +I]	0	
vc7tilt	SFInt32	001000110	001000110	001000110		[-I, +I]	0	
vt1roll	SFInt32	001000111	001000111	001000111		[-I, +I]	0	
vt1torsion	SFInt32	001001000	001001000	001001000		[-I, +I]	0	
vt1tilt	SFInt32	001001001	001001001	001001001		[-I, +I]	0	
vt2roll	SFInt32	001001010	001001010	001001010		[-I, +I]	0	
vt2torsion	SFInt32	001001011	001001011	001001011		[-I, +I]	0	
vt2tilt	SFInt32	001001100	001001100	001001100		[-I, +I]	0	
vt3roll	SFInt32	001001101	001001101	001001101		[-I, +I]	0	
vt3torsion	SFInt32	001001110	001001110	001001110		[-I, +I]	0	
vt3tilt	SFInt32	001001111	001001111	001001111		[-I, +I]	0	
vt4roll	SFInt32	001010000	001010000	001010000		[-I, +I]	0	
vt4torsion	SFInt32	001010001	001010001	001010001		[-I, +I]	0	
vt4tilt	SFInt32	001010010	001010010	001010010		[-I, +I]	0	
vt5roll	SFInt32	001010011	001010011	001010011		[-I, +I]	0	
vt5torsion	SFInt32	001010100	001010100	001010100		[-I, +I]	0	
vt5tilt	SFInt32	001010101	001010101	001010101		[-I, +I]	0	
vt6roll	SFInt32	001010110	001010110	001010110		[-I, +I]	0	
vt6torsion	SFInt32	001010111	001010111	001010111		[-I, +I]	0	
vt6tilt	SFInt32	001011000	001011000	001011000		[-I, +I]	0	
vt7roll	SFInt32	001011001	001011001	001011001		[-I, +I]	0	
vt7torsion	SFInt32	001011010	001011010	001011010		[-I, +I]	0	
vt7tilt	SFInt32	001011011	001011011	001011011		[-I, +I]	0	
vt8roll	SFInt32	001011100	001011100	001011100		[-I, +I]	0	
vt8torsion	SFInt32	001011101	001011101	001011101		[-I, +I]	0	
vt8tilt	SFInt32	001011110	001011110	001011110		[-I, +I]	0	
vt9roll	SFInt32	001011111	001011111	001011111		[-I, +I]	0	
vt9torsion	SFInt32	001100000	001100000	001100000		[-I, +I]	0	
vt9tilt	SFInt32	001100001	001100001	001100001		[-I, +I]	0	
vt10roll	SFInt32	001100010	001100010	001100010		[-I, +I]	0	
vt10torsion	SFInt32	001100011	001100011	001100011		[-I, +I]	0	
vt10tilt	SFInt32	001100100	001100100	001100100		[-I, +I]	0	
vt11roll	SFInt32	001100101	001100101	001100101		[-I, +I]	0	
vt11torsion	SFInt32	001100110	001100110	001100110		[-I, +I]	0	
vt11tilt	SFInt32	001100111	001100111	001100111		[-I, +I]	0	
vt12roll	SFInt32	001101000	001101000	001101000		[-I, +I]	0	
vt12torsion	SFInt32	001101001	001101001	001101001		[-I, +I]	0	
vt12tilt	SFInt32	001101010	001101010	001101010		[-I, +I]	0	
vl1roll	SFInt32	001101011	001101011	001101011		[-I, +I]	0	
vl1torsion	SFInt32	001101100	001101100	001101100		[-I, +I]	0	
vl1tilt	SFInt32	001101101	001101101	001101101		[-I, +I]	0	
vl2roll	SFInt32	001101110	001101110	001101110		[-I, +I]	0	
vl2torsion	SFInt32	001101111	001101111	001101111		[-I, +I]	0	
vl2tilt	SFInt32	001110000	001110000	001110000		[-I, +I]	0	
vl3roll	SFInt32	001110001	001110001	001110001		[-I, +I]	0	
vl3torsion	SFInt32	001110010	001110010	001110010		[-I, +I]	0	
vl3tilt	SFInt32	001110011	001110011	001110011		[-I, +I]	0	
vl4roll	SFInt32	001110100	001110100	001110100		[-I, +I]	0	
vl4torsion	SFInt32	001110101	001110101	001110101		[-I, +I]	0	
vl4tilt	SFInt32	001110110	001110110	001110110		[-I, +I]	0	

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000362

vl5roll	SFInt32	001110111	001110111	001110111		[-I, +I]	0	
vl5torsion	SFInt32	001111000	001111000	001111000		[-I, +I]	0	
vl5tilt	SFInt32	001111001	001111001	001111001		[-I, +I]	0	
l_pinky0_flexion	SFInt32	001111010	001111010	001111010		[-I, +I]	0	
r_pinky0_flexion	SFInt32	001111011	001111011	001111011		[-I, +I]	0	
l_pinky1_flexion	SFInt32	001111100	001111100	001111100		[-I, +I]	0	
r_pinky1_flexion	SFInt32	001111101	001111101	001111101		[-I, +I]	0	
l_pinky1_pivot	SFInt32	001111110	001111110	001111110		[-I, +I]	0	
r_pinky1_pivot	SFInt32	001111111	001111111	001111111		[-I, +I]	0	
l_pinky1_twisting	SFInt32	010000000	010000000	010000000		[-I, +I]	0	
r_pinky1_twisting	SFInt32	010000001	010000001	010000001		[-I, +I]	0	
l_pinky2_flexion	SFInt32	010000010	010000010	010000010		[-I, +I]	0	
r_pinky2_flexion	SFInt32	010000011	010000011	010000011		[-I, +I]	0	
l_pinky3_flexion	SFInt32	010000100	010000100	010000100		[-I, +I]	0	
r_pinky3_flexion	SFInt32	010000101	010000101	010000101		[-I, +I]	0	
l_ring0_flexion	SFInt32	010000110	010000110	010000110		[-I, +I]	0	
r_ring0_flexion	SFInt32	010000111	010000111	010000111		[-I, +I]	0	
l_ring1_flexion	SFInt32	010001000	010001000	010001000		[-I, +I]	0	
r_ring1_flexion	SFInt32	010001001	010001001	010001001		[-I, +I]	0	
l_ring1_pivot	SFInt32	010001010	010001010	010001010		[-I, +I]	0	
r_ring1_pivot	SFInt32	010001011	010001011	010001011		[-I, +I]	0	
l_ring1_twisting	SFInt32	010001100	010001100	010001100		[-I, +I]	0	
r_ring1_twisting	SFInt32	010001101	010001101	010001101		[-I, +I]	0	
l_ring2_flexion	SFInt32	010001110	010001110	010001110		[-I, +I]	0	
r_ring2_flexion	SFInt32	010001111	010001111	010001111		[-I, +I]	0	
l_ring3_flexion	SFInt32	010010000	010010000	010010000		[-I, +I]	0	
r_ring3_flexion	SFInt32	010010001	010010001	010010001		[-I, +I]	0	
l_middle0_flexion	SFInt32	010010010	010010010	010010010		[-I, +I]	0	
r_middle0_flexion	SFInt32	010010011	010010011	010010011		[-I, +I]	0	
l_middle1_flexion	SFInt32	010010100	010010100	010010100		[-I, +I]	0	
r_middle1_flexion	SFInt32	010010101	010010101	010010101		[-I, +I]	0	
l_middle1_pivot	SFInt32	010010110	010010110	010010110		[-I, +I]	0	
r_middle1_pivot	SFInt32	010010111	010010111	010010111		[-I, +I]	0	
l_middle1_twisting	SFInt32	010011000	010011000	010011000		[-I, +I]	0	
r_middle1_twisting	SFInt32	010011001	010011001	010011001		[-I, +I]	0	
l_middle2_flexion	SFInt32	010011010	010011010	010011010		[-I, +I]	0	
r_middle2_flexion	SFInt32	010011011	010011011	010011011		[-I, +I]	0	
l_middle3_flexion	SFInt32	010011100	010011100	010011100		[-I, +I]	0	
r_middle3_flexion	SFInt32	010011101	010011101	010011101		[-I, +I]	0	
l_index0_flexion	SFInt32	010011110	010011110	010011110		[-I, +I]	0	
r_index0_flexion	SFInt32	010011111	010011111	010011111		[-I, +I]	0	
l_index1_flexion	SFInt32	010100000	010100000	010100000		[-I, +I]	0	
r_index1_flexion	SFInt32	010100001	010100001	010100001		[-I, +I]	0	
l_index1_pivot	SFInt32	010100010	010100010	010100010		[-I, +I]	0	
r_index1_pivot	SFInt32	010100011	010100011	010100011		[-I, +I]	0	
l_index1_twisting	SFInt32	010100100	010100100	010100100		[-I, +I]	0	
r_index1_twisting	SFInt32	010100101	010100101	010100101		[-I, +I]	0	
l_index2_flexion	SFInt32	010100110	010100110	010100110		[-I, +I]	0	
r_index2_flexion	SFInt32	010100111	010100111	010100111		[-I, +I]	0	
l_index3_flexion	SFInt32	010101000	010101000	010101000		[-I, +I]	0	
r_index3_flexion	SFInt32	010101001	010101001	010101001		[-I, +I]	0	
l_thumb1_flexion	SFInt32	010101010	010101010	010101010		[-I, +I]	0	
r_thumb1_flexion	SFInt32	010101011	010101011	010101011		[-I, +I]	0	
l_thumb1_pivot	SFInt32	010101100	010101100	010101100		[-I, +I]	0	
r_thumb1_pivot	SFInt32	010101101	010101101	010101101		[-I, +I]	0	
l_thumb1_twisting	SFInt32	010101110	010101110	010101110		[-I, +I]	0	
r_thumb1_twisting	SFInt32	010101111	010101111	010101111		[-I, +I]	0	
l_thumb2_flexion	SFInt32	010110000	010110000	010110000		[-I, +I]	0	
r_thumb2_flexion	SFInt32	010110001	010110001	010110001		[-I, +I]	0	
l_thumb3_flexion	SFInt32	010110010	010110010	010110010		[-I, +I]	0	
r_thumb3_flexion	SFInt32	010110011	010110011	010110011		[-I, +I]	0	
humanoidRoot_tr_vertical	SFInt32	010110100	010110100	010110100		[-I, +I]	0	

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000363

humanoidRoot_tr_lateral	SFInt32	010110101	010110101	010110101		[-I, +I]	0	
humanoidRoot_tr_frontal	SFInt32	010110110	010110110	010110110		[-I, +I]	0	
humanoidRoot_rt_body_turn	SFInt32	010110111	010110111	010110111		[-I, +I]	0	
humanoidRoot_rt_body_roll	SFInt32	010111000	010111000	010111000		[-I, +I]	0	
humanoidRoot_rt_body_tilt	SFInt32	010111001	010111001	010111001		[-I, +I]	0	
extensionBap187	SFInt32	010111010	010111010	010111010		[-I, +I]	0	
extensionBap188	SFInt32	010111011	010111011	010111011		[-I, +I]	0	
extensionBap189	SFInt32	010111100	010111100	010111100		[-I, +I]	0	
extensionBap190	SFInt32	010111101	010111101	010111101		[-I, +I]	0	
extensionBap191	SFInt32	010111110	010111110	010111110		[-I, +I]	0	
extensionBap192	SFInt32	010111111	010111111	010111111		[-I, +I]	0	
extensionBap193	SFInt32	011000000	011000000	011000000		[-I, +I]	0	
extensionBap194	SFInt32	011000001	011000001	011000001		[-I, +I]	0	
extensionBap195	SFInt32	011000010	011000010	011000010		[-I, +I]	0	
extensionBap196	SFInt32	011000011	011000011	011000011		[-I, +I]	0	
extensionBap197	SFInt32	011000100	011000100	011000100		[-I, +I]	0	
extensionBap198	SFInt32	011000101	011000101	011000101		[-I, +I]	0	
extensionBap199	SFInt32	011000110	011000110	011000110		[-I, +I]	0	
extensionBap200	SFInt32	011000111	011000111	011000111		[-I, +I]	0	
extensionBap201	SFInt32	011001000	011001000	011001000		[-I, +I]	0	
extensionBap202	SFInt32	011001001	011001001	011001001		[-I, +I]	0	
extensionBap203	SFInt32	011001010	011001010	011001010		[-I, +I]	0	
extensionBap204	SFInt32	011001011	011001011	011001011		[-I, +I]	0	
extensionBap205	SFInt32	011001100	011001100	011001100		[-I, +I]	0	
extensionBap206	SFInt32	011001101	011001101	011001101		[-I, +I]	0	
extensionBap207	SFInt32	011001110	011001110	011001110		[-I, +I]	0	
extensionBap208	SFInt32	011001111	011001111	011001111		[-I, +I]	0	
extensionBap209	SFInt32	011010000	011010000	011010000		[-I, +I]	0	
extensionBap210	SFInt32	011010001	011010001	011010001		[-I, +I]	0	
extensionBap211	SFInt32	011010010	011010010	011010010		[-I, +I]	0	
extensionBap212	SFInt32	011010011	011010011	011010011		[-I, +I]	0	
extensionBap213	SFInt32	011010100	011010100	011010100		[-I, +I]	0	
extensionBap214	SFInt32	011010101	011010101	011010101		[-I, +I]	0	
extensionBap215	SFInt32	011010110	011010110	011010110		[-I, +I]	0	
extensionBap216	SFInt32	011010111	011010111	011010111		[-I, +I]	0	
extensionBap217	SFInt32	011011000	011011000	011011000		[-I, +I]	0	
extensionBap218	SFInt32	011011001	011011001	011011001		[-I, +I]	0	
extensionBap219	SFInt32	011011010	011011010	011011010		[-I, +I]	0	
extensionBap220	SFInt32	011011011	011011011	011011011		[-I, +I]	0	
extensionBap221	SFInt32	011011100	011011100	011011100		[-I, +I]	0	
extensionBap222	SFInt32	011011101	011011101	011011101		[-I, +I]	0	
extensionBap223	SFInt32	011011110	011011110	011011110		[-I, +I]	0	
extensionBap224	SFInt32	011011111	011011111	011011111		[-I, +I]	0	
extensionBap225	SFInt32	011100000	011100000	011100000		[-I, +I]	0	
extensionBap226	SFInt32	011100001	011100001	011100001		[-I, +I]	0	
extensionBap227	SFInt32	011100010	011100010	011100010		[-I, +I]	0	
extensionBap228	SFInt32	011100011	011100011	011100011		[-I, +I]	0	
extensionBap229	SFInt32	011100100	011100100	011100100		[-I, +I]	0	
extensionBap230	SFInt32	011100101	011100101	011100101		[-I, +I]	0	
extensionBap231	SFInt32	011100110	011100110	011100110		[-I, +I]	0	
extensionBap232	SFInt32	011100111	011100111	011100111		[-I, +I]	0	
extensionBap233	SFInt32	011101000	011101000	011101000		[-I, +I]	0	
extensionBap234	SFInt32	011101001	011101001	011101001		[-I, +I]	0	
extensionBap235	SFInt32	011101010	011101010	011101010		[-I, +I]	0	
extensionBap236	SFInt32	011101011	011101011	011101011		[-I, +I]	0	
extensionBap237	SFInt32	011101100	011101100	011101100		[-I, +I]	0	
extensionBap238	SFInt32	011101101	011101101	011101101		[-I, +I]	0	
extensionBap239	SFInt32	011101110	011101110	011101110		[-I, +I]	0	
extensionBap240	SFInt32	011101111	011101111	011101111		[-I, +I]	0	
extensionBap241	SFInt32	011110000	011110000	011110000		[-I, +I]	0	
extensionBap242	SFInt32	011110001	011110001	011110001		[-I, +I]	0	
extensionBap243	SFInt32	011110010	011110010	011110010		[-I, +I]	0	

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000364

extensionBap244	SFInt32	011110011	011110011	011110011		[-I, +I]	0	
extensionBap245	SFInt32	011110100	011110100	011110100		[-I, +I]	0	
extensionBap246	SFInt32	011110101	011110101	011110101		[-I, +I]	0	
extensionBap247	SFInt32	011110110	011110110	011110110		[-I, +I]	0	
extensionBap248	SFInt32	011110111	011110111	011110111		[-I, +I]	0	
extensionBap249	SFInt32	011111000	011111000	011111000		[-I, +I]	0	
extensionBap250	SFInt32	011111001	011111001	011111001		[-I, +I]	0	
extensionBap251	SFInt32	011111010	011111010	011111010		[-I, +I]	0	
extensionBap252	SFInt32	011111011	011111011	011111011		[-I, +I]	0	
extensionBap253	SFInt32	011111100	011111100	011111100		[-I, +I]	0	
extensionBap254	SFInt32	011111101	011111101	011111101		[-I, +I]	0	
extensionBap255	SFInt32	011111110	011111110	011111110		[-I, +I]	0	
extensionBap256	SFInt32	011111111	011111111	011111111		[-I, +I]	0	
extensionBap257	SFInt32	100000000	100000000	100000000		[-I, +I]	0	
extensionBap258	SFInt32	100000001	100000001	100000001		[-I, +I]	0	
extensionBap259	SFInt32	100000010	100000010	100000010		[-I, +I]	0	
extensionBap260	SFInt32	100000011	100000011	100000011		[-I, +I]	0	
extensionBap261	SFInt32	100000100	100000100	100000100		[-I, +I]	0	
extensionBap262	SFInt32	100000101	100000101	100000101		[-I, +I]	0	
extensionBap263	SFInt32	100000110	100000110	100000110		[-I, +I]	0	
extensionBap264	SFInt32	100000111	100000111	100000111		[-I, +I]	0	
extensionBap265	SFInt32	100001000	100001000	100001000		[-I, +I]	0	
extensionBap266	SFInt32	100001001	100001001	100001001		[-I, +I]	0	
extensionBap267	SFInt32	100001010	100001010	100001010		[-I, +I]	0	
extensionBap268	SFInt32	100001011	100001011	100001011		[-I, +I]	0	
extensionBap269	SFInt32	100001100	100001100	100001100		[-I, +I]	0	
extensionBap270	SFInt32	100001101	100001101	100001101		[-I, +I]	0	
extensionBap271	SFInt32	100001110	100001110	100001110		[-I, +I]	0	
extensionBap272	SFInt32	100001111	100001111	100001111		[-I, +I]	0	
extensionBap273	SFInt32	100010000	100010000	100010000		[-I, +I]	0	
extensionBap274	SFInt32	100010001	100010001	100010001		[-I, +I]	0	
extensionBap275	SFInt32	100010010	100010010	100010010		[-I, +I]	0	
extensionBap276	SFInt32	100010011	100010011	100010011		[-I, +I]	0	
extensionBap277	SFInt32	100010100	100010100	100010100		[-I, +I]	0	
extensionBap278	SFInt32	100010101	100010101	100010101		[-I, +I]	0	
extensionBap279	SFInt32	100010110	100010110	100010110		[-I, +I]	0	
extensionBap280	SFInt32	100010111	100010111	100010111		[-I, +I]	0	
extensionBap281	SFInt32	100011000	100011000	100011000		[-I, +I]	0	
extensionBap282	SFInt32	100011001	100011001	100011001		[-I, +I]	0	
extensionBap283	SFInt32	100011010	100011010	100011010		[-I, +I]	0	
extensionBap284	SFInt32	100011011	100011011	100011011		[-I, +I]	0	
extensionBap285	SFInt32	100011100	100011100	100011100		[-I, +I]	0	
extensionBap286	SFInt32	100011101	100011101	100011101		[-I, +I]	0	
extensionBap287	SFInt32	100011110	100011110	100011110		[-I, +I]	0	
extensionBap288	SFInt32	100011111	100011111	100011111		[-I, +I]	0	
extensionBap289	SFInt32	100100000	100100000	100100000		[-I, +I]	0	
extensionBap290	SFInt32	100100001	100100001	100100001		[-I, +I]	0	
extensionBap291	SFInt32	100100010	100100010	100100010		[-I, +I]	0	
extensionBap292	SFInt32	100100011	100100011	100100011		[-I, +I]	0	
extensionBap293	SFInt32	100100100	100100100	100100100		[-I, +I]	0	
extensionBap294	SFInt32	100100101	100100101	100100101		[-I, +I]	0	
extensionBap295	SFInt32	100100110	100100110	100100110		[-I, +I]	0	
extensionBap296	SFInt32	100100111	100100111	100100111		[-I, +I]	0	

H.3.5 BDP

BDP	SFWorldNode SFBDPNode					0110 10		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
bodyDefTables	MFBodyDefTableNode	0	0	0				
bodySceneGraph	MF3DNode	1	1	1				

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000365

H.3.6 Body

Body	SFWorldNode SF3DNode SF2DNode					0111 011 11		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
bdp	SFBDPNode	00	00	00				
bap	SFBAPNode	01	01	01				
renderedBody	MF3DNode	10	10	10				

H.3.7 BodyDefTable

BodyDefTable	SFWorldNode SFBodyDefTableNode					1000 10		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
bodySceneGraphNodeName	SFString	000	000	000				
bapIDs	MFInt32	001	001	001		[1, 296]	13,9	
vertexIds	MFInt32	010	010	010		[0, +I]	0	
bapCombinations	MFInt32	011	011	011		[-I, +I]	0	
displacements	MFVec3f	100	100	100				
numInterpolateKeys	SFInt32	101	101	101		[2, +I]	0	

H.3.8 BodySegmentConnectionHint

BodySegmentConnectionHint	SFWorldNode SFBodySegmentConnectionHintNode					1001 10		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
firstSegmentNodeName	SFString	00	00	00				
secondSegmentNodeName	SFString	01	01	01				
firstVertexIdList	MFInt32	10	10	10		[0, +I]	0	
secondVertexIdList		MFInt32	11	11	11		[0, +I]	

H.3.9 DirectiveSound

DirectiveSound	SFWorldNode SF3DNode					1010 100		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
direction	SFVec3f	0000	000	000	00	[-I, +I]	9	9
intensity	SFFloat	0001	001	001	01	[0, +I]	0	7
location	SFVec3f	0010	010	010	10	[-I, +I]	1	1
source	SFAudioNode	0011	011	011				
perceptualParameters	SFPerceptualParameterNode	0100	100	100				
roomEffect	SFBool	0101	101	101				
spatialize	SFBool	0110	110	110				
directivity	MFFloat	0111				[-I, +I]	0	
angles	MFFloat	1000				[0, 3.14159265]	6	
Frequency	MFFloat	1001				[0, +I]	0	
speedOfSound	SFFloat	1010				[0, +I]	1	
distance	SFFloat	1011				[0, +I]	0	
UseAirabs	SFBool	1100						

H.3.10 Hierarchical3DMesh

Hierarchical3DMesh	SFWorldNode SF3DNode					1011 101		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
triangleBudget	SFInt32		0			[-1, +I]	0	
level	SFFloat	0	1	0		[-1, +I]	0	
url	MFURL	1						
DoneLoading	SFBool			1				

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000366

H.3.11 MaterialKey

MaterialKey	SFWorldNode SFMaterialNode					1100 11		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
isKeyed	SFBool	000	000	000				
isRGB	SFBool	001	001	001				
keyColor	SFColor	010	010	010	00	[0, 1]	4	4
LowThreshold	SFFloat	011	011	011	01	[0, 1]	4	8
highThreshold	SFFloat	100	100	100	10	[0, 1]	4	8
transparency	SFFloat	101	101	101	11	[0, 1]	4	8

H.3.12 PerceptualParameters

PerceptualParameters	SFWorldNode SFPerceptualParameterNode					1101 10		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
sourcePresence	SFFloat	00000	00000	00000	00000	[-I, +I]	0	7
sourceWarmth	SFFloat	00001	00001	00001	00001	[-I, +I]	0	7
sourceBrilliance	SFFloat	00010	00010	00010	00010	[-I, +I]	0	7
roomPresence	SFFloat	00011	00011	00011	00011	[-I, +I]	0	7
runningReverberance	SFFloat	00100	00100	00100	00100	[-I, +I]	0	7
envelopment	SFFloat	00101	00101	00101	00101	[-I, +I]	0	7
lateReverberance	SFFloat	00110	00110	00110	00110	[-I, +I]	0	7
heavyness	SFFloat	00111	00111	00111	00111	[-I, +I]	0	7
liveness	SFFloat	01000	01000	01000	01000	[-I, +I]	0	7
omniDirectivity	MFFloat	01001	01001	01001	01001	[-I, +I]	0	7
directFilterGains	MFFloat	01010	01010	01010	01010	[-I, +I]	0	7
inputFilterGains	MFFloat	01011	01011	01011	01011	[-I, +I]	0	7
refDistance	SFFloat	01100	01100	01100	01100	[-I, +I]	0	7
freqLow	SFFloat	01101	01101	01101	01101	[-I, +I]	0	7
freqHigh	SFFloat	01110	01110	01110	01110	[-I, +I]	0	7
timeLimit1	SFTime	01111	01111	01111	01111	[-I, +I]	0	0
timeLimit2	SFTime	10000	10000	10000	10000	[-I, +I]	0	0
timeLimit3	SFTime	10001	10001	10001	10001	[-I, +I]	0	0
modalDensity	SFTime	10010	10010	10010	10010	[-I, +I]	0	0

H.4 Node Definition Type Tables for extended node types

Node Definition Type		Number of nodes			
Node name	nodeType	DEF	IN	OUT	DYN

H.4.1 SF2DNode

SF2DNode		**2 Nodes**			
reserved	00				
PROTO	01				
ApplicationWindow	10	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
Body	11	2 DEF bits	2 IN bits	2 OUT bits	N bits

H.4.2 SF3DNode

SF3DNode		**4 Nodes**			
reserved	000				
PROTO	001				
AcousticScene	010	3 DEF bits	1 IN bits	1 OUT bits	1 DYN bits
Body	011	2 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
DirectiveSound	100	4 DEF bits	3 IN bits	3 OUT bits	2 DYN bits
Hierarchical3DMesh	101	1 DEF bits	1 IN bits	1 OUT bits	bits

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000367

H.4.3 SFBAPNode

SFBAPNode		1 Node			
reserved	00				
PROTO	01				
BAP	10	9 DEF bits	9 IN bits	9 OUT bits	bits

H.4.4 SFBDPNode

SFBDPNode		1 Node			
reserved	00				
PROTO	01				
BDP	10	1 DEF bits	1 IN bits	1 OUT bits	0 DYN bits

H.4.5 SFBodyDefTableNode

SFBodyDefTableNode		1 Node			
reserved	00				
PROTO	01				
BodyDefTable	10	3 DEF bits	3 IN bits	3 OUT bits	N bits

H.4.6 SFBodySegmentConnectionHintNode

SFBodySegmentConnectionHintNode		1 Node			
reserved	00				
PROTO	01				
BodySegmentConnectionHint	10	2 DEF bits	2 IN bits	2 OUT bits	bits

H.4.7 SFMaterialNode

SFMaterialNode		2 Nodes			
reserved	00				
PROTO	01				
AcousticMaterial	10	4 DEF bits	3 IN bits	3 OUT bits	3 DYN bits
MaterialKey	11	3 DEF bits	3 IN bits	3 OUT bits	bits

H.4.8 SFPerceptualParameterNode

SFPerceptualParameterNode		1 Node			
reserved	00				
PROTO	01				
PerceptualParameters	10	5 DEF bits	5 IN bits	5 OUT bits	bits

H.4.9 SFWorldNode

SFWorldNode		12 Nodes			
reserved	0000				
PROTO	0001				
AcousticMaterial	0010	4 DEF bits	3 IN bits	3 OUT bits	3 DYN bits
AcousticScene	0011	3 DEF bits	1 IN bits	1 OUT bits	1 DYN bits
ApplicationWindow	0100	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
BAP	0101	9 DEF bits	9 IN bits	9 OUT bits	0 DYN bits
BDP	0110	1 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
Body	0111	2 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
BodyDefTable	1000	3 DEF bits	3 IN bits	3 OUT bits	0 DYN bits
BodySegmentConnectionHint	1001	2 DEF bits	2 IN bits	2 OUT bits	0 DYN bits
DirectiveSound	1010	4 DEF bits	3 IN bits	3 OUT bits	2 DYN bits
Hierarchical3DMesh	1011	1 DEF bits	1 IN bits	1 OUT bits	0 DYN bits
MaterialKey	1100	3 DEF bits	3 IN bits	3 OUT bits	2 DYN bits
PerceptualParameters	1101	5 DEF bits	5 IN bits	5 OUT bits	5 DYN bits

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000368

Annex I
(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 2001 – All rights reserved

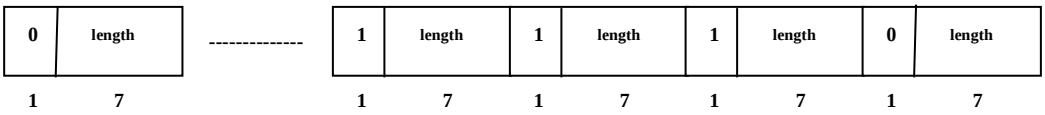
Hulu, LLC - Ex. 1024, Page 000369

Annex J
(informative)

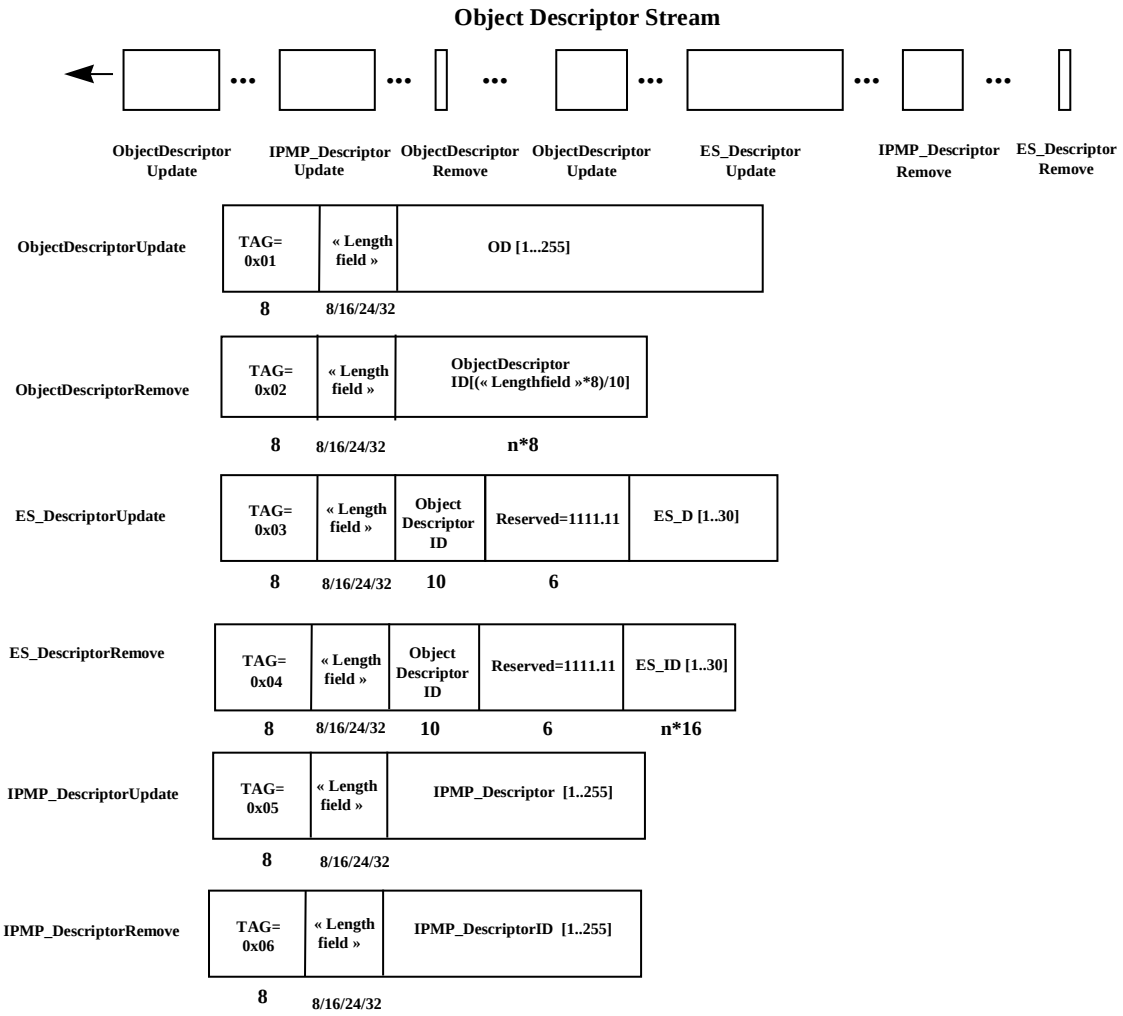
Graphical representation of object descriptor and sync layer syntax

J.1 Length encoding of descriptors and commands

« Length field » : from one byte, up to four bytes



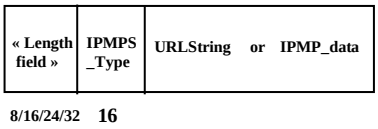
J.2 Object Descriptor Stream and OD commands



© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000370

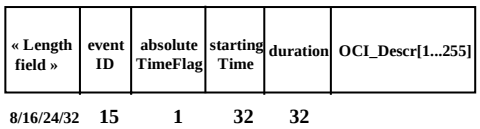
J.3 IPMP stream

IPMP message



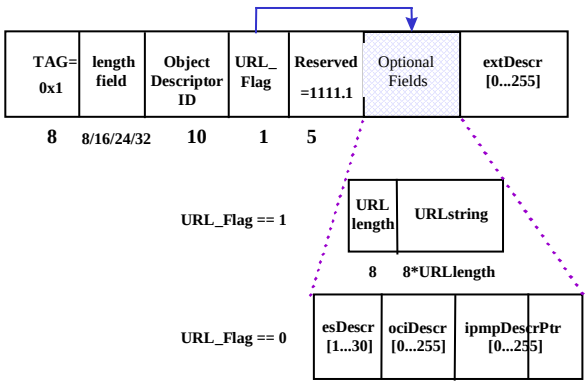
J.4 OCI stream

OCI_Events

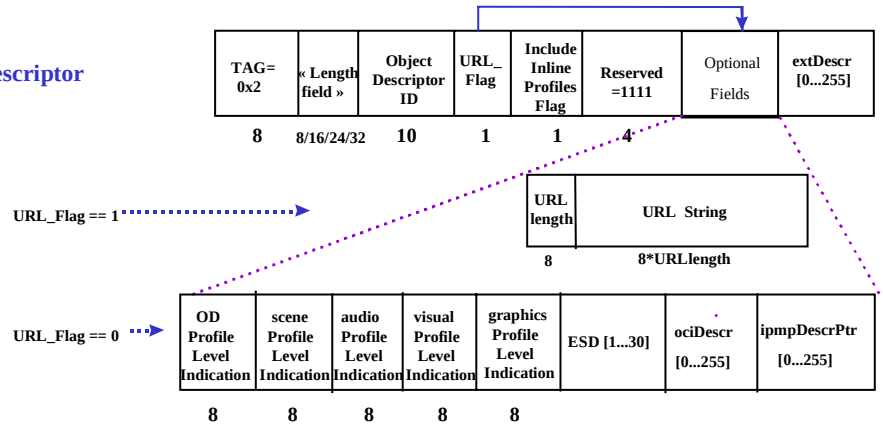


J.5 Object descriptor and its components

ObjectDescriptor



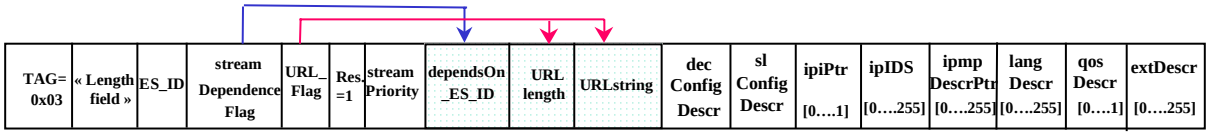
InitialObjectDescriptor



© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000371

ES_Descriptor

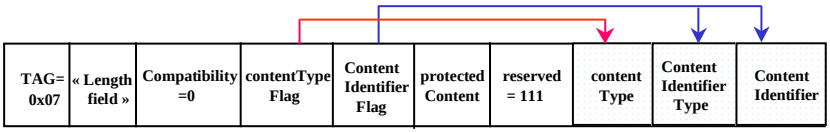


TAG= 0x03	« Length field »	ES_ID	stream Dependence Flag	URL_ Flag	Res =1	stream Priority	dependsOn _ES_ID	URL length	URLstring	dec Config Descr	sl Config Descr	ipiPtr [0….1]	ipIDS [0….255]	ipmp DescrPtr [0….255]	lang Descr [0….255]	qos Descr [0….1]	extDescr [0….255]
8	8/16/24/32	16	1	1	1	5	16	8	8*URLlength								

DecoderConfigDescriptor

TAG =0x04	« Length field »	object Type Indication	stream Type	upStream	reserved =1	bufferSizeDB	maxBitRate	avg BitRate	dec Specific Info[0..1]
8	8/16/24/32	8	6	1	1	24	32	32	

ContentIdentificationDescriptor



TAG= 0x07	« Length field »	Compatibility =0	contentType Flag	Content Identifier Flag	protected Content	reserved = 111	content Type	Content Identifier Type	Content Identifier
8	8/16/24/32	2	1	1	1	3	8	8	n*8

SupplementaryContentIdentificationDescriptor

TAG= 0x8	« Length field »	languageCode	Suppl Content Identifier Title Length	Suppl Content Identifier Title	Suppl Content Identifier Value Length	Suppl Content Identifier Value
8	8/16/24/32	24	8	8*TitleLength	8	8*ValueLength

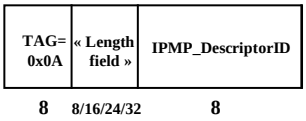
IPI_DescrPointer

TAG= 0x09	« Length field »	IPI_ES_ID
8	8/16/24/32	16

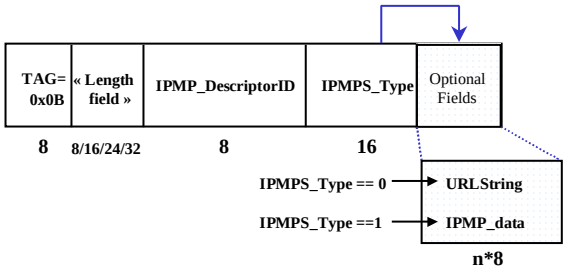
© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000372

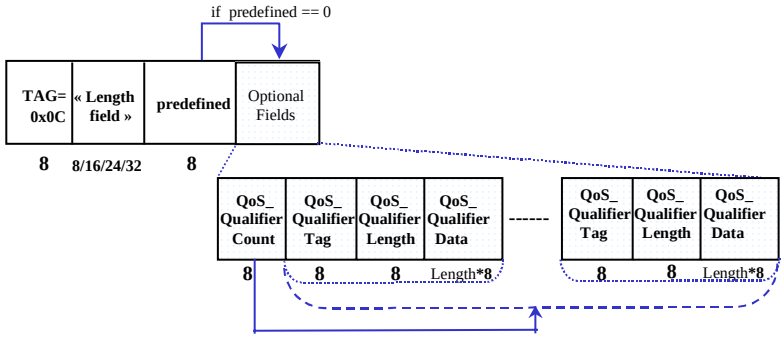
IPMP_DescriptorPointer



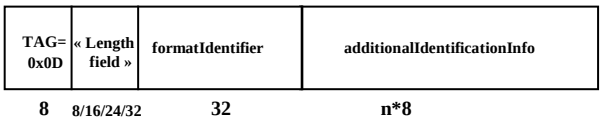
IPMPDescriptor



QoS_Descriptor



RegistrationDescriptor

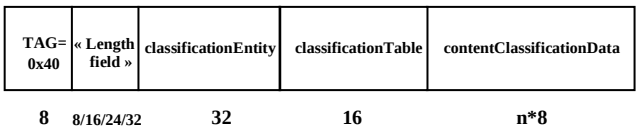


© ISO/IEC 2001 – All rights reserved

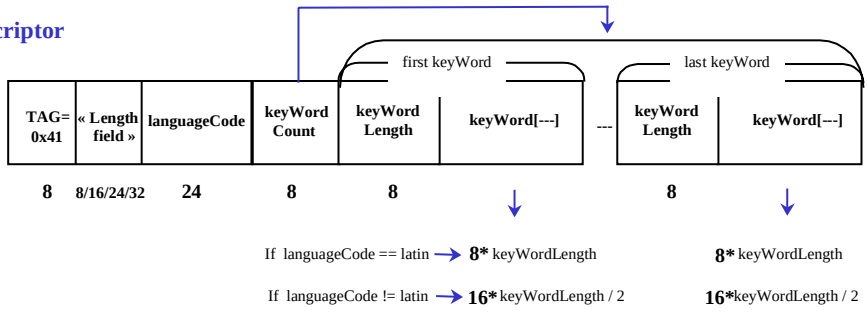
Hulu, LLC - Ex. 1024, Page 000373

J.6 OCI Descriptors

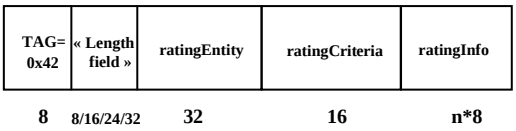
ContentClassificationDescriptor



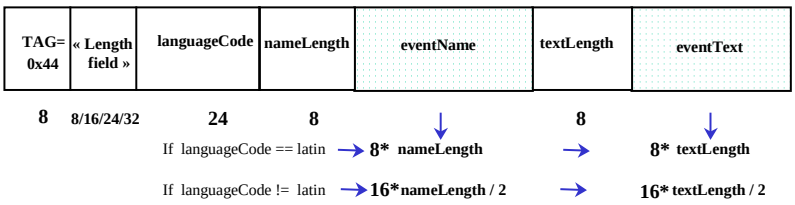
KeyWordDescriptor



RatingDescriptor

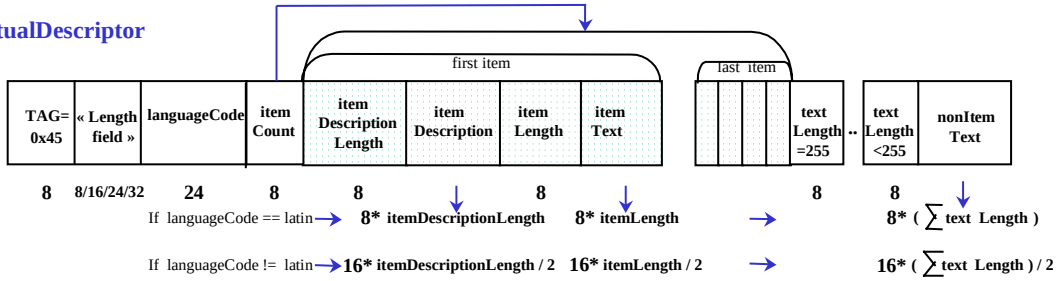


ShortTextualDescriptor

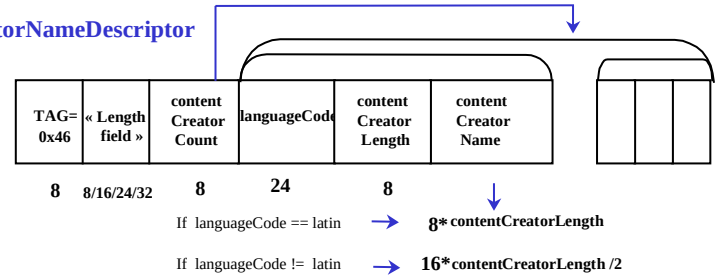


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000374

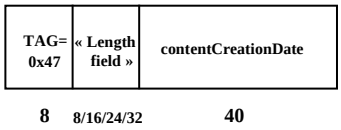
ExpandedTextualDescriptor



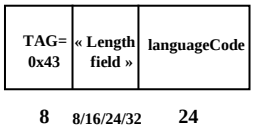
ContentCreatorNameDescriptor



ContentCreationDateDescriptor



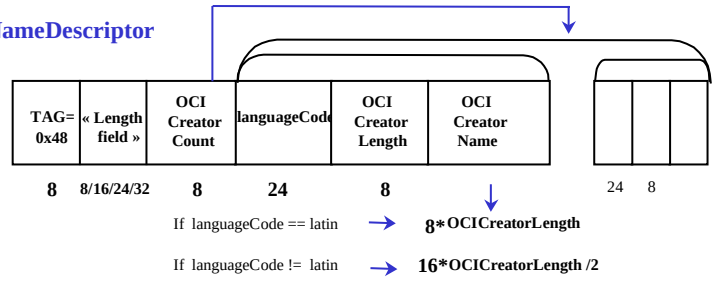
LanguageDescriptor



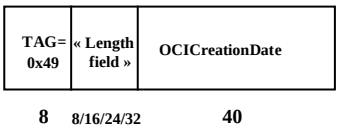
© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000375

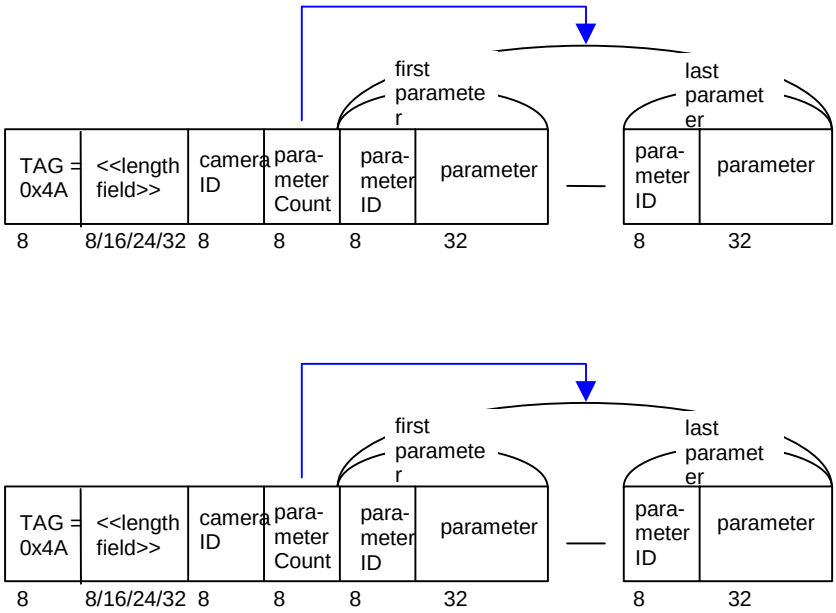
OCICreatorNameDescriptor



OCICreationDateDescriptor



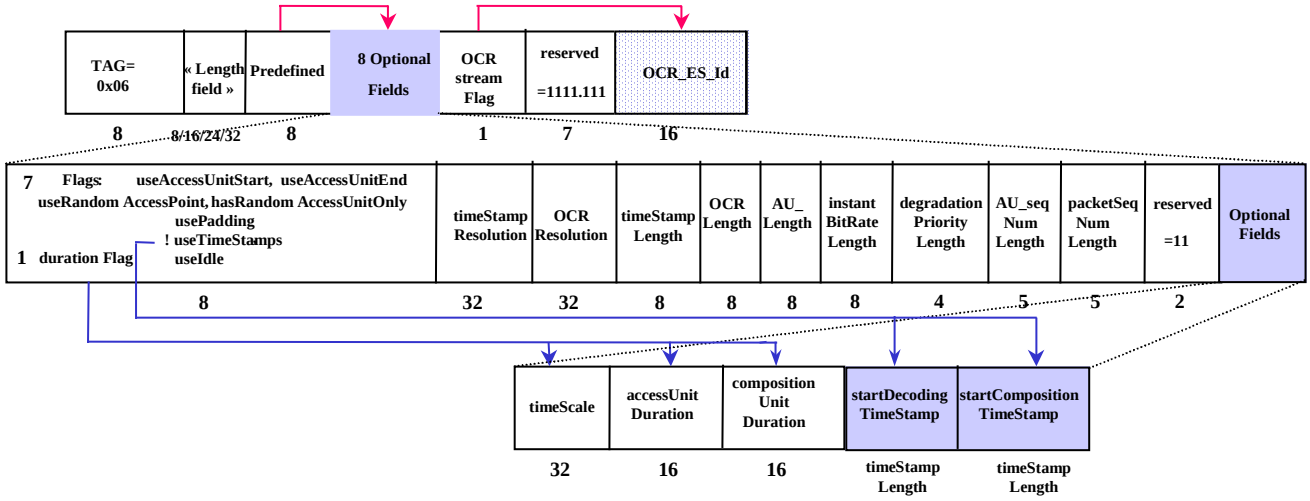
SmpteCameraPositionDescriptor

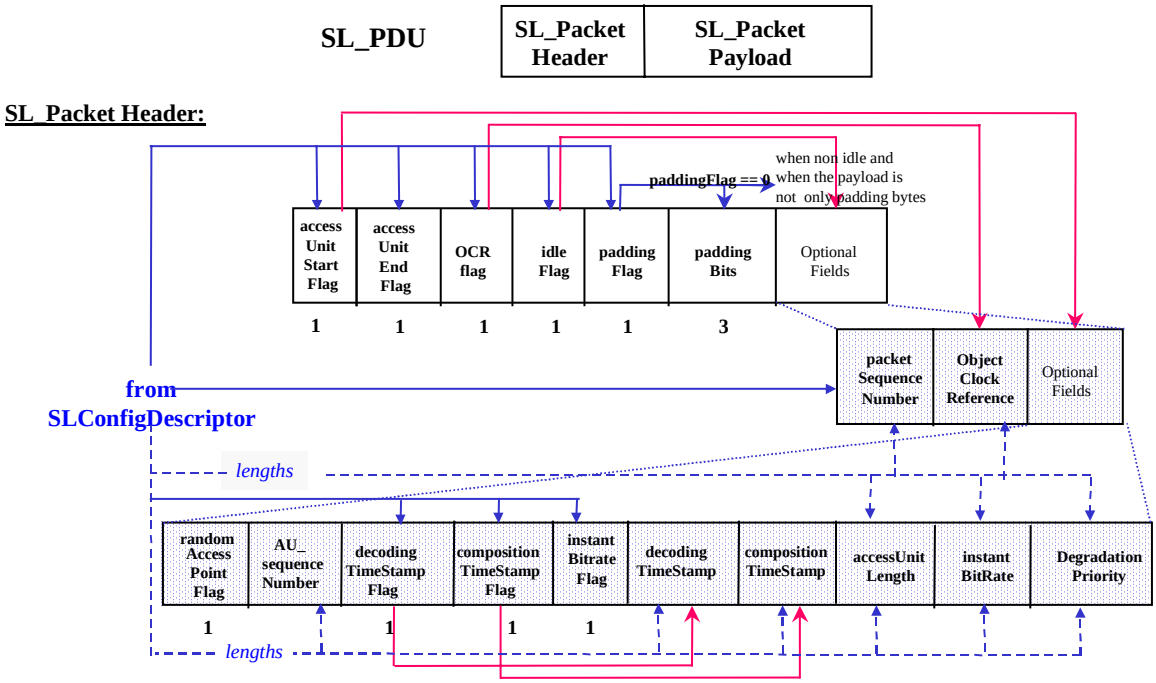


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000376

J.7 Sync layer configuration and syntax

SLConfigDescriptor





SL_Packet Payload:

according to padding flag and bits, it is either **Payload** or **Payload** padding bits or only **Padding bytes**

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000377

Annex K
(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 right 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.

K.1 Patent Statements for Version 1

The table summarises the formal patent statements received and indicates the parts of the standard to which the statement applies. (S: Systems, V: Visual, A: Audio, R: Reference Software, D: DMIF) The list includes all organisations that have submitted informal patent statements. However, if no "X" is present, no formal patent statement has yet been received from that organisation.

	Company	**S**	**V**	**A**	**R**	**D**
1.	Alcatel	x	x	x	x	x
2.	AT&T					
3.	BBC	x	x	x	x	x
4.	Bosch		x	x	x	
5.	British Telecommunications	x	x	x	x	x
6.	Canon	x	x	x	x	x
7.	CCETT	x	x	x	x	x
8.	Columbia University	x	x	x	x	x
9.	Creative	x		x	x	
10.	CSELT			x		
11.	DEmoGraFX		x		x	
12.	DirecTV	x	x	x		
13.	Dolby	x	x	x	x	x
14.	EPFL	x	x	x		
15.	ETRI	x	x	x	x	x
16.	FhG	x	x	x	x	x
17.	France Telecom	x	x	x	x	x
18.	Fujitsu Limited	x	x	x	x	x
19.	GC Technology Corporation	x	x	x		
20.	General Instrument		x		x	
21.	Hitachi	x	x	x	x	x
22.	Hyundai	x	x	x	x	x
23.	IBM					
24.	Institut für Rundfunktechnik	x	x	x		x
25.	InterTrust					
26.	JVC	x	x	x	x	x

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000378

27.	KDD Corporation	x	x			
28.	KPN	x	x	x	x	x
29.	LG Semicon					
30.	Lucent					
31.	Matsushita	x	x	x	x	x
32.	Microsoft	x	x	x	x	x
33.	MIT					
34.	Mitsubishi	x	x	x	x	
35.	Motorola		x		x	
36.	NEC Corporation	x	x	x	x	x
37.	NHK	x	x	x	x	x
38.	Nokia		x	x	x	
39.	NTT	x	x	x	x	x
40.	OKI	x	x	x	x	x
41.	Philips	x	x	x	x	x
42.	PictureTel Corporation		x		x	
43.	Rockwell	x	x	x	x	x
44.	Samsung	x	x	x		
45.	Sarnoff	x	x	x	x	x
46.	Scientific Atlanta	x	x	x	x	x
47.	Sharp	x	x	x	x	x
48.	Siemens	x	x	x		
49.	Sony	x	x	x	x	x
50.	Telenor	x	x	x	x	x
51.	Teltec DCU		x		x	
52.	Texas Instruments					
53.	Thomson	x	x	x		
54.	Toshiba		x			
55.	Unisearch Ltd.		x		x	
56.	Vector Vision		x			

K.2 Patent Statements for Version 2

The table summarises the patent statements received for Version 2 and indicates the parts of the Version 2 standard to which the statement applies. A Legend to interpret the table is given below.

Legend: The presence of a name of a company in the list below indicates that a patent statement has been received from that company

The presence of a cross indicates that the statement identifies the part of the MPEG-4 version 2 standard to which the statement applies

No cross in a line indicates that the statement does not identify which part of the standard the statement applies

	Company	**S**	**V**	**A**	**R**	**D**
1.	Apple	x			x	
2.	British Telecommunications					
3.	Bosch	x	x	x	x	x
4.	CCETT	x	x	x	x	x
5.	Columbia Innovation Enterprise	x	x	x	x	x
6.	DemoGraFX	x	x	x	x	x
7.	DirecTV	x	x	x		
8.	Dolby	x	x	x	x	x

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000379

9.	EPFL	x	x		x	
10.	France Telecom	x	x	x	x	x
11.	Fraunhofer			x	x	
12.	Fujitsu		x		x	
13.	Hitachi	x	x	x	x	x
14.	Hyundai	x	x	x	x	x
15.	IBM	x	x	x	x	x
16.	Intertrust					
17.	JVC	x	x	x	x	x
18.	KPN		x			
19.	Lucent					
20.	Matsushita Electric Industrial Co., Ltd.	x	x	x	x	x
21.	Microsoft	x	x	x	x	x
22.	Mitsubishi	x	x	x	x	x
23.	NEC	x	x	x	x	x
24.	NHK	x	x	x	x	x
25.	Nokia	x	x	x	x	x
26.	NTT		x			
27.	NTT Mobile Communication Networks			x		
28.	OKI	x	x		x	
29.	Optibase	x			x	
30.	Philips					
31.	Samsung	x	x	x	x	
32.	Sarnoff	x	x	x	x	x
33.	Sharp	x	x	x	x	x
34.	Siemens	x	x	x	x	x
35.	Sony	x	x	x	x	x
36.	Sun	x				
37.	Thomson	x	x	x	x	x
38.	Toshiba		x			

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000380

Annex L
(informative)

Elementary Stream Interface

The elementary stream interface (ESI) is a conceptual interface that specifies which data need to be exchanged between the entity that generates an elementary stream and the sync layer. Communication between the coding and sync layers cannot only include compressed media, but requires additional information such as time codes, length of access units, etc.

An implementation of ISO/IEC 14496-1, however, does not have to implement the elementary stream interface. It is possible to integrate parsing of the SL-packetized stream and media data decompression in one decoder entity. Note that even in this case the decoder receives a sequence of packets at its input through the DMIF Application Interface (see 10.3) rather than a data stream.

The interface to receive elementary stream data from the sync layer has a number of parameters that reflect the side information that has been retrieved while parsing the incoming SL-packetized stream:

ESI.receiveData (*ESdata, dataLength, idleFlag, objectClockReference, decodingTimeStamp, compositionTimeStamp, accessUnitStartFlag, randomAccessFlag, accessUnitEndFlag, accessUnitLength, degradationPriority, errorStatus*)

ESdata - a number of *dataLength* data bytes for this elementary stream

dataLength - the length in byte of *ESdata*

idleFlag – if set to one it indicates that this elementary stream will not produce further data for an undetermined period of time.

objectClockReference – contains a reading of the object time base valid for the point in time when the first byte of ESdata enters the decoder buffer.

decodingTimeStamp - the decoding time for the access unit to which this *ESdata* belongs

compositionTimeStamp - the composition time for the access unit to which this *ESdata* belongs

accessUnitStartFlag - indicates that the first byte of *ESdata* is the start of an access unit

randomAccessFlag - indicates that the first byte of *ESdata* is the start of an access unit allowing for random access

accessUnitEndFlag - indicates that the last byte of *ESdata* is the end of an access unit

accessUnitLength - the length of the access unit to which this Esdata belongs in byte

degradationPriority - indicates the degradation priority for this access unit

errorStatus - indicates whether *ESdata* is error free, possibly erroneous or whether data has been lost preceding the current *ESdata* bytes

A similar interface to send elementary stream data to the sync layer requires the following parameters that will subsequently be encoded on the sync layer:

ESI.sendData (*ESdata, dataLength, idleFlag, objectClockReference, decodingTimeStamp, compositionTimeStamp, accessUnitStartFlag, randomAccessFlag, accessUnitEndFlag, accessUnitLength, degradationPriority*)

ESdata - a number of *dataLength* data bytes for this elementary stream

dataLength - the length in byte of *ESdata*

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000381

idleFlag – if set to one it indicates that this elementary stream will not produce further data for an undetermined period of time.

objectClockReference – contains a reading of the object time base valid for the point in time when the first byte of ESdata enters the decoder buffer.

decodingTimeStamp - the decoding time for the access unit to which this *ESdata* belongs

compositionTimeStamp - the composition time for the access unit to which this *ESdata* belongs

accessUnitStartFlag - indicates that the first byte of *ESdata* is the start of an access unit

randomAccessFlag - indicates that the first byte of *ESdata* is the start of an access unit allowing for random access

accessUnitEndFlag - indicates that the last byte of *ESdata* is the end of an access unit

accessUnitLength - the length of the access unit to which this Esdata belongs in byte

degradationPriority - indicates the degradation priority for this access unit

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000382

Annex M
(Informative)

Definition of bodySceneGraph nodes

M.1 Introduction

This Annex includes the normative definitions of the nodes used in the **bodySceneGraph** field of the **BDP** node (see 9.4.2.17)

M.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.

M.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.

M.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.

M.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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000383

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           []
]
```

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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000384

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
]
```

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.

M.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 `joints`, `segments` and `views` 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        [ ]
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000385

```
    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.

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.

M.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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000386

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 14496-2:1999.

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.

M.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
  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"
      ...
    }
  ]
}
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000387

M.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 `Joints` are descended from either vl5 or sacroiliac. If those `Joints` are missing, lower-level `Joints` can be children of the `HumanoidRoot`.

M.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
```

M.4.3.4 Hierarchy

The complete hierarchy is as follows, with the segment names listed beside the `Joints` 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
  |             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
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000388

```
|       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
            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 `Joints`, and are stored in the `bodyDefinition` field of the `Humanoid` node.

The `l_sternoclavicular` and `r_sternoclavicular` `Joints` 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000389

A Joint node may contain 1-3 BAPs. The following table presents the joints, and associated BAPs and segment names.

Table 65 - 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
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

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000390

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 `Joints` may be omitted, such as most of the vertebrae, the midtarsal, and the acromioclavicular. The spinal `Joints` 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.

M.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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000391

Annex N
(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)

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.

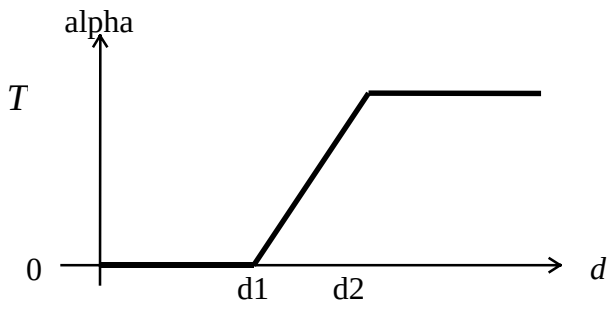


Figure 39 - **Alpha value as a function of distance measure**

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000392

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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000393

Annex O
(Informative)

Example implementation of spatial audio processing (perceptual approach)

O.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).

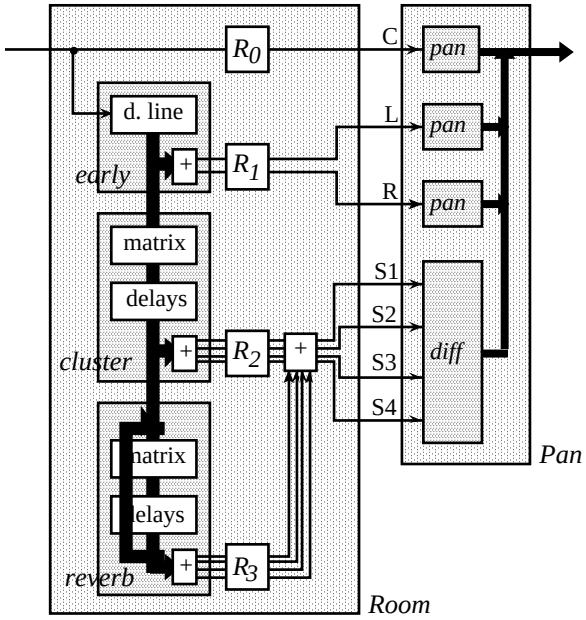


Figure 40 - Association of *Room* and *Pan* modules forming a *Spat* processor.

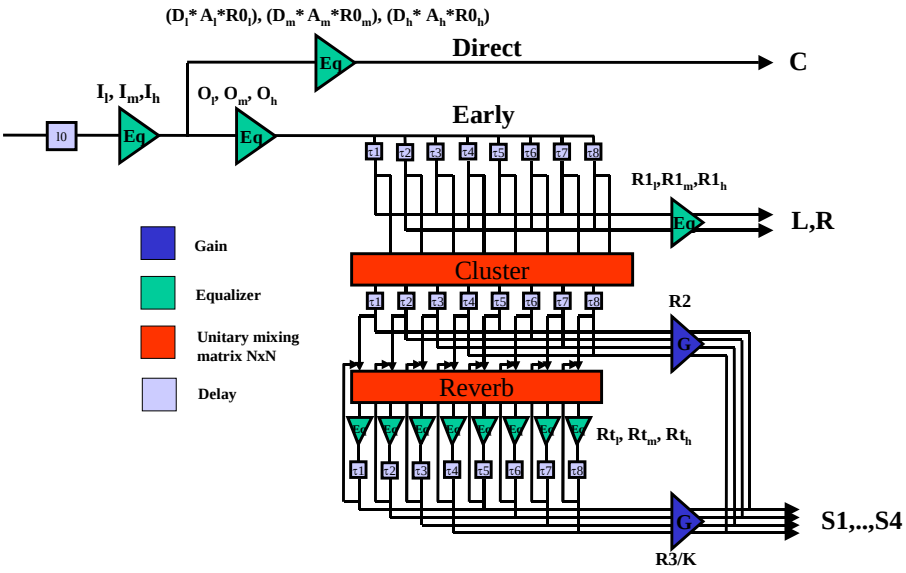
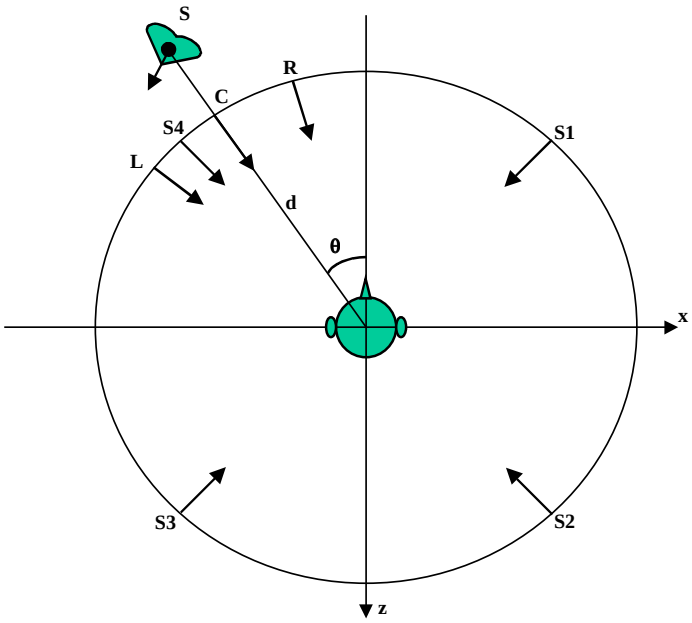


Figure 41 - Block diagram of the *Room* module

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000394



Figrue 42 - Directional rendering by the Pan module

O.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 :

$y(n)=b_0x(n)+b_1x(n-1)+b_2x(n-2)-a_1y(n-1)-a_2y(n-2)$.

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)

$g^r_{low} = g^r_{low}/g$

$g^r_{mid} = g^r_{mid}/g$

$g^r_{high} = g^r_{high}/g$

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000395

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)

$k_1 = g^r_{low} / g^r_{mid}$

$r_1 = \tan(\pi * f^r_{low}) / k_1^{0.5}$

$\alpha_1 = (r_1 - 1)/(r_1 + 1)$

$\beta_1 = (k_1 * r_1 - 1)/(k_1 * r_1 + 1)$

$\chi_1 = (k_1 * r_1 - 1)/(r_1 + 1)$

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)

$k = g^r_{mid} * g$

$b_o = \chi_1 * \chi_2 * k$

$b_1 = (\beta_1 + \beta_2) * b_o$

$b_2 = (\beta_1 * \beta_2) * b_o$

$a_1 = \alpha_1 + \alpha_2$

$a_2 = \alpha_1 * \alpha_2$

O.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.

O.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.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000396

O.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 exact values to be used are given in 1.2.6.6.6.

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 :

$[Cscramb_i]_{i=0,...,7}=H_{8x8}[early_i]_{,i=0,...,7}$

The scrambled signals are independently delayed :

$Cscramb^d[i](t)=Cscramb[i](t-\tau_i)$, 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 $R2_3$

O.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:

$Rscramb^d[i](t)=Rscramb[i](t-\tau_i)$, i=0,...,7

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000397

The values of the τ_i are randomly distributed between 60 and 140 ms approximately (for a large room). The exact values to be used are given in 1.2.6.6.6.

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 complete diffuse field is simply calculated from the cluster and the late reverberation fields signals as follows :

$S1= R2_0 + R3_0$

$S2= R2_1 + R3_1$

$S3= R2_2 + R3_2$

$S4= R2_3 + R3_3$

O.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 66 - 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).

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000398

O.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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000399

Annex P
(informative)

Upstream walkthrough

P.1 Introduction

Upstream messages from a client terminal to the server terminal are categorized in two types, application specific command messages and media stream specific messages. Application specific command messages are general messages applied to a set of different media streams, for example, stream control messages. These messages may be defined based on the BIFS ServerCommand node. Media stream specific messages are used to establish communication between a specific media stream decoder and its encoder. This may be used, for example, to control the encoder remotely from the client terminal side as a result of the decoding process or user interaction. The syntax and semantics of media stream specific messages are defined in the relevant part of the standard. For example, the syntax and semantics of messages for the visual NEWPRED tool are defined in IS 14496-2, defining the Visual tools of this specification.

The need for an upstream channel is signaled to the client terminal by supplying an appropriate elementary stream descriptor declaring the parameters for that stream. The client terminal opens this upstream channel in a similar manner as it opens the downstream channels. The entities (e.g. media encoders & decoders) that are connected through an upstream channel are known from the parameters in its elementary stream descriptor and from the association of the elementary stream descriptor to a specific object descriptor.

Packetization of upstream messages for transmission and synchronization with downstream channel data is done by the synchronization layer. The configuration of the SL packet header for upstreams may be selected as appropriate. All messages that are related to a single point in time should be packetized into a single access unit.

P.2 Configuration

An upstream can be associated to a single downstream or a group of downstreams. The scope of the upstream is defined by the stream type of the downstream to which the upstream is associated. When the upstream is associated to a single downstream it carries messages about the downstream it is associated to. If the upstream should carry messages related to a group of downstreams, its elementary stream descriptor is associated to the ObjectDescriptorStream containing object descriptors or the SceneDescriptionStream describing the scene, as specified in 8.7.1.5.2.

In the case that the upstream is attached to the ObjectDescriptorStream, only the object descriptors grouped together for this single upstream would be carried by it. The other object descriptors outside the scope of this upstream would be carried by other ObjectDescriptorStreams. This implies that the object descriptors requiring a single upstream should be carried separately from the other object descriptors. If the upstream depends on a SceneDescriptionStream, all the objects inside the scene would get the upstream messages from this upstream.

Detailed configuration rules for each case are as described below.

P.2.1 Upstream for single ES

In this case the upstream is attached to a single independent ES and will carry media specific information valid for a single downstream it is dependent on. Because only one of the independent elementary streams defined in the same OD can be selected for use in the scene, the upstream is not related to the ES itself but rather to the object represented by this OD.

The ObjectDescriptor has one or more additional ES_Descriptors defining upstream configuration for each ES which needs a backchannel.

ES_Descriptor of upstream shall be defined as follows

streamDependenceFlag shall be set to ‘1’ to indicate this stream depends on a downstream.

dependsOn_ES_ID shall be set to the ES_ID value of the downstream.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000400

DecoderConfigDescriptor in ES_Descriptor of upstream shall be defined as follows.

objectTypeIndication and streamType shall be set to the same value of the downstream

upStream flag shall be set to '1' to indicate this is a backchannel stream.

bufferSizeDB, maxBitrate, avgBitrate and DecoderSpecificInfor shall be set appropriately.

P.2.2 Upstream for a group of ESs

In this case the upstream is attached to an ObjectDescriptorStream or a SceneDescriptionStream to be used as an upstream for a group of elementary streams. The basic configuration rules for the ObjectDescriptor are the same as in the case of upstream for a single ES. The scope and type of messages carried by the upstream is decided by the following rules.

If an upstream is configured to be dependent on a certain ObjectDescriptorStream and its streamType is either VisualStream or AudioStream, it carries media stream specific information that may relate to more than one of the downstreams that are described by the ObjectDescriptors transmitted within the ObjectDescriptorStream upon which the upstream depends. All decoders for streams with matching streamType within that set of streams may use the upstream channel to send messages.

If an upstream is configured to be dependent on a certain SceneDescriptionStream and its streamType is either VisualStream or AudioStream, it carries media specific information for downstreams in the whole scene as described by the SceneDescriptionStream upon which the upstream depends. All decoders for streams with matching streamType within that set of streams may use the upstream channel to send messages.

If an upstream is configured to be dependent on a certain SceneDescriptionStream and its streamType is SceneDescriptionStream, it will carry messages related to the BIFS scene or to application signaling (e.g. based on the ServerCommand specification).

P.3 Content access procedure with DAI

When the receiving terminal receives a DecoderConfigDescriptor whose upStream flag is set to '1', it opens a logical channel for the upstream ES by setting the 'direction' field of the DA_ChannelAdd primitive to UPSTREAM. Other procedures and rules for accessing and managing content at the client terminal are basically the same as for the case of downstream. The syntax and semantics of upstream messages, defining their functionality and the expected interaction between encoder and decoder, are defined in the appropriate part of ISO/IEC 14496. Messages related to streams of streamType SceneDescriptionStream and ObjectDescriptorStream are defined in this part of the specification. Concerning upstream management at the sending terminal, this standard does not normatively specify any behavioral procedures or rules.

P.4 Example

This section describes an example of the setup and the usage of MPEG-4 upstreams, according to the rules described in the above sections.

P.4.1 Example scene having objects with upstream

Figure P-1 shows a simple scene with 3 objects (1 natural video and 2 SNHC objects) for which information is gathered through different upstreams:

- ServerCommand upstream is used to control the animation (start, stop, …) of **all** objects (natural video and SNHC) in the scene (possibly also audio objects).
- NewPred upstream is used for error correction of a **single natural video** object.
- SNHC_QoS upstream conveys information of the client terminal w.r.t. its decoding and rendering capabilities for **all 3D (SNHC)** objects.

The example scene of Figure P-1 is described through the object descriptor Full_Scene, which points to different streams:

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000401

```
InitialObjectDescriptor Full_Scene{
   bit(10) Full_Scene_ID (=OD_ID);
   bit(1) 0 (=URL_Flag);
   bit(1) 1 (=includeInlineProfileLevelFlag);
   const bit(4) reserved=0b1111;
   bit(8) ODProfileLevelIndication;
   bit(8) sceneProfileLevelIndication;
   bit(8) audioProfileLevelIndication;
   bit(8) visualProfileLevelIndication;
   bit(8) graphicsProfileLevelIndication;
   ES_Descriptor SceneDescriptionStream_Scene_1_down;
   ES_Descriptor SceneDescriptionStream_Scene_1_up;
   ES_Descriptor ObjectDescriptorStream_Scene_1_down;
   ES_Descriptor ObjectDescriptorStream_Scene_1_up;
}
```

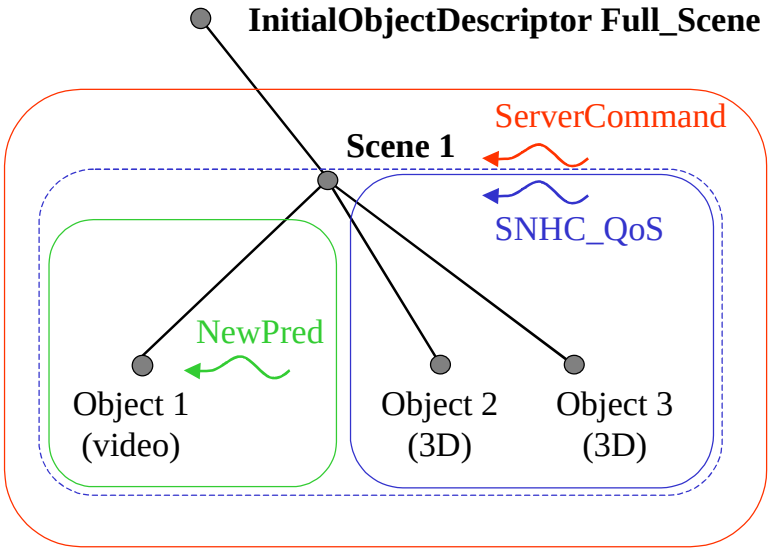


Figure 43 - Backchannel information transport in a simple audio-visual scene

P.4.2 Stream configuration

Graphical summaries of the stream configuration for the example scene shown in Figure 43 are given in Figure 44 to Figure 46. In those figures configuration rules of the important fields and stream dependencies are described in detail.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000402

```
InitialObjectDescriptor Full_Scene{
ES_Descriptor SceneDescriptionStream_Scene_1_down;
ES_Descriptor SceneDescriptionStream_Scene_1_up;
ES_Descriptor ObjectDescriptorStream_Scene_1_down;
ES_Descriptor ObjectDescriptorStream_Scene_1_up;
}

ES_Descriptor SceneDescriptionStream_Scene_1_down {
          bit(16) SceneDescriptionStream_Scene_1_down_ID (=ES_ID);
          bit(1) 0 (=streamDependenceFlag);
          DecoderConfigDescriptor dec_config_SceneDescriptionStream_Scene_1_down;
}

DecoderConfigDescriptor dec_config_SceneDescriptionStream_Scene_1_down {
          bit(8) 0x01 (=objectTypeIndication);    // System ISO/IEC 14496-1
          bit(6) 0x03 (=streamType);              // SceneDescriptionStream
          bit(1) 0 (=down/upstream);              // Downstream
}

ES_Descriptor SceneDescriptionStream_Scene_1_up {
          bit(16) SceneDescriptionStream_Scene_1_up_ID (=ES_ID);
          bit(1) 1 (=streamDependenceFlag);
          bit(16) SceneDescriptionStream_Scene_1_down_ID;
          DecoderConfigDescriptor dec_config_SceneDescriptionStream_Scene_1_up;
}

DecoderConfigDescriptor dec_config_SceneDescriptionStream_Scene_1_up {
          bit(8) 0x01 (=objectTypeIndication);    // System ISO/IEC 14496-1
          bit(6) 0x03 (=streamType);              // SceneDescriptionStream
                                                  //    => conveys ServerCommand
          bit(1) 1 (=down/upstream);              // Upstream
}
```

2 1 3 4

Figure 44 - Syntax for SceneDescription streams

```
InitialObjectDescriptor Full_Scene{
ES_Descriptor SceneDescriptionStream_Scene_1_down;
ES_Descriptor SceneDescriptionStream_Scene_1_up;
ES_Descriptor ObjectDescriptorStream_Scene_1_down;
ES_Descriptor ObjectDescriptorStream_Scene_1_up;
}

ES_Descriptor ObjectDescriptorStream_Scene_1_down {
          bit(16) ObjectDescriptorStream_Scene_1_down_ID (=ES_ID);
          bit(1) 0 (=streamDependenceFlag);
          DecoderConfigDescriptor dec_config_ObjectDescriptorStream_Scene_1_down;
}

DecoderConfigDescriptor dec_config_ObjectDescriptorStream_Scene_1_down {
          bit(8) 0x01 (=objectTypeIndication);    // System ISO/IEC 14496-1
          bit(6) 0x01 (=streamType);              // ObjectDescriptorStream
          bit(1) 0 (=down/upstream);              // Downstream
}

ES_Descriptor ObjectDescriptorStream_Scene_1_up {
          bit(16) ObjectDescriptorStream_Scene_1_up_ID (=ES_ID);
          bit(1) 1 (=streamDependenceFlag);
          bit(16) ObjectDescriptorStream_Scene_1_down_ID;
          DecoderConfigDescriptor dec_config_ObjectDescriptorStream_Scene_1_up;
}

DecoderConfigDescriptor dec_config_ObjectDescriptorStream_Scene_1_up {
          bit(8) 0x20 (=objectTypeIndication);    // Visual ISO/IEC 14496-2
          bit(6) 0x01 (=streamType);              // ObjectDescriptorStream
                                                  //    => conveys SNHC_QoS
          bit(1) 1 (=down/upstream);              // Upstream
}
```

6 5 7 8

Figure 45 - Syntax for ObjectDescriptor streams

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000403

In Figure 44, dependencies and configurations of two SceneDescriptionStreams are shown. Upstream SceneDescriptionStream_Scene_1_up is dependent on downstream SceneDescriptionStream_Scene_1_down (see arrows 1 and 2 in figure). Its `streamType` is set to `SceneDescriptionStream` since it will carry ServerCommand messages (see arrows 3 and 4 in figure).

In Figure 45, dependencies and configurations of two ObjectDescriptionStreams are shown. Upstream ObjectDescriptionStream_Scene_1_up is dependent on downstream ObjectDescriptionStream_Scene_1_down (see arrows 5 and 6 in figure). Its `streamType` is set to `VisualStream` since it will carry SNHC_QoS messages for object 2 and object 3 in this example (see arrows 7 and 8 in figure). ObjectDescriptorStream_Scene_1_up conveys SNHC_QoS information that is related to all SNHC objects of the underlying group of objects (possibly single object). This SNHC_QoS information is basically attached to all Visual objects (see dashed box in Figure P-1), but the definition of SNHC_QoS constrains the scope of application to SNHC objects only. Whether the Visual objects are of type SNHC or Natural cannot be determined at the system level : it is determined at the Visual syntax level, by the visual_object_type, in accordance to table 6-5 of ISO/IEC14496-2.

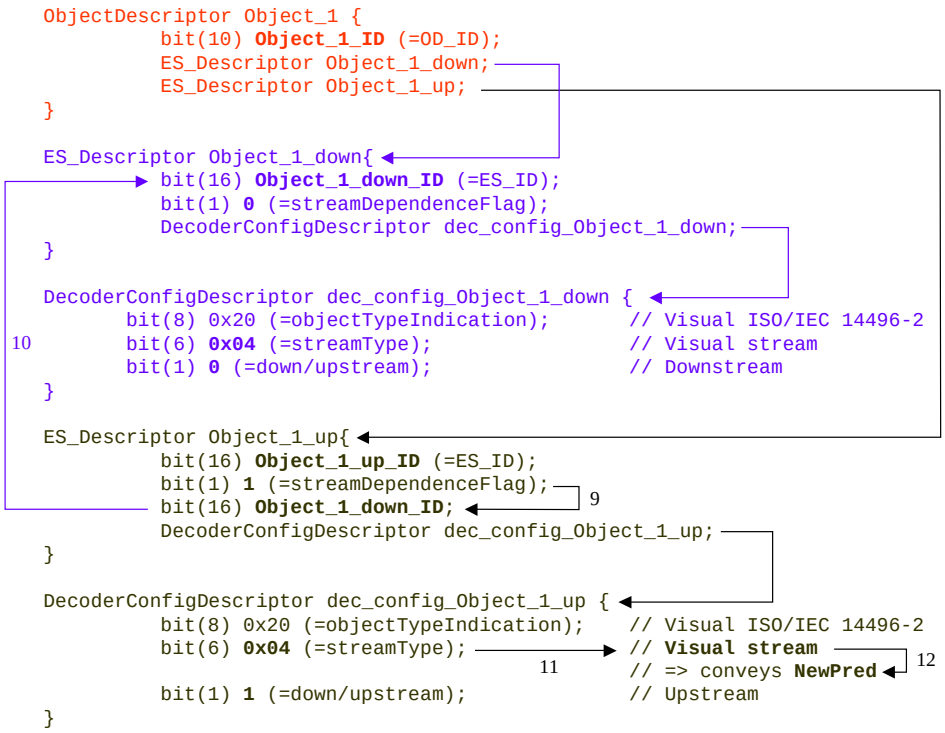


Figure 46 - Syntax for Object_1 streams

In Figure 46, dependencies and configurations of two Elementary Streams are shown. Upstream Object_1_up is dependent on downstream Object_1_down (see arrows 9 and 10 in figure). Its `streamType` is set to `VisualStream` since it will carry NewPred messages for object 1 in this example (see arrows 11 and 12 in figure). Object_1_up conveys NewPred information for the corresponding natural video object. The definition of NewPred automatically constrains its application to single natural video objects, i.e. the behavior of the server-client system is undefined if a NewPred command is associated to a group of objects and/or a single non-natural video object.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000404

Annex Q
(Informative)

Layout of Media Data

The MP4 format provides a great deal of flexibility in how the media data is physically arranged within a file. This flexibility is desirable and useful. However, it allows media layouts to be created that may be inefficient for play back on a given device. It should also be noted, however, that a media layout which is inefficient for a given device may be very efficient for another. Therefore, it is not the intention of this discussion to define a given type of media layout as wrong. Rather, the intention is to define some common uses of MP4 files and describe the media layout in these circumstances.

An MP4 file can reference media data stored in a number of files, including the MP4 file itself. If an MP4 file only references media data contained within itself, the MP4 file is said to be "self-contained".

An MP4 file can reference media data stored in files that are not MP4 files. This is because the MP4 format references media within a URL by file offset, rather than by a data structuring mechanism of a particular file format. This allows an MP4 file to refer to data stored in any container format.

It is often convenient to store a single elementary stream per file, for example when encoding content. It is also useful for content reuse — to reuse an elementary stream, it is not necessary to extract it from a larger, possibly multiplexed file. Because MP4 can reference media stored in any file, it is not required that elementary stream files be stored in the MP4 format. However, this is recommended. Putting the elementary streams in a MP4 file has several advantages, particularly in enabling interchange of the content between different tools (since MP4 is the only normative file format for MPEG-4 media). Further, the MP4 format adds very little overhead to the elementary stream — as little as a few hundred bytes in many cases — so there is no great penalty in storage space. It may be useful to give a name to these types of files — MP4 files that contain a single elementary stream — so that they can be consistently described.

One of the issues facing any device (server, local workstation) that is attempting to play back an MP4 file in real time is the number of file seeks that must be performed. It is possible to arrange the data in an MP4 file to minimize, and potentially eliminate, any seeks during the course of normal playback. (Of course random access and other kinds of interactivity will require seeks). This is accomplished by interleaving the elementary stream data in the MP4 file in such a way that the layout of the media in the file corresponds to the order in which the media data will be required. It is expected that most servers, for example, will stream MPEG-4 media using the facilities of the hint track. Let's take a scenario where the MP4 file contains a single hint track that reference an audio and a visual elementary stream. In order to eliminate all seeks, the hint track media must be interleaved with the audio and visual stream data. Furthermore, because the hint track sample must always be read before the audio and/or visual media that it references, the hint track samples must always immediately precede the samples they reference. Here's a simple illustration of the ordering of data (time and file offset increasing from left to right).

H0 A0 H1 V1 H2 V2 H3 A1 H4 A2 V3 H5 V4

Note that when a single hint sample references to multiple pieces of elementary stream data, those pieces of media data must be occur in the order that they are referenced.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000405

Annex R
(Informative)

Random Access

Seeking with a MP4 file is accomplished primarily by using the subatoms contained in the Sample Table atom. If an edit list is present, it must also be consulted. When asked to seek a given track to a time T, where T is in the time scale of the movie header atom, you may perform the following operations:

If the track contains an edit list, determine which edit contains the time T by iterating over the edits. The start time of the edit in the Movie time scale must then be subtracted from the time T to generate T', the duration into the edit in the Movie time scale. T' is next converted to the timescale of the track's media to generate T''. Finally, the time in the media scale to use is calculated by adding the media start time of the edit to T''.

The time-to-sample atom for a track indicates what times are associated with which sample for that track. Use this track to find the first sample prior to the given time.

The sample that was located in step one may not be a random access point. Locating the nearest random access point requires consulting two atoms. The sync sample table indicates which samples are in fact random access points. Using this table, you can locate which is the first sync sample prior to the specified time. The absence of the sync sample table indicates that all samples are synchronization points, and makes this problem easy. The shadow sync atom gives the opportunity for a content author to provide samples that are not delivered in the normal course of delivery, but which can be inserted to provide additional random access points. This improves random access without impacting bitrate during normal delivery. This atom maps samples that are not random access points to alternate samples that are. You should also consult this table if present to find the first shadow sync sample prior to the sample in question. Having consulted the sync sample table and the shadow sync table, you probably wish to seek to whichever resultant sample is closest to, but prior to, the sample found in step 1.

At this point you know the sample that will be used for random access. Use the sample-to-chunk table to determine in which chunk this sample is located.

Knowing which chunk contained the sample in question, use the chunk offset atom to figure out where that chunk begins.

Starting from this offset, you can use the information contained in the sample-to-chunk atom and the sample size atom to figure out where within this chunk the sample in question is located. This is the desired information.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000406

Annex S
(Informative)

Starting the Java Virtual Machine

The objective of this annex 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.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000407

Annex T
(Informative)

Examples of MPEG-J API usage

This annex will include code snippets on the usage of the different categories of MPEG-J APIs. The purpose of this annex is, however, not to show typical applications. The scope of this annex is to enable a user to quickly get started on using MPEG-J APIs.

T.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 {
            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 ) { }

    }
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000408

```
	public void stop(){}
	public void destroy(){}
}
```

T.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;
			notify();
		}
	}
}
```

T.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.
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000409

```
 */
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);
      }
}
```

T.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 {
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000410

```
            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);
      }
}
```

T.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.
 */
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;
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000411

```
	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 {
	float[] m_value;

	MySFVec3f(float x, float y, float z) {
		m_value = new float[] { x, y, z };
	}

	public float[] getSFVec3fValue() {
		return m_value;
	}
}
```

T.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;
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000412

```
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) { }

  // 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
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000413

```
        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(){}

}
```

T.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();
      }
      public DecoderEventHandler( ) {
            super();
      }
}
```

T.2.1 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();
    }
}
```

T.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
{
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000414

```
      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(){}
}
```

T.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;
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000415

```
    }
    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) { }

    // sections matching this SectionFilter object will stop.
      ssFilter.stopFiltering();
  }
  public void init() {}
  public void stop(){}
  public void destroy(){}
}
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000416

Annex U
(Normative)

MPEG-J APIs Listing (HTML)

This is a normative annex containing the javadocs of the MPEG-J APIs in the HTML format. This is included in the electronic attachment to this Amendment.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000417

Annex V
(Normative)

MPEG-J APIs Listing

This is a normative annex containing the javadocs of the MPEG-J APIs. The API documentation was automatically generated from Java sources of the specified MPEG-J APIs.

Package Summary	
Packages	
Package org.iso.mpeg.mpegj	Access to MpegjTerminal which sets the namescope and provides the managers for all the other categories (ResourceManager, SceneManager, etc.)
Package org.iso.mpeg.mpegj.decoder	Access and control to the decoders used to decode the audio-visual objects.
Package org.iso.mpeg.mpegj.net	Access and control of the network components of the MPEG-4 player.
Package org.iso.mpeg.mpegj.resource	Centralized facility for managing system resources Access to the static and dynamic capabilities of the terminal.
Package org.iso.mpeg.mpegj.scene	Means by which MPEG-J applications access and manipulate the scene graph

V.1 package org.iso.mpeg.mpegj

V.1.1 Description

Class Summary	
Interfaces	
DecoderConfigDescriptor	interface for DecoderConfigDescriptor
ObjectDescriptor	interface for ObjectDescriptor
MPEGlet	This the interface that has to be implemented by a remote MPEG-J Application.
ESDescriptor	interface for ESDescriptor
Classes	
MpegjTerminal	MPEG-J Terminal class.
Exceptions	
MPEGJException	The class MPEGJException is a subclass of java.lang.Exception and is the parent of all MPEG-J Exceptions.
NetworkManagerNotFoundException	
SceneManagerNotFoundException	

V.1.2 org.iso.mpeg.mpegj.DecoderConfigDescriptor

V.1.2.1 Syntax

```
public interface DecoderConfigDescriptor
```

Description

interface for DecoderConfigDescriptor

Member Summary	
Methods	
int	getStreamType()
int	getavgBitRate()
int	getBuffersizeDB()
int	getmaxBitrate()
int	getObjectTypeIndication()

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000418

V.1.2.2 Methods

getavgBitRate()
`public int getavgBitRate()`

Returns average bitrate in bits per second of this elementary stream

Returns:
average bit rate

getBuffersizeDB()
`public int getBuffersizeDB()`

Returns the size of the decoding buffer for this elementary stream in byte

Returns:
size of the buffer

getmaxBitrate()
`public int getmaxBitrate()`

Returns the maximum bitrate in bits/second of this elementary stream

Returns:
maximum bitrate

getObjectTypeIndication()
`public int getObjectTypeIndication()`

Returns the indication of the object or scene description type

Returns:
value representing the object or scene description type that needs to be supported by the decoder for this elementary stream

getStreamType()
`public int getStreamType()`

Returns the type of this elementary stream

Returns:
stream type value

V.1.3 org.iso.mpeg.mpegj.ESDescriptor

V.1.3.1 Syntax

```
public interface ESDescriptor
```

V.1.3.2 Description

interface for ESDescriptor

Member Summary	
Methods	
int	getESID()
boolean	isStreamDependent()
int	getdependsOn_ESID()
int	getStreamPriority()
DecoderConfigDescriptor	getDecoderConfigDescriptor()

V.1.3.3 Methods

getDecoderConfigDescriptor()
`public org.iso.mpeg.mpegj.DecoderConfigDescriptor getDecoderConfigDescriptor()`

Returns the decoder config Descriptor

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000419

Returns: DecoderConfigDescriptor

getdependsOn_ESID()
`public int getdependsOn_ESID()`

Returns the ESID of another elementary stream on which this elementary stream depends.

Returns:
ESID or -1 if the stream is not dependent on another

getESID()
`public int getESID()`

Returns the ID associated with this ESdescriptor

Returns:
ESID

getStreamPriority()
`public int getStreamPriority()`

Returns the relative measure for the priority of this stream. An elementary stream with higher stream priority is more important than one with a lower stream priority.

Returns:
priority

isStreamDependent()
`public boolean isStreamDependent()`

Returns whether this elementary stream depends another elementary stream

Returns:
true or false

V.1.4 org.iso.mpeg.mpegj.MPEGJException

V.1.4.1 Syntax

```
public class MPEGJException extends java.lang.Exception
```

java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--**org.iso.mpeg.mpegj.MPEGJException**

Direct Known Subclasses:
`org.iso.mpeg.mpegj.net.AccessLayerException`,
`org.iso.mpeg.mpegj.scene.BadNodeException`,
`org.iso.mpeg.mpegj.scene.BadParameterException`,
`org.iso.mpeg.mpegj.resource.CapabilityManagerNotFoundException`,
`org.iso.mpeg.mpegj.resource.IllegalDecoderMediaEventsException`,
`org.iso.mpeg.mpegj.resource.IllegalRendererMediaEventsException`,
`org.iso.mpeg.mpegj.resource.IllegalStreamMediaEventsException`,
`org.iso.mpeg.mpegj.scene.InvalidNodeException`,
`org.iso.mpeg.mpegj.scene.InvalidSceneException`,
`org.iso.mpeg.mpegj.decoder.MediaDecoderException`, `org.iso.mpeg.mpegj.resource.RendererNotFoundException`, `org.iso.mpeg.mpegj.resource.ResourceManagerNotFoundException`

All Implemented Interfaces:
java.io.Serializable

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000420

V.1.4.2 Description

The class MPEGJException is a subclass of java.lang.Exception and is the parent of all MPEG-J Exceptions.

Member Summary	
Constructors	
	MPEGJException() MPEGJException(String)

V.1.4.3 Constructors

MPEGJException()
`public MPEGJException()`

Constructs an MPEGJException with no specified detail message.

MPEGJException(String)
`public MPEGJException(java.lang.String)`

Constructs an MPEGJException with a detailed message.

V.1.5 org.iso.mpeg.mpegj.MpegjTerminal

V.1.5.1 Syntax

```
public class MpegjTerminal extends java.lang.Object
```

java.lang.Object
 |
 +--**org.iso.mpeg.mpegj.MpegjTerminal**

V.1.5.2 Description

MPEG-J Terminal class. This class provides the information about the managers that are implemented in the terminal. Each applet or application instantiates a new environment once it is loaded.

Member Summary	
Constructors	
	MpegjTerminal() MpegjTerminal(MPEGlet)
Methods	
ResourceManager	getResourceManager()
SceneManager	getSceneManager()
NetworkManager	getNetworkManager()
ObjectDescriptor	getinitOD()
Vector	getODs(int)

V.1.5.3 Constructors

MpegjTerminal()
`public MpegjTerminal()`

Constructor for the MpegjTerminal. This constructor may only be used by local applications.

Throws:
`SecurityException` - Thrown if this method is called by a non-local application.

MpegjTerminal(MPEGlet)
`public MpegjTerminal(org.iso.mpeg.mpegj.MPEGlet)`

Constructor for the MpegjTerminal. This constructor may be called by an MPEGlet to indicate the name scope in which this MPEG-J Terminal object should operate.

Parameters:
`mpeglet` - the MPEGlet that identifies the name scope to use.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000421

Throws:
`SecurityException` - Thrown if this method is called by an unrecognized caller.

V.1.5.4 Methods

getinitOD()
`public org.iso.mpeg.mpegj.ObjectDescriptor getinitOD()`

Returns initOD associated with the scene

Returns:
ObjectDescriptor

getNetworkManager()
`public org.iso.mpeg.mpegj.net.NetworkManager getNetworkManager()`

Obtain the NetworkManager used to get all the information related to a DMIF session and to enable/disable DMIF channels.

Returns:
the NetworkManager if available.

Throws:
`org.iso.mpeg.mpegj.NetworkManagerNotFoundException` - Thrown when the NetworkManager is not available.

getODs(int)
`public java.util.Vector getODs(int)`

Returns the vector of ODs associated to that scene that were received by the OD stream.

Parameters:
`int` - initOD

Returns:
vector of Object Descriptors

getResourceManager()
`public org.iso.mpeg.mpegj.resource.ResourceManager getResourceManager()`

Obtain the ResourceManager.

Returns:
the ResourceManager if available.

Throws:
`org.iso.mpeg.mpegj.resource.ResourceManagerNotFoundException` - Thrown when the ResourceManager is not available.

getSceneManager()
`public org.iso.mpeg.mpegj.scene.SceneManager getSceneManager()`

Obtain the SceneManager.

Returns:
the SceneManager if available.

Throws:
`org.iso.mpeg.mpegj.SceneManagerNotFoundException` - Thrown when the SceneManager is not available.

V.1.6 org.iso.mpeg.mpegj.MPEGlet

V.1.6.1 Syntax

```
public interface MPEGlet extends java.lang.Runnable
```

All Superinterfaces:
java.lang.Runnable

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000422

V.1.6.2 Description

This the interface that has to be implemented by a remote MPEG-J Application. This methods in this interface are very similar syntactically and semantically to the Applet Class, except that this is an interface and it extends the Runnable interface, so that it can be executed as separate thread. Note that there is no start method. This is because run() of the Runnable interface is used instead of start

Member Summary	
Methods	
void	init()
void	stop()
void	destroy()

V.1.6.3 Methods

destroy()
```
public void destroy()
```
Called by the MPEG-J player to inform the MPEGlet that is beinf reclaimed and that is should destroy any resources it has allocated. The stop method will always be called before destroy. Any operations before it is destroyed should go in here.

init()
```
public void init()
```
Called by the MPEG-J player to inform the MPEGlet that it has been loaded into the system. This is always called before the first time the MPEGlet is run as a separate by calling the run() method. Any initialization goes here.

stop()
```
public void stop()
```
Called by the MPEG-J player to inform the MPEGlet that it should stop its execution. This always called before destroy() is called. This can also be called anytime the execution of the MPEGlet has to be stopped. Any operations before it is stopped should go in here.

V.1.7 org.iso.mpeg.mpegj.NetworkManagerNotFoundException

V.1.7.1 Syntax

```
public class NetworkManagerNotFoundException extends java.lang.Exception
```

java.lang.Object
 |
 +--java.lang.Throwable
  |
  +--java.lang.Exception
  |
  +--**org.iso.mpeg.mpegj.NetworkManagerNotFoundException**

All Implemented Interfaces:
java.io.Serializable

V.1.7.2 Description

Member Summary	
Constructors	
	NetworkManagerNotFoundException()

V.1.7.3 Constructors

NetworkManagerNotFoundException()
```
public  NetworkManagerNotFoundException()
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000423

V.1.8 org.iso.mpeg.mpegj.ObjectDescriptor

V.1.9 Syntax

```
public interface ObjectDescriptor
```

V.1.9.1 Description

interface for ObjectDescriptor

Member Summary	
Methods	
int	getObjectDescriptorID()
Vector	getESDescriptors()
ESDescriptor	getESDescriptor(int)
String	getURL()

V.1.9.2 Methods

getESDescriptor(int)
`public org.iso.mpeg.mpegj.ESDescriptor getESDescriptor(int)`

Returns the ESDescriptor associated with the given ESID

Parameters:
`ESDescriptor` - ID

Returns:
ESDescriptor object

getESDescriptors()
`public java.util.Vector getESDescriptors()`

Returns the ESDescriptors associated with this ObjectDescriptor

Returns:
vector of ESDescriptors

getObjectDescriptorID()
`public int getObjectDescriptorID()`

Returns the ID associated with this Object Descriptor

Returns:
ObjectDescriptorID

getURL()
`public java.lang.String getURL()`

Returns the URL String

Returns:
String

V.1.10 org.iso.mpeg.mpegj.SceneManagerNotFoundException

V.1.10.1 Syntax

```
public class SceneManagerNotFoundException extends java.lang.Exception
```

java.lang.Object
 |
 +--java.lang.Throwable
  |
  +--java.lang.Exception
  |
  +--**org.iso.mpeg.mpegj.SceneManagerNotFoundException**

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000424

All Implemented Interfaces:
java.io.Serializable

V.1.10.2 Description

Member Summary	
Constructors	
	SceneManagerNotFoundException()

V.1.10.3 Constructors

SceneManagerNotFoundException()
```
public  SceneManagerNotFoundException()
```

V.2 package org.iso.mpeg.mpegj.resource

V.2.1 Description

Class Summary	
Interfaces	
MPRendererFrameEventGenerator	An interface for adding and removing frame event listeners.
Dimension	Interface for Dimension with set and get methods for Width and Height
StaticCapability	This interface is used to provide developers a simple way to access basic dynamic terminal capabilities.
MPRendererMediaListener	An interface that is called by the renderer when registered.
MPRendererEventGenerator	An interface for adding and removing Renderer event listeners.
TerminalProfileManager	Thisinterface allows applications to query the terminal profiles.
Renderer	Interface for the Rendering Engine.
DynamicCapabilityObserver	Interface to observe Dynamic capabilities.
MPDecoderMediaListener	An interface that is called by an MPDecoder when registered.
DynamicCapability	This interface is used to provide developers a simple way to access basic dynamic terminal capabilities.
MPDecoderEventGenerator	An interface for adding and removing Decoder event listeners.
MPRendererFrameListener	An interface that is called periodically by the renderer.
Classes	
MPDecoderMediaEvents	An event object that is given to an MPDecoderMediaListener.
MPDecoderEventGeneratorImp	An interface for adding and removing Decoder event listeners.
MPRendererFrameEvents	An event object that is given to an MPRendererFrameListener.
CapabilityManager	A CapabilityManager monitors latency and bandwidth of network connections and manages peripheral devices.
ResourceManager	The resource manager is used for regulation of performance.
CapabilityInfo	This class is used to store information related to dynamic observers.
MPRendererMediaEvents	An event object that is given to an MPRendererMediaListener.
CPUmonitor	The CPUmonitor class is used to monitor selected terminal activities including CPU load, and free/available system memory utilisation.
CapabilityObserver	This class is used to store information related to dynamic observers.
Exceptions	
IllegalDecoderMediaEventsException	
ResourceManagerNotFoundException	The class ResourceManagerNotFoundException is a subclass of MPEGJException.
RendererNotFoundException	The class RendererNotFoundException is a subclass of MPEGJException.
IllegalStreamMediaEventsException	

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000425

CapabilityManagerNot-FoundException	The class CapabilityManagerNotFoundException is a subclass of MPEGJException.
IllegalRendererMedi-aEventsException	This a subclass of MPEGJException.

V.2.2 org.iso.mpeg.mpegj.resource.CapabilityInfo

V.2.2.1 Syntax

```
public class CapabilityInfo extends java.lang.Object
```

java.lang.Object
 |
 +--**org.iso.mpeg.mpegj.resource.CapabilityInfo**

V.2.2.2 Description

This class is used to store information related to dynamic observers.

Member Summary	
Constructors	
	CapabilityInfo(Observer, long, boolean)
Methods	
boolean	verifyCapabilityValue(long)

V.2.2.3 Constructors

CapabilityInfo(Observer, long, boolean)
```
public  CapabilityInfo(java.util.Observer, long, boolean)
```

V.2.2.4 Methods

verifyCapabilityValue(long)
```
public boolean verifyCapabilityValue(long)
```
Verify Capability Value

Parameters:
`value` - the capability value that has to be verified

Returns:
true is the passed value exceeds the threshold, false otherwise.

V.2.3 org.iso.mpeg.mpegj.resource.CapabilityManager

V.2.3.1 Syntax

```
public class CapabilityManager extends java.lang.Object implements
   org.iso.mpeg.mpegj.resource.DynamicCapability,
   org.iso.mpeg.mpegj.resource.StaticCapability,
   org.iso.mpeg.mpegj.resource.DynamicCapabilityObserver,
   org.iso.mpeg.mpegj.resource.TerminalProfileManager
```

java.lang.Object
 |
 +--**org.iso.mpeg.mpegj.resource.CapabilityManager**

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000426

All Implemented Interfaces:
org.iso.mpeg.mpegj.resource.DynamicCapability,
org.iso.mpeg.mpegj.resource.DynamicCapabilityObserver,
org.iso.mpeg.mpegj.resource.StaticCapability,
org.iso.mpeg.mpegj.resource.TerminalProfileManager

V.2.3.2 Description

A CapabilityManager monitors latency and bandwidth of network connections and manages peripheral devices.

Functionality:

Interacts with DMIF.

Interacts with presentation devices.

Throws profile exceptions.

Interacts with input devices.

Throws MPEG4Events.

The CapabilityManager class handles capability through the CapabilityManager. The CapabilityManager offers a simple and generic way to handle capabilities. The CapabilityManager implements some interfaces in order to simplify developers work when basic capabilities need to be managed. [An interface incapsulates a coherent set of services and attributes, i.e. a Role, without exlicitly binding this function to that of any particular object or code]. The interfaces are:

DynamicCapability This interface is used to provide developers a simple way to access basic dynamic terminal capabilities. A capability is dynamic if it can change value/status at runtime. For instance memory usage is a dynamic capability.

StaticCapabilityThis interface is used to provide developers a simple way to access basic dynamic terminal capabilities. A capability is considered static if its value/static cannot change at runtime. For instance the audio card type is a static capability.

CapabilityObserverThis interface is used to provide developers a few methods that allow them to simply monitor the value of common capabilities such as memory and resource usage.

Member Summary	
Constructors	
	CapabilityManager()
Methods	
void	notifyApplicationFreeMemory(long, Observer)
void	notifyTerminalFreeMemory(long, Observer)
void	notifyTerminalLoad(long, Observer)
void	notifyTerminalNetworkLoad(long, Observer)
void	deleteObserver(Observer)
int	getNumCPUs()
int	getCPUSpeed(int)
String	getCPUType(int)
String	getMouseType()
int	getDisplayColorDepth()
String	getDisplayType()
String	getKeyboardType()
String	getNetworkType()
int	getNumParallelPorts()
int	getNumSerialPorts()
Dimension	getScreenSize()
short	getScreenDepth()
short	getScreenResolution()
String	getOSLanguage()
String	getOSType()
String	getTerminalArchitecture()
String[]	getAudioDrivers()
String[]	getMIDIDrivers()
String[]	getVideoDrivers()
String	getModemType()

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000427

long	getFreeTerminalMemory()
long	getTotalTerminalMemory()
long	getTotalApplicationMemory()
long	getFreeApplicationMemory()
long	getTerminalLoad()
long	getNetworkLoad()
short	getSceneDescriptionProfile()
short	getVisualProfile()
short	getAudioProfile()

V.2.3.3 Constructors

CapabilityManager()
`public CapabilityManager()`

V.2.3.4 Methods

deleteObserver(Observer)
`public void deleteObserver(java.util.Observer)`

deleteObserver

Specified By:
deleteObserver(Observer) in interface org.iso.mpeg.mpegj.resource.DynamicCapability-Observer

Parameters:
`objToDeregister` - observer object previously registered using registerXXX methods

getAudioDrivers()
`public java.lang.String[] getAudioDrivers()`

Terminal Audio Enc/Dec

Specified By:
getAudioDrivers() in interface org.iso.mpeg.mpegj.resource.StaticCapability

Returns:
the available audio drivers.

getAudioProfile()
`public short getAudioProfile()`

Specified By:
getAudioProfile() in interface org.iso.mpeg.mpegj.resource.TerminalProfileManager

getCPUSpeed(int)
`public int getCPUSpeed(int)`

Terminal CPU Speed

Specified By:
getCPUSpeed(int) in interface org.iso.mpeg.mpegj.resource.StaticCapability

Returns:
the speed of the 'idx' CPU or -1 is an error occurs

getCPUType(int)
`public java.lang.String getCPUType(int)`

Terminal CPU type (Pentium for instance)

Specified By:
getCPUType(int) in interface org.iso.mpeg.mpegj.resource.StaticCapability

Parameters:
`idx` - the index of the CPU (idx >= 0)

Returns:
the type of CPU

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000428

Throws:
`IllegalArgumentException` - the index is either negative or out of range.

getDisplayColorDepth()
`public int getDisplayColorDepth()`

Terminal Display Color Depth

Specified By:
`getDisplayColorDepth()` in interface `org.iso.mpeg.mpegj.resource.StaticCapability`

Returns:
the display color depth (bits per pixel)

getDisplayType()
`public java.lang.String getDisplayType()`

Terminal Display Type (e.g. VGA)

Specified By:
`getDisplayType()` in interface `org.iso.mpeg.mpegj.resource.StaticCapability`

Returns:
the display type

getFreeApplicationMemory()
`public long getFreeApplicationMemory()`

Free application memory (RAM)

Specified By:
`getFreeApplicationMemory()` in interface `org.iso.mpeg.mpegj.resource.DynamicCapability`

Returns:
the amount of available (at application level e.g. JavaVM) free memory (bytes).

getFreeTerminalMemory()
`public long getFreeTerminalMemory()`

Terminal free memory (RAM)

Specified By:
`getFreeTerminalMemory()` in interface `org.iso.mpeg.mpegj.resource.DynamicCapability`

Returns:
the amount of free memory (bytes) installed on the terminal.

getKeyboardType()
`public java.lang.String getKeyboardType()`

Terminal Keyboard Type

Specified By:
`getKeyboardType()` in interface `org.iso.mpeg.mpegj.resource.StaticCapability`

Returns:
the keyboard type, NULL if the keyboard is not present.

getMIDIDrivers()
`public java.lang.String[] getMIDIDrivers()`

Terminal MIDI Enc/Dec

Specified By:
`getMIDIDrivers()` in interface `org.iso.mpeg.mpegj.resource.StaticCapability`

Returns:
the available MIDI drivers.

getModemType()
`public java.lang.String getModemType()`

Terminal modem type

Specified By:
`getModemType()` in interface `org.iso.mpeg.mpegj.resource.StaticCapability`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000429

Returns:
the modem speed ("28K", "56K" etc.) or "" is the modem is not installed.

getMouseType()
`public java.lang.String getMouseType()`

Terminal Mouse Type

Specified By:
`getMouseType()` in interface `org.iso.mpeg.mpegj.resource.StaticCapability`

Returns:
the mouse type or null if an error occurred

getNetworkLoad()
`public long getNetworkLoad()`

Terminal Network Load

Specified By:
`getNetworkLoad()` in interface `org.iso.mpeg.mpegj.resource.DynamicCapability`

Returns:
the network load (percentage).

getNetworkType()
`public java.lang.String getNetworkType()`

Terminal Network Type

Specified By:
`getNetworkType()` in interface `org.iso.mpeg.mpegj.resource.StaticCapability`

Returns:
the network type or null if an error occurred.

getNumCPUs()
`public int getNumCPUs()`

Number of Terminal CPUs

Specified By:
`getNumCPUs()` in interface `org.iso.mpeg.mpegj.resource.StaticCapability`

Returns:
the number of CPUs that equip the terminal or -1 is an error occurs

getNumParallelPorts()
`public int getNumParallelPorts()`

Terminal parallel IEEE-1284 port(s)

Specified By:
`getNumParallelPorts()` in interface `org.iso.mpeg.mpegj.resource.StaticCapability`

Returns:
the number of parallel ports present on the terminal.

getNumSerialPorts()
`public int getNumSerialPorts()`

Terminal serial RS-232 port(s)

Specified By:
`getNumSerialPorts()` in interface `org.iso.mpeg.mpegj.resource.StaticCapability`

Returns:
the number of serial ports present on the terminal.

getOSLanguage()
`public java.lang.String getOSLanguage()`

Terminal OS language

Specified By:
`getOSLanguage()` in interface `org.iso.mpeg.mpegj.resource.StaticCapability`

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000430

Returns:
the language of the OS version installed on the terminal.

getOSType()
`public java.lang.String getOSType()`

Terminal OS type (for instance JavaPC, Windows, MacOS)

Specified By:
`getOSType()` in interface `org.iso.mpeg.mpegj.resource.StaticCapability`

Returns:
the OS type installed on the terminal.

getSceneDescriptionProfile()
`public short getSceneDescriptionProfile()`

Scene Description Profile

Specified By:
`getSceneDescriptionProfile()` in interface `org.iso.mpeg.mpegj.resource.TerminalProfileManager`

Returns:
the actual scene description profile.

getScreenDepth()
`public short getScreenDepth()`

Terminal screen depth

Specified By:
`getScreenDepth()` in interface `org.iso.mpeg.mpegj.resource.StaticCapability`

Returns:
the screen depth (bits/inch)

getScreenResolution()
`public short getScreenResolution()`

Terminal screen resolution

Specified By:
`getScreenResolution()` in interface `org.iso.mpeg.mpegj.resource.StaticCapability`

Returns:
the screen resolution (dots/inch)

getScreenSize()
`public org.iso.mpeg.mpegj.resource.Dimension getScreenSize()`

Terminal screen size

Specified By:
`getScreenSize()` in interface `org.iso.mpeg.mpegj.resource.StaticCapability`

Returns:
the screen size (pixel x pixel)

getTerminalArchitecture()
`public java.lang.String getTerminalArchitecture()`

Terminal architecture (for instance Alpha, x86)

Specified By:
`getTerminalArchitecture()` in interface `org.iso.mpeg.mpegj.resource.StaticCapability`

Returns:
the terminal architecture type.

getTerminalLoad()
`public long getTerminalLoad()`

Terminal CPU Load

Specified By:
`getTerminalLoad()` in interface `org.iso.mpeg.mpegj.resource.DynamicCapability`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000431

Returns:
the load (percentage) of the terminal CPUs

getTotalApplicationMemory()
`public long getTotalApplicationMemory()`

Total application memory (RAM)

Specified By:
`getTotalApplicationMemory()` in interface `org.iso.mpeg.mpegj.resource.DynamicCapability`

Returns:
the total amount of memory (bytes).

getTotalTerminalMemory()
`public long getTotalTerminalMemory()`

Terminal total memory (RAM)

Specified By:
`getTotalTerminalMemory()` in interface `org.iso.mpeg.mpegj.resource.DynamicCapability`

Returns:
the total amount of memory (bytes) installed on the terminal.

getVideoDrivers()
`public java.lang.String[] getVideoDrivers()`

Terminal Video Enc/Dec

Specified By:
`getVideoDrivers()` in interface `org.iso.mpeg.mpegj.resource.StaticCapability`

Returns:
the available video drivers.

getVisualProfile()
`public short getVisualProfile()`

Specified By:
`getVisualProfile()` in interface `org.iso.mpeg.mpegj.resource.TerminalProfileManager`

notifyApplicationFreeMemory(long, Observer)
`public void notifyApplicationFreeMemory(long, java.util.Observer)`

notifyApplicationFreeMemory

Specified By:
`notifyApplicationFreeMemory(long, Observer)` in interface `org.iso.mpeg.mpegj.resource.DynamicCapabilityObserver`

Parameters:
`freeApplicationMemory` - application memory threshold

`objToNotify` - object to notify when the specified threshold is exceeded

Throws:
`IllegalArgumentException` - the freeApplicationMemory is either negative or out of range.

notifyTerminalFreeMemory(long, Observer)
`public void notifyTerminalFreeMemory(long, java.util.Observer)`

notifyTerminalFreeMemory

Specified By:
`notifyTerminalFreeMemory(long, Observer)` in interface `org.iso.mpeg.mpegj.resource.DynamicCapabilityObserver`

Parameters:
`freeTerminalMemory` - terminal memory threshold

`objToNotify` - object to notify when the specified threshold is exceeded

Throws:
`IllegalArgumentException` - the freeTerminalMemory is either negative or out of range.

notifyTerminalLoad(long, Observer)
`public void notifyTerminalLoad(long, java.util.Observer)`

notifyTerminalLoad

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000432

Specified By:
notifyTerminalLoad(long, Observer) in interface org.iso.mpeg.mpegj.resource.Dynamic-CapabilityObserver

Parameters:
cpuLoadPercentage - terminal CPU load percentage threshold

objToNotify - object to notify when the specified threshold is exceeded

Throws:
IllegalArgumentException - the cpuLoadPercentage is either negative or out of range.

notifyTerminalNetworkLoad(long, Observer)
public void notifyTerminalNetworkLoad(long, java.util.Observer)

notifyTerminalNetworkLoad

Specified By:
notifyTerminalNetworkLoad(long, Observer) in interface org.iso.mpeg.mpegj.resource.DynamicCapabilityObserver

Parameters:
nwLoadPercentage - network load percentage threshold

objToNotify - object to notify when the specified threshold is exceeded

Throws:
IllegalArgumentException - the nwLoadPercentage is either negative or out of range.

V.2.4 org.iso.mpeg.mpegj.resource.CapabilityManagerNotFoundException

V.2.4.1 Syntax

```
public class CapabilityManagerNotFoundException extends
   org.iso.mpeg.mpegj.MPEGJException
```

java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--**org.iso.mpeg.mpegj.resource.CapabilityManagerNotFoundException**

All Implemented Interfaces:
java.io.Serializable

V.2.4.2 Description

The class CapabilityManagerNotFoundException is a subclass of MPEGJException. This Exception is thrown by the method getCapabilityManager() of Resource Manager.

Member Summary	
Constructors	CapabilityManagerNotFoundException() CapabilityManagerNotFoundException(String)

V.2.4.3 Constructors

CapabilityManagerNotFoundException()
public CapabilityManagerNotFoundException()

Constructs an CapabilityManagerNotFoundException with no specified detail message.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000433

CapabilityManagerNotFoundException(String)
`public CapabilityManagerNotFoundException(java.lang.String)`

Constructs an CapabilityManagerFoundException with a detailed message.

V.2.5 org.iso.mpeg.mpegj.resource.CapabilityObserver

V.2.5.1 Syntax

```
public class CapabilityObserver extends java.util.Observable implements
   java.lang.Runnable
```

java.lang.Object
 |
 +--java.util.Observable
     |
     +--**org.iso.mpeg.mpegj.resource.CapabilityObserver**

All Implemented Interfaces:
java.lang.Runnable

V.2.5.2 Description

This class is used to store information related to dynamic observers.

Member Summary	
Constructors	
	`CapabilityObserver(CapabilityManager, short)`
Methods	
`void`	`addObserver(Observer, long, boolean)`
`void`	`deleteObserver(Observer)`
`void`	`deleteObservers()`
`void`	`run()`

V.2.5.3 Constructors

CapabilityObserver(CapabilityManager, short)
`public CapabilityObserver(org.iso.mpeg.mpegj.resource.CapabilityManager, short)`

CapabilityObserver constructor

Parameters:
`t` - a reference to the CapabilityManager

`_capabilityCode` - objToDeregister the object to be deregistered.

V.2.5.4 Methods

addObserver(Observer, long, boolean)
`public synchronized void addObserver(java.util.Observer, long, boolean)`

Add a new observer

Parameters:
o - the Observer

`_threshold` - the threshold value

deleteObserver(Observer)
`public synchronized void deleteObserver(java.util.Observer)`

deleteObserver: overrided method

Overrides:
java.util.Observable.deleteObserver(java.util.Observer) in class java.util.Observable

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000434

Parameters:
o - the Observer

deleteObservers()
```
public synchronized void deleteObservers()
```

deleteObservers: overrided method

Overrides:
java.util.Observable.deleteObservers() in class java.util.Observable

Parameters:
o - the Observer

run()
```
public void run()
```

run: the run() method (called by Thread)

Specified By:
java.lang.Runnable.run() in interface java.lang.Runnable

V.2.6 org.iso.mpeg.mpegj.resource.CPUmonitor

V.2.6.1 Syntax

```
public class CPUmonitor extends java.lang.Object
```

java.lang.Object
 |
 +--**org.iso.mpeg.mpegj.resource.CPUmonitor**

V.2.6.2 Description

The CPUmonitor class is used to monitor selected terminal activities including CPU load, and free/available system memory utilisation.

Member Summary	
Constructors	
	CPUmonitor()
Methods	
short	getCPUload()
long	getFreeSystemMemory()
long	getAvailSystemMemory()

V.2.6.3 Constructors

CPUmonitor()
```
public  CPUmonitor()
```

V.2.6.4 Methods

getAvailSystemMemory()
```
public static native long getAvailSystemMemory()
```

getCPUload()
```
public static native short getCPUload()
```

getFreeSystemMemory()
```
public static native long getFreeSystemMemory()
```

V.2.7 org.iso.mpeg.mpegj.resource.Dimension

V.2.7.1 Syntax

```
public interface Dimension
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000435

V.2.7.2 Description

Interface for Dimension with set and get methods for Width and Height

Member Summary	
Methods	
int	getHeight()
int	getWidth()
int	setHeight(int)
int	setWidth(int)

V.2.7.3 Methods

getHeight()
`public int getHeight()`

Returns Height as an integer

getWidth()
`public int getWidth()`

Returns Width as an integer

setHeight(int)
`public int setHeight(int)`

Sets the Height to the specified value

setWidth(int)
`public int setWidth(int)`

Sets the Width to the specified value

V.2.8 org.iso.mpeg.mpegj.resource.DynamicCapability

V.2.8.1 Syntax

```
public interface DynamicCapability
```

All Known Implementing Classes:
org.iso.mpeg.mpegj.resource.CapabilityManager

V.2.8.2 Description

This interface is used to provide developers a simple way to access basic dynamic terminal capabilities. A capability is dynamic if it can change value/status at runtime. For instance memory usage is a dynamic capability.

Member Summary	
Methods	
long	getFreeTerminalMemory()
long	getTotalTerminalMemory()
long	getTotalApplicationMemory()
long	getFreeApplicationMemory()
long	getTerminalLoad()
long	getNetworkLoad()

V.2.8.3 Methods

getFreeApplicationMemory()
`public long getFreeApplicationMemory()`

Free application memory (RAM)

Returns:
the amount of available (at application level e.g. JavaVM) free memory (bytes).

getFreeTerminalMemory()
`public long getFreeTerminalMemory()`

Terminal free memory (RAM)

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000436

Returns:
the amount of free memory (bytes) installed on the terminal.

getNetworkLoad()
`public long getNetworkLoad()`

Terminal Network Load

Returns:
the network load (percentage).

getTerminalLoad()
`public long getTerminalLoad()`

Terminal CPU Load

Returns:
the load (percentage) of the terminal CPUs

getTotalApplicationMemory()
`public long getTotalApplicationMemory()`

Total application memory (RAM)

Returns:
the total amount of memory (bytes).

getTotalTerminalMemory()
`public long getTotalTerminalMemory()`

Terminal total memory (RAM)

Returns:
the total amount of memory (bytes) installed on the terminal.

V.2.9 org.iso.mpeg.mpegj.resource.DynamicCapabilityObserver

V.2.9.1 Syntax

```
public interface DynamicCapabilityObserver
```

All Known Implementing Classes:
`org.iso.mpeg.mpegj.resource.CapabilityManager`

V.2.9.2 Description

Interface to observe Dynamic capabilities. This interface is used to be notified of varying capabilities.

Member Summary	
Methods	
void	notifyApplicationFreeMemory(long, Observer)
void	notifyTerminalFreeMemory(long, Observer)
void	notifyTerminalLoad(long, Observer)
void	notifyTerminalNetworkLoad(long, Observer)
void	deleteObserver(Observer)

V.2.9.3 Methods

deleteObserver(Observer)
`public void deleteObserver(java.util.Observer)`

This method is used to deregister an observer previously registered using any of the notifyXXX methods.

Parameters:
`objToDeregister` - the object to be deregistered.

notifyApplicationFreeMemory(long, Observer)
`public void notifyApplicationFreeMemory(long, java.util.Observer)`

This method is used by AVSession objects that want to be notified when the application free memory goes below a certain threshold.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000437

Parameters:
`freeApplicationMemory` - the free application memory threshold (bytes).

`objToNotify` - the object to be notified.

Throws:
`IllegalArgumentException` - the threshold is out of range.

notifyTerminalFreeMemory(long, Observer)
`public void notifyTerminalFreeMemory(long, java.util.Observer)`

This method is used by AVSession objects that want to be notified when the total terminal free memory goes below a certain threshold.

Parameters:
`freeTerminalMemory` - the free application memory threshold (bytes).

`objToNotify` - the object to be notified.

Throws:
`IllegalArgumentException` - the threshold is out of range.

notifyTerminalLoad(long, Observer)
`public void notifyTerminalLoad(long, java.util.Observer)`

This method is used by AVSession objects that want to be notified when the terminal CPU load goes above a certain percentage.

Parameters:
`cpuLoadPercentage` - the cpu load (percentage) threshold.

`objToNotify` - the object to be notified.

Throws:
`IllegalArgumentException` - the threshold out of range [0...100].

notifyTerminalNetworkLoad(long, Observer)
`public void notifyTerminalNetworkLoad(long, java.util.Observer)`

This method is used by AVSession objects that want to be notified when the terminal network load goes above a certain percentage.

Parameters:
`nwLoadPercentage` - the network load (percentage) threshold.

`objToNotify` - the object to be notified.

Throws:
`IllegalArgumentException` - the threshold out of range [0...100].

V.2.10 org.iso.mpeg.mpegj.resource.IllegalDecoderMediaEventsException

V.2.10.1 Syntax

```
public class IllegalDecoderMediaEventsException extends
   org.iso.mpeg.mpegj.MPEGJException
```

```
java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--org.iso.mpeg.mpegj.resource.IllegalDecoderMediaEventsException
```

All Implemented Interfaces:
java.io.Serializable

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000438

V.2.10.2 Description

Member Summary	
Constructors	IllegalDecoderMediaEventsException() IllegalDecoderMediaEventsException(String)

V.2.10.3 Constructors

IllegalDecoderMediaEventsException()
`public IllegalDecoderMediaEventsException()`

Constructs an IllegalDecoderMediaEventsException with no specified detail message.

IllegalDecoderMediaEventsException(String)
`public IllegalDecoderMediaEventsException(java.lang.String)`

Constructs an IllegalDecoderMediaEventsException with a detailed message.

V.2.11 org.iso.mpeg.mpegj.resource.IllegalRendererMediaEventsException

V.2.11.1 Syntax

```
public class IllegalRendererMediaEventsException extends
   org.iso.mpeg.mpegj.MPEGJException
```

java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--**org.iso.mpeg.mpegj.resource.IllegalRendererMediaEventsException**

All Implemented Interfaces:
java.io.Serializable

V.2.11.2 Description

This a subclass of MPEGJException. This is thrown when an illegal Renderer Media Event is posted

Member Summary	
Constructors	IllegalRendererMediaEventsException() IllegalRendererMediaEventsException(String)

V.2.11.3 Constructors

IllegalRendererMediaEventsException()
`public IllegalRendererMediaEventsException()`

Constructs an IllegalRendererMediaEventsException with no specified detail message.

IllegalRendererMediaEventsException(String)
`public IllegalRendererMediaEventsException(java.lang.String)`

Constructs an IllegalRendererMediaEventsException with a detailed message.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000439

V.2.12 org.iso.mpeg.mpegj.resource.IllegalStreamMediaEventsException

V.2.12.1 Syntax

```
public class IllegalStreamMediaEventsException extends
   org.iso.mpeg.mpegj.MPEGJException
```

java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--**org.iso.mpeg.mpegj.resource.IllegalStreamMediaEventsException**

All Implemented Interfaces:
java.io.Serializable

V.2.12.2 Description

Member Summary	
Constructors	IllegalStreamMediaEventsException()

V.2.12.3 Constructors

IllegalStreamMediaEventsException()
```
public  IllegalStreamMediaEventsException()
```

V.2.13 org.iso.mpeg.mpegj.resource.MPDecoderEventGenerator

V.2.13.1 Syntax

```
public interface MPDecoderEventGenerator
```

All Known Subinterfaces:
org.iso.mpeg.mpegj.decoder.MPDecoder

All Known Implementing Classes:
org.iso.mpeg.mpegj.resource.MPDecoderEventGeneratorImp

V.2.13.2 Description

An interface for adding and removing Decoder event listeners.

Member Summary	
Methods	
void	addMPDecoderMediaListener(MPDecoderMediaListener)
void	removeMPDecoderMediaListener(MPDecoderMediaListener)

V.2.13.3 Methods

addMPDecoderMediaListener(MPDecoderMediaListener)
```
public void
addMPDecoderMediaListener(org.iso.mpeg.mpegj.resource.MPDecoderMediaListener)
```
Add a Render event listener. The listener is called by an MPdecoder when registered.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000440

Parameters:
`l` - the listener to add.

removeMPDecoderMediaListener(MPDecoderMediaListener)
`public void removeMPDecoderMediaListener(org.iso.mpeg.mpegj.resource.MPDecoderMediaListener)`

Remove a Decoder event listener.

Parameters:
`l` - the listener to remove.

V.2.14 org.iso.mpeg.mpegj.resource.MPDecoderEventGeneratorImp

V.2.14.1 Syntax

```
public class MPDecoderEventGeneratorImp extends java.lang.Object implements
   org.iso.mpeg.mpegj.resource.MPDecoderEventGenerator
```

java.lang.Object
 |
 +--**org.iso.mpeg.mpegj.resource.MPDecoderEventGeneratorImp**

All Implemented Interfaces:
`org.iso.mpeg.mpegj.resource.MPDecoderEventGenerator`

V.2.14.2 Description

An interface for adding and removing Decoder event listeners.

Member Summary	
Constructors	
	`MPDecoderEventGeneratorImp()`
Methods	
void	`addMPDecoderMediaListener(MPDecoderMediaListener)`
void	`removeMPDecoderMediaListener(MPDecoderMediaListener)`

V.2.14.3 Constructors

MPDecoderEventGeneratorImp()
`public MPDecoderEventGeneratorImp()`

V.2.14.4 Methods

addMPDecoderMediaListener(MPDecoderMediaListener)
`public void addMPDecoderMediaListener(org.iso.mpeg.mpegj.resource.MPDecoderMediaListener)`

Add a Render event listener. The listener is called by an MPdecoder when registered.

Specified By:
`addMPDecoderMediaListener(MPDecoderMediaListener)` in interface `org.iso.mpeg.mpegj.resource.MPDecoderEventGenerator`

Parameters:
`l` - the listener to add.

removeMPDecoderMediaListener(MPDecoderMediaListener)
`public void removeMPDecoderMediaListener(org.iso.mpeg.mpegj.resource.MPDecoderMediaListener)`

Remove a Decoder event listener.

Specified By:
`removeMPDecoderMediaListener(MPDecoderMediaListener)` in interface `org.iso.mpeg.mpegj.resource.MPDecoderEventGenerator`

Parameters:
`l` - the listener to remove.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000441

V.2.15 org.iso.mpeg.mpegj.resource.MPDecoderMediaEvents

V.2.15.1 Syntax

```
public class MPDecoderMediaEvents extends java.util.EventObject
```

java.lang.Object
 |
 +--java.util.EventObject
     |
     +--**org.iso.mpeg.mpegj.resource.MPDecoderMediaEvents**

All Implemented Interfaces:
java.io.Serializable

V.2.15.2 Description

An event object that is given to an MPDecoderMediaListener.

Member Summary	
Fields	
int	DECODER_OVERFLOW
int	DECODER_SYNC_ERROR
int	STREAM_UNDERFLOW
int	STREAM_OVERFLOW
int	STREAM_START
int	STREAM_END
Constructors	
	MPDecoderMediaEvents(Object, int)
Methods	
int	getCondition()

V.2.15.3 Fields

DECODER_OVERFLOW
```
public static final int DECODER_OVERFLOW
```
DECODER_OVERFLOW = 0

DECODER_SYNC_ERROR
```
public static final int DECODER_SYNC_ERROR
```
DECODER_SYNC_ERROR = 1

STREAM_END
```
public static final int STREAM_END
```
STREAM_END = 5

STREAM_OVERFLOW
```
public static final int STREAM_OVERFLOW
```
STREAM_OVERFLOW = 3

STREAM_START
```
public static final int STREAM_START
```
STREAM_START = 4

STREAM_UNDERFLOW
```
public static final int STREAM_UNDERFLOW
```
STREAM_UNDERFLOW = 2

V.2.15.4 Constructors

MPDecoderMediaEvents(Object, int)
```
public  MPDecoderMediaEvents(java.lang.Object, int)
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000442

Construct an event object with condition. condition can be decoder overflow, sync error, or end of decode, otherwise throws exception

Parameters:
`source` - the source of the event

`condition` - The condition variable can correspond the to decoder overflow, sync error, stream start or stop

Throws:
`org.iso.mpeg.mpegj.resource.IllegalDecoderMediaEventsException` - thrown an Illegal Decoder Event is thrown

V.2.15.5 Methods

getCondition()
`public int getCondition()`

The obtained condition can correspond to decoder overflow, sync error, stream underflow, stream overflow, stream start or stream end

V.2.16 org.iso.mpeg.mpegj.resource.MPDecoderMediaListener

V.2.16.1 Syntax

```
public interface MPDecoderMediaListener extends java.util.EventListener
```

All Superinterfaces:
java.util.EventListener

V.2.16.2 Description

An interface that is called by an MPDecoder when registered.

See Also:
`org.iso.mpeg.mpegj.resource.MPDecoderEventGenerator`

Member Summary	
Methods	
void	mPDecoderMediaHandler(MPDecoderMediaEvents)

V.2.16.3 Methods

mPDecoderMediaHandler(MPDecoderMediaEvents)
`public void mPDecoderMediaHandler(org.iso.mpeg.mpegj.resource.MPDecoderMediaEvents)`

Notification of a Renderer event, called by a MPDecoder during decoder overflow, sync error, stream start or stop.

Parameters:
`event` - refers to the source of the event.

V.2.17 org.iso.mpeg.mpegj.resource.MPRendererEventGenerator

V.2.17.1 Syntax

```
public interface MPRendererEventGenerator
```

All Known Subinterfaces:
`org.iso.mpeg.mpegj.resource.Renderer`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000443

V.2.17.2 Description

An interface for adding and removing Renderer event listeners.

Member Summary	
Methods	
void	addMPRendererMediaListener(MPRendererMediaListener)
void	removeMPRendererMediaListener(MPRendererMediaListener)

V.2.17.3 Methods

addMPRendererMediaListener(MPRendererMediaListener)
`public void addMPRendererMediaListener(org.iso.mpeg.mpegj.resource.MPRendererMediaListener)`

Add a Renderer event listener. The listener is called by the renderer when registered.

Parameters:
`l` - the listener to add.

removeMPRendererMediaListener(MPRendererMediaListener)
`public void removeMPRendererMediaListener(org.iso.mpeg.mpegj.resource.MPRendererMediaListener)`

Remove a Renderer event listener.

Parameters:
`l` - the listener to remove.

V.2.18 org.iso.mpeg.mpegj.resource.MPRendererFrameEventGenerator

V.2.18.1 Syntax

```
public interface MPRendererFrameEventGenerator
```

All Known Subinterfaces:
`org.iso.mpeg.mpegj.resource.Renderer`

V.2.18.2 Description

An interface for adding and removing frame event listeners.

Member Summary	
Methods	
void	addMPRendererFrameListener(MPRendererFrameListener, int)
void	removeMPRendererFrameListener(MPRendererFrameListener)

V.2.18.3 Methods

addMPRendererFrameListener(MPRendererFrameListener, int)
`public void addMPRendererFrameListener(org.iso.mpeg.mpegj.resource.MPRendererFrameListener, int)`

Add a frame event listener. The listener is called periodically by the renderer. The frames parameter indicates the number of frames between each callback.

Parameters:
`l` - the listener to add.

`frames` - the number of frames between each callback.

removeMPRendererFrameListener(MPRendererFrameListener)
`public void removeMPRendererFrameListener(org.iso.mpeg.mpegj.resource.MPRendererFrameListener)`

Remove a frame event listener.

Parameters:
`l` - the listener to remove.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000444

V.2.19 org.iso.mpeg.mpegj.resource.MPRendererFrameEvents

V.2.19.1 Syntax

```
public class MPRendererFrameEvents extends java.util.EventObject
```

java.lang.Object
 |
 +--java.util.EventObject
  |
  +--**org.iso.mpeg.mpegj.resource.MPRendererFrameEvents**

All Implemented Interfaces:
java.io.Serializable

V.2.19.2 Description

An event object that is given to an MPRendererFrameListener.

Member Summary	
Constructors	
	MPRendererFrameEvents(Object)

V.2.19.3 Constructors

MPRendererFrameEvents(Object)
`public MPRendererFrameEvents(java.lang.Object)`

Construct a frame event object with a source.

Parameters:
`source` - the source of the event.

V.2.20 org.iso.mpeg.mpegj.resource.MPRendererFrameListener

V.2.20.1 Syntax

```
public interface MPRendererFrameListener
```

V.2.20.2 Description

An interface that is called periodically by the renderer. The number of frames between each call is specified when the listener is registered.

See Also:
org.iso.mpeg.mpegj.resource.MPRendererFrameEventGenerator

Member Summary	
Methods	
void	frameEvent(MPRendererFrameEvents)

V.2.20.3 Methods

frameEvent(MPRendererFrameEvents)
`public void frameEvent(org.iso.mpeg.mpegj.resource.MPRendererFrameEvents)`

Notification of a frame event, called periodically by the renderer.

Parameters:
`event` - refers to the source of the event.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000445

V.2.21 org.iso.mpeg.mpegj.resource.MPRendererMediaEvents

V.2.21.1 Syntax

```
public class MPRendererMediaEvents extends java.util.EventObject
```

java.lang.Object
 |
 +--java.util.EventObject
     |
     +--**org.iso.mpeg.mpegj.resource.MPRendererMediaEvents**

All Implemented Interfaces:
java.io.Serializable

V.2.21.2 Description

An event object that is given to an MPRendererMediaListener.

Member Summary	
Fields	
int	DECODER_UNDERFLOW
int	MISSED_FRAMES
Constructors	
	MPRendererMediaEvents(Object, int)
Methods	
int	getCondition()

V.2.21.3 Fields

DECODER_UNDERFLOW
`public static final int DECODER_UNDERFLOW`

DECODER_UNDERFLOW = 0

MISSED_FRAMES
`public static final int MISSED_FRAMES`

MISSED_FRAMES = 1

V.2.21.4 Constructors

MPRendererMediaEvents(Object, int)
`public  MPRendererMediaEvents(java.lang.Object, int)`

Construct an event object with condition. condition can be decoder underflow or missed frames

Parameters:
`source` - the source of the event

`condition` - The condition variable can correspond the to decoder underflow or missed frame.

Throws:
`org.iso.mpeg.mpegj.resource.IllegalRendererMediaEventsException` - thrown an Illegal Renderer Event is thrown

V.2.21.5 Methods

getCondition()
`public int getCondition()`

Gets the Condtion. The obtained condition can correspond to decoder underflow or a missed frame.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000446

V.2.22 org.iso.mpeg.mpegj.resource.MPRendererMediaListener

V.2.22.1 Syntax

```
public interface MPRendererMediaListener extends java.util.EventListener
```

All Superinterfaces:
java.util.EventListener

V.2.22.2 Description

An interface that is called by the renderer when registered.

See Also:
org.iso.mpeg.mpegj.resource.MPRendererEventGenerator

Member Summary	
Methods	
void	mPCompositeMediaHandler(MPRendererMediaEvents)

V.2.22.3 Methods

mPCompositeMediaHandler(MPRendererMediaEvents)
public void
mPCompositeMediaHandler(org.iso.mpeg.mpegj.resource.MPRendererMediaEvents)

Notification of a Renderer event, called by the renderer during decoder underflow or a missed frame.

Parameters:
event - refers to the source of the event.

V.2.23 org.iso.mpeg.mpegj.resource.Renderer

V.2.23.1 Syntax

```
public interface Renderer extends
   org.iso.mpeg.mpegj.resource.MPRendererEventGenerator,
   org.iso.mpeg.mpegj.resource.MPRendererFrameEventGenerator
```

All Superinterfaces:
org.iso.mpeg.mpegj.resource.MPRendererEventGenerator,
org.iso.mpeg.mpegj.resource.MPRendererFrameEventGenerator

V.2.23.2 Description

Interface for the Rendering Engine.

V.2.24 org.iso.mpeg.mpegj.resource.RendererNotFoundException

V.2.24.1 Syntax

```
public class RendererNotFoundException extends org.iso.mpeg.mpegj.MPEGJException
```

java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--**org.iso.mpeg.mpegj.resource.RendererNotFoundException**

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000447

All Implemented Interfaces:
java.io.Serializable

V.2.24.2 Description

The class RendererNotFoundException is a subclass of MPEGJException. This Exception is thrown by the method getRenderer() of Resource Manager.

Member Summary	
Constructors	
	RendererNotFoundException() RendererNotFoundException(String)

V.2.24.3 Constructors

RendererNotFoundException()
```
public  RendererNotFoundException()
```
Constructs an RendererNotFoundException with no specified detail message.

RendererNotFoundException(String)
```
public  RendererNotFoundException(java.lang.String)
```
Constructs an RendererFoundException with a detailed message.

V.2.25 org.iso.mpeg.mpegj.resource.ResourceManager

V.2.25.1 Syntax

```
public class ResourceManager extends java.lang.Object
```

java.lang.Object
 |
 +--**org.iso.mpeg.mpegj.resource.ResourceManager**

V.2.25.2 Description

The resource manager is used for regulation of performance. The Resource Manager API provides a centralized facility for managing resources. It is a collection of a number of classes and interfaces summarized as follows.

Member Summary	
Constructors	
	ResourceManager()
Methods	
CapabilityManager	getCapabilityManager()
void	setDecPriority(MPDecoder, int)
int	getDecPriority(MPDecoder)
MPDecoder	getDecoder(ObjectDescriptor, ESDescriptor)
Renderer	getRenderer()
void	changeDecoder(ObjectDescriptor, ESDescriptor, MPDecoder)
MPDecoder[]	getAvailableDecoder(DecoderType)

V.2.25.3 Constructors

ResourceManager()
```
public  ResourceManager()
```

V.2.25.4 Methods

changeDecoder(ObjectDescriptor, ESDescriptor, MPDecoder)
```
public void changeDecoder(org.iso.mpeg.mpegj.ObjectDescriptor,
org.iso.mpeg.mpegj.ESDescriptor, org.iso.mpeg.mpegj.decoder.MPDecoder)
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000448

Change decoder on the given Object Descriptor and ESDescriptor The underlying implementation is expected to instantiate, attach and start a decoder similar to establishing a new decoder for a BIFS node. All the elementary streams attached to the original decoder is attached to the new decoder.

Throws:
org.iso.mpeg.mpegj.decoder.DecoderNotFoundException - Thrown when the Decoder is not available

org.iso.mpeg.mpegj.scene.BadNodeException - is thrown when the OD does not correspond to a valid node in the scenegraph

getAvailableDecoder(DecoderType)
`public org.iso.mpeg.mpegj.decoder.MPDecoder getAvailableDecoder(org.iso.mpeg.mpegj.decoder.DecoderType)`

Get available decoders given decoder type.

Returns:
returns an array of available decoders that are not used currently.

Throws:
org.iso.mpeg.mpegj.decoder.InvalidDecoderTypeException - Thrown when the DecoderType not valid

getCapabilityManager()
`public org.iso.mpeg.mpegj.resource.CapabilityManager getCapabilityManager()`

Returns the CapabilityManager used to retrieve all the capabilities (static, dynamic, profile) of the terminal.

Returns:
the CapabilityManager if available

Throws:
org.iso.mpeg.mpegj.resource.CapabilityManagerNotFoundException - Thrown when the CapabilityManager is not available (e.g. it is impossible to retrieve information about CPU load, memory etc.)

getDecoder(ObjectDescriptor, ESDescriptor)
`public org.iso.mpeg.mpegj.decoder.MPDecoder getDecoder(org.iso.mpeg.mpegj.ObjectDescriptor, org.iso.mpeg.mpegj.ESDescriptor)`

Get Decoder for the given ObjectDescriptor and ESDescriptor.

Throws:
org.iso.mpeg.mpegj.decoder.DecoderNotFoundException - Thrown when the Decoder is not available

org.iso.mpeg.mpegj.scene.BadNodeException - is thrown when the nodeId does not correspond to a valid node in the scenegraph

getDecPriority(MPDecoder)
`public int getDecPriority(org.iso.mpeg.mpegj.decoder.MPDecoder)`

Get priority of a media elementary stream.

Throws:
org.iso.mpeg.mpegj.decoder.DecoderNotFoundException - Thrown when the Decoder is not available

getRenderer()
`public org.iso.mpeg.mpegj.resource.Renderer getRenderer()`

Get renderer and be ready to add events to renderer.

Throws:
org.iso.mpeg.mpegj.resource.RendererNotFoundException - is thrown if the Renderer is not available

setDecPriority(MPDecoder, int)
`public void setDecPriority(org.iso.mpeg.mpegj.decoder.MPDecoder, int)`

Override/change priority of a media elementary stream.

Throws:
org.iso.mpeg.mpegj.decoder.DecoderNotFoundException - Thrown when the Decoder is not available

org.iso.mpeg.mpegj.decoder.InvalidDecoderPriorityException - thrown when the prirority is invalid

V.2.26 org.iso.mpeg.mpegj.resource.ResourceManagerNotFoundException

V.2.26.1 Syntax

```
public class ResourceManagerNotFoundException extends
   org.iso.mpeg.mpegj.MPEGJException
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000449

java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--**org.iso.mpeg.mpegj.resource.ResourceManagerNotFoundException**

All Implemented Interfaces:
java.io.Serializable

V.2.26.2 Description

The class ResourceManagerNotFoundException is a subclass of MPEGJException. This Exception is thrown by the method getRM() of MpegjTerminal.

Member Summary	
Constructors	ResourceManagerNotFoundException() ResourceManagerNotFoundException(String)

V.2.26.3 Constructors

ResourceManagerNotFoundException()
`public ResourceManagerNotFoundException()`

Constructs an ResourceManagerNotFoundException with no specified detail message.

ResourceManagerNotFoundException(String)
`public ResourceManagerNotFoundException(java.lang.String)`

Constructs an ResourceManagerNotFoundException with a detailed message.

V.2.27 org.iso.mpeg.mpegj.resource.StaticCapability

V.2.27.1 Syntax

```
public interface StaticCapability
```

All Known Implementing Classes:
org.iso.mpeg.mpegj.resource.CapabilityManager

V.2.27.2 Description

This interface is used to provide developers a simple way to access basic dynamic terminal capabilities. A capability is considered static if its value cannot change at runtime. For instance the audio card type is a static capability.

Member Summary	
Methods	
int	getNumCPUs()
int	getCPUSpeed(int)
String	getCPUType(int)
String	getMouseType()
int	getDisplayColorDepth()
String	getDisplayType()
String	getKeyboardType()
String	getNetworkType()
int	getNumParallelPorts()
int	getNumSerialPorts()

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000450

Dimension	getScreenSize()
short	getScreenDepth()
short	getScreenResolution()
String	getOSLanguage()
String	getOSType()
String	getTerminalArchitecture()
String[]	getAudioDrivers()
String[]	getMIDIDrivers()
String[]	getVideoDrivers()
String	getModemType()

V.2.27.3 Methods

getAudioDrivers()
`public java.lang.String[] getAudioDrivers()`

Terminal Audio Enc/Dec

Returns:
the available audio drivers.

getCPUSpeed(int)
`public int getCPUSpeed(int)`

Terminal CPU Speed

Parameters:
`idx` - the index of the video card (idx >= 0)

Returns:
the speed (Mhz) of a terminal CPU.

Throws:
`IllegalArgumentException` - the index is either negative or out of range.

getCPUType(int)
`public java.lang.String getCPUType(int)`

Terminal CPU type (Pentium for instance)

Parameters:
`idx` - the index of the CPU (idx >= 0)

Returns:
the type of CPU

Throws:
`IllegalArgumentException` - the index is either negative or out of range.

getDisplayColorDepth()
`public int getDisplayColorDepth()`

Terminal Display Color Depth

Returns:
the display color depth (bits per pixel)

getDisplayType()
`public java.lang.String getDisplayType()`

Terminal Display Type (e.g. VGA)

Returns:
the display type

getKeyboardType()
`public java.lang.String getKeyboardType()`

Terminal Keyboard Type

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000451

Returns:
the keyboard type, NULL if the keyboard is not present.

getMIDIDrivers()
`public java.lang.String[] getMIDIDrivers()`

Terminal MIDI Enc/Dec

Returns:
the available MIDI drivers.

getModemType()
`public java.lang.String getModemType()`

Terminal modem type [lsInternalModem]

Returns:
the modem speed ("28K", "56K" etc.) or NULL is the modem is not installed.

getMouseType()
`public java.lang.String getMouseType()`

Terminal Mouse Type

Returns:
the mouse type

getNetworkType()
`public java.lang.String getNetworkType()`

Terminal Network Type

Returns:
the network type.

getNumCPUs()
`public int getNumCPUs()`

Number of Terminal CPUs

Returns:
the number of CPUs that equip the terminal

getNumParallelPorts()
`public int getNumParallelPorts()`

Terminal parallel IEEE-1284 port(s)

Returns:
the number of parallel ports present on the terminal.

getNumSerialPorts()
`public int getNumSerialPorts()`

Terminal serial RS-232 port(s)

Returns:
the number of serial ports present on the terminal.

getOSLanguage()
`public java.lang.String getOSLanguage()`

Terminal OS language [lsLanguage]

Returns:
the language of the OS version installed on the terminal.

getOSType()
`public java.lang.String getOSType()`

Terminal OS type (for instance JavaPC, Windows, MacOS) [lsOpSys/lsSysType]

Returns:
the OS type installed on the terminal.

getScreenDepth()
`public short getScreenDepth()`

Terminal screen depth

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000452

Returns:
the screen depth (bits/inch)

getScreenResolution()
`public short getScreenResolution()`

Terminal screen resolution

Returns:
the screen resolution (dots/inch)

getScreenSize()
`public org.iso.mpeg.mpegj.resource.Dimension getScreenSize()`

Terminal screen size

Returns:
the screen size (pixel x pixel)

getTerminalArchitecture()
`public java.lang.String getTerminalArchitecture()`

Terminal architecture (for instance Alpha, x86)

Returns:
the terminal architecture type.

getVideoDrivers()
`public java.lang.String[] getVideoDrivers()`

Terminal Video Enc/Dec

Returns:
the available video drivers.

V.2.28 org.iso.mpeg.mpegj.resource.TerminalProfileManager

V.2.28.1 Syntax

```
public interface TerminalProfileManager
```

All Known Implementing Classes:
org.iso.mpeg.mpegj.resource.CapabilityManager

V.2.28.2 Description

Thisinterface allows applications to query the terminal profiles. Profiles are specified in the document ISO/IEC 14496-1.

Member Summary	
Methods	
short	getVisualProfile()
short	getAudioProfile()
short	getSceneDescriptionProfile()
short	getODProfile()
short	getGraphicsProfile()
short	getMPEGJProfile()

V.2.28.3 Methods

getAudioProfile()
`public short getAudioProfile()`

This method returns the supported audio profile The value of the return value is as given in audioProfileLevelIndication Table in ISO/ IEC 14496-1

Returns:
the audio description profile and level.

getGraphicsProfile()
`public short getGraphicsProfile()`

This method returns the supported scene description profile The value of the return value is as given in graphicsProfileLevelIndication Table in ISO/IEC 14496-1

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000453

Returns:
the supported Graphics profile and level.

getMPEGJProfile()
`public short getMPEGJProfile()`

This method returns the supported scene description profile The value of the return value is as given in MPEG-JProfileLevelIndication Table in ISO/IEC 14496-1

Returns:
the supported MPEG-J profile and level.

getODProfile()
`public short getODProfile()`

This method returns the supported scene description profile The value of the return value is as given in ODProfileLevelIndication Table in ISO/IEC 14496-1

Returns:
the supported OD profile and level.

getSceneDescriptionProfile()
`public short getSceneDescriptionProfile()`

This method returns the supported scene description profile The value of the return value is as given in sceneProfileLevelIndication Table in ISO/IEC 14496-1

Returns:
the supported sceen description profile and level.

getVisualProfile()
`public short getVisualProfile()`

This method returns the supported visual profile The value of the return value is as given in visualProfileLevelIndication Table in ISO/ IEC 14496-1

Returns:
the supported visual profile and level.

V.3 package org.iso.mpeg.mpegj.decoder

V.3.1 Description

Class Summary	
Interfaces	
MPDecoder	This is the base interface for media decoders.
DecoderType	Interface for decoder types.
Exceptions	
MediaDecoderException	The class MediaDecoderException is a subclass of MPEGJException and is the parent of all ScenManagerExceptions.
InvalidDecoderLevelException	The class InvalidDecoderLevelException is a subclass of MediaDecoderException.
DecoderNotFoundException	The class DecoderNotFoundException is a subclass of MediaDecoderException.
InvalidDecoderPriorityException	The class InvalidDecoderPriorityException is a subclass of MediaDecoderException.
InvalidDecoderSpeedException	The class InvalidDecoderSpeedException is a subclass of MediaDecoderException.
InvalidDecoderModeException	The class InvalidDecoderModeException is a subclass of MediaDecoderException.
DecoderNotRunningException	The class DecoderNotRunningException is a subclass of MediaDecoderException.
DecoderAlreadyAttachedException	The class DecoderAlreadyAttachedException is a subclass of MediaDecoderException.
DecoderNotAttachedException	The class DecoderNotAttachedException is a subclass of MediaDecoderException.
DecoderAlreadyRunningException	The class DecoderAlreadyRunningException is a subclass of MediaDecoderException.
InvalidDecoderTypeException	The class InvalidDecoderTypeException is a subclass of MediaDecoderException.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000454

V.3.2 org.iso.mpeg.mpegj.decoder.DecoderAlreadyAttachedException

V.3.2.1 Syntax

```
public class DecoderAlreadyAttachedException extends
   org.iso.mpeg.mpegj.decoder.MediaDecoderException
```

java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--org.iso.mpeg.mpegj.decoder.MediaDecoderException
                 |
                 +--**org.iso.mpeg.mpegj.decoder.DecoderAlreadyAttachedException**

All Implemented Interfaces:
java.io.Serializable

V.3.2.2 Description

The class DecoderAlreadyAttachedException is a subclass of MediaDecoderException. This Exception is thrown when a Decoder that is already attached is attempted to attach again.

Member Summary	
Constructors	DecoderAlreadyAttachedException() DecoderAlreadyAttachedException(String)

V.3.2.3 Constructors

DecoderAlreadyAttachedException()
```
public  DecoderAlreadyAttachedException()
```
Constructs an DecoderAlreadyAttachedException with no specified detail message.

DecoderAlreadyAttachedException(String)
```
public  DecoderAlreadyAttachedException(java.lang.String)
```
Constructs an DecoderAttachedException with a detailed message.

V.3.3 org.iso.mpeg.mpegj.decoder.DecoderAlreadyRunningException

V.3.3.1 Syntax

```
public class DecoderAlreadyRunningException extends
   org.iso.mpeg.mpegj.decoder.MediaDecoderException
```

java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000455

```
          +--org.iso.mpeg.mpegj.decoder.MediaDecoderException
              |
              +--org.iso.mpeg.mpegj.decoder.DecoderAlreadyRunningException
```

All Implemented Interfaces:
java.io.Serializable

V.3.3.2 Description

The class DecoderAlreadyRunningException is a subclass of MediaDecoderException. This Exception is thrown when a Decoder that is already running is attempted to start.

Member Summary	
Constructors	DecoderAlreadyRunningException() DecoderAlreadyRunningException(String)

V.3.3.3 Constructors

DecoderAlreadyRunningException()
`public DecoderAlreadyRunningException()`

Constructs an DecoderAlreadyRunningException with no specified detail message.

DecoderAlreadyRunningException(String)
`public DecoderAlreadyRunningException(java.lang.String)`

Constructs an DecoderRunningException with a detailed message.

V.3.4 org.iso.mpeg.mpegj.decoder.DecoderNotAttachedException

V.3.4.1 Syntax

```
public class DecoderNotAttachedException extends
   org.iso.mpeg.mpegj.decoder.MediaDecoderException
```

```
java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--org.iso.mpeg.mpegj.decoder.MediaDecoderException
                 |
                 +--org.iso.mpeg.mpegj.decoder.DecoderNotAttachedException
```

All Implemented Interfaces:
java.io.Serializable

V.3.4.2 Description

The class DecoderNotAttachedException is a subclass of MediaDecoderException. This Exception is thrown when a Decoder that is not attached is attempted detach.

Member Summary	
Constructors	DecoderNotAttachedException() DecoderNotAttachedException(String)

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000456

V.3.4.3 Constructors

DecoderNotAttachedException()
`public DecoderNotAttachedException()`

Constructs an DecoderNotAttachedException with no specified detail message.

DecoderNotAttachedException(String)
`public DecoderNotAttachedException(java.lang.String)`

Constructs an DecoderAttachedException with a detailed message.

V.3.5 org.iso.mpeg.mpegj.decoder.DecoderNotFoundException

V.3.5.1 Syntax

```
public class DecoderNotFoundException extends
   org.iso.mpeg.mpegj.decoder.MediaDecoderException
```

```
java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--org.iso.mpeg.mpegj.decoder.MediaDecoderException
                 |
                 +--org.iso.mpeg.mpegj.decoder.DecoderNotFoundException
```

All Implemented Interfaces:
java.io.Serializable

V.3.5.2 Description

The class DecoderNotFoundException is a subclass of MediaDecoderException. This Exception is thrown by the method getDecoder() of ResourceManager.

Member Summary	
Constructors	DecoderNotFoundException() DecoderNotFoundException(String)

V.3.5.3 Constructors

DecoderNotFoundException()
`public DecoderNotFoundException()`

Constructs an DecoderNotFoundException with no specified detail message.

DecoderNotFoundException(String)
`public DecoderNotFoundException(java.lang.String)`

Constructs an DecoderFoundException with a detailed message.

V.3.6 org.iso.mpeg.mpegj.decoder.DecoderNotRunningException

V.3.6.1 Syntax

```
public class DecoderNotRunningException extends
   org.iso.mpeg.mpegj.decoder.MediaDecoderException
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000457

java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--org.iso.mpeg.mpegj.decoder.MediaDecoderException
                 |
                 +--**org.iso.mpeg.mpegj.decoder.DecoderNotRunningException**

All Implemented Interfaces:
java.io.Serializable

V.3.6.2 Description

The class DecoderNotRunningException is a subclass of MediaDecoderException. This Exception is thrown when a Decoder that is not running is attempted to pause or stop.

Member Summary	
Constructors	DecoderNotRunningException() DecoderNotRunningException(String)

V.3.6.3 Constructors

DecoderNotRunningException()
`public DecoderNotRunningException()`

Constructs an DecoderNotRunningException with no specified detail message.

DecoderNotRunningException(String)
`public DecoderNotRunningException(java.lang.String)`

Constructs an DecoderRunningException with a detailed message.

V.3.7 org.iso.mpeg.mpegj.decoder.DecoderType

V.3.7.1 Syntax

```
public interface DecoderType
```

V.3.7.2 Description

Interface for decoder types.

Member Summary	
Fields	
int	MPEG1V
int	MPEG1A
int	MPEG2TS
int	MPEG2V
int	MPEG2A
int	MPEG4S
int	MPEG4V
int	MPEG4VS
int	MPEG4T
int	MPEG4TS
int	MPEG4M
int	MPEG4F
int	MPEG4FS
int	MPEG4A

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000458

	int	MPEG4AS
	int	MPEG4TT
	int	MPEG4SA
Methods		
	int	getDecoderType()

V.3.7.3 Fields

MPEG1A
`public static final int MPEG1A`

MPEG1 audio. MPEG1A = 1

MPEG1V
`public static final int MPEG1V`

MPEG1 video. MPEG1V = 0

MPEG2A
`public static final int MPEG2A`

MPEG2 audio. MPEG2A = 4

MPEG2TS
`public static final int MPEG2TS`

MPEG2 transport stream. MPEG2TS = 2

MPEG2V
`public static final int MPEG2V`

MPEG2 video. MPEG2V = 3

MPEG4A
`public static final int MPEG4A`

MPEG4 audio. MPEG4A = 13

MPEG4AS
`public static final int MPEG4AS`

MPEG4 scalable audio MPEG4AS = 14

MPEG4F
`public static final int MPEG4F`

MPEG4 face decoder. MPEG4F = 11

MPEG4FS
`public static final int MPEG4FS`

MPEG4 scalable face decoder. MPEG4FS = 12

MPEG4M
`public static final int MPEG4M`

MPEG4M = 10

MPEG4S
`public static final int MPEG4S`

MPEG4 systems. MPEG4S = 5

MPEG4SA
`public static final int MPEG4SA`

MPEG4 structured audio. MPEG4SA = 16

MPEG4T
`public static final int MPEG4T`

MPEG4 image texture. MPEG4T = 8

MPEG4TS
`public static final int MPEG4TS`

MPEG4 scalable texture. MPEG4TS = 9

MPEG4TT
`public static final int MPEG4TT`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000459

MPEG4 TTS audio MPEG4TT = 15

MPEG4V
`public static final int MPEG4V`

MPEG4 video. MPEG4V = 6

MPEG4VS
`public static final int MPEG4VS`

MPEG4 scalable video. MPEG4VS = 7

V.3.7.4 Methods

getDecoderType()
`public int getDecoderType()`

Get decoder type.

V.3.8 org.iso.mpeg.mpegj.decoder.InvalidDecoderLevelException

V.3.8.1 Syntax

```
public class InvalidDecoderLevelException extends
   org.iso.mpeg.mpegj.decoder.MediaDecoderException
```

java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--org.iso.mpeg.mpegj.decoder.MediaDecoderException
                 |
                 +--**org.iso.mpeg.mpegj.decoder.InvalidDecoderLevelException**

All Implemented Interfaces:
java.io.Serializable

V.3.8.2 Description

The class InvalidDecoderLevelException is a subclass of MediaDecoderException. This Exception is thrown by the when a decoder Level is attempted to be set to an invalid value (by the Media APIs).

Member Summary	
Constructors	`InvalidDecoderLevelException()` `InvalidDecoderLevelException(String)`

V.3.8.3 Constructors

InvalidDecoderLevelException()
`public  InvalidDecoderLevelException()`

Constructs an InvalidDecoderLevelException with no specified detail message.

InvalidDecoderLevelException(String)
`public  InvalidDecoderLevelException(java.lang.String)`

Constructs an InvalidDecoderLevelException with a detailed message.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000460

V.3.9 org.iso.mpeg.mpegj.decoder.InvalidDecoderModeException

V.3.9.1 Syntax

```
public class InvalidDecoderModeException extends
   org.iso.mpeg.mpegj.decoder.MediaDecoderException
```

```
java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--org.iso.mpeg.mpegj.decoder.MediaDecoderException
                 |
                 +--org.iso.mpeg.mpegj.decoder.InvalidDecoderModeException
```

All Implemented Interfaces:
java.io.Serializable

V.3.9.2 Description

The class InvalidDecoderModeException is a subclass of MediaDecoderException. This Exception is thrown by the when a decoder mode is attempted to be set to an invalid value (by the Media APIs).

Member Summary	
Constructors	InvalidDecoderModeException() InvalidDecoderModeException(String)

V.3.9.3 Constructors

InvalidDecoderModeException()
`public InvalidDecoderModeException()`

Constructs an InvalidDecoderModeException with no specified detail message.

InvalidDecoderModeException(String)
`public InvalidDecoderModeException(java.lang.String)`

Constructs an InvalidDecoderModeException with a detailed message.

V.3.10 org.iso.mpeg.mpegj.decoder.InvalidDecoderPriorityException

V.3.10.1 Syntax

```
public class InvalidDecoderPriorityException extends
   org.iso.mpeg.mpegj.decoder.MediaDecoderException
```

```
java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--org.iso.mpeg.mpegj.decoder.MediaDecoderException
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000461

```
                |
                +--org.iso.mpeg.mpegj.decoder.InvalidDecoderPriorityException
```

All Implemented Interfaces:
java.io.Serializable

V.3.10.2 Description

The class InvalidDecoderPriorityException is a subclass of MediaDecoderException. This Exception is thrown by the when a decoder priority is attempted to be set to an invalid value (by the Resource Manager).

Member Summary	
Constructors	`InvalidDecoderPriorityException()` `InvalidDecoderPriorityException(String)`

V.3.10.3 Constructors

InvalidDecoderPriorityException()
`public InvalidDecoderPriorityException()`

Constructs an InvalidDecoderPriorityException with no specified detail message.

InvalidDecoderPriorityException(String)
`public InvalidDecoderPriorityException(java.lang.String)`

Constructs an InvalidDecoderPriorityException with a detailed message.

V.3.11 org.iso.mpeg.mpegj.decoder.InvalidDecoderSpeedException

V.3.11.1 Syntax

```
public class InvalidDecoderSpeedException extends
   org.iso.mpeg.mpegj.decoder.MediaDecoderException
```

```
java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--org.iso.mpeg.mpegj.decoder.MediaDecoderException
                 |
                 +--org.iso.mpeg.mpegj.decoder.InvalidDecoderSpeedException
```

All Implemented Interfaces:
java.io.Serializable

V.3.11.2 Description

The class InvalidDecoderSpeedException is a subclass of MediaDecoderException. This Exception is thrown by the when a decoder mode is attempted to be set to an invalid value (by the Media APIs).

Member Summary	
Constructors	`InvalidDecoderSpeedException()` `InvalidDecoderSpeedException(String)`

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000462

V.3.11.3 Constructors

InvalidDecoderSpeedException()
`public InvalidDecoderSpeedException()`

Constructs an InvalidDecoderSpeedException with no specified detail message.

InvalidDecoderSpeedException(String)
`public InvalidDecoderSpeedException(java.lang.String)`

Constructs an InvalidDecoderSpeedException with a detailed message.

V.3.12 org.iso.mpeg.mpegj.decoder.InvalidDecoderTypeException

V.3.12.1 Syntax

```
public class InvalidDecoderTypeException extends
   org.iso.mpeg.mpegj.decoder.MediaDecoderException
```

java.lang.Object
 |
 +--java.lang.Throwable
  |
  +--java.lang.Exception
  |
  +--org.iso.mpeg.mpegj.MPEGJException
  |
  +--org.iso.mpeg.mpegj.decoder.MediaDecoderException
  |
  +--**org.iso.mpeg.mpegj.decoder.InvalidDecoderTypeException**

All Implemented Interfaces:
java.io.Serializable

V.3.12.2 Description

The class InvalidDecoderTypeException is a subclass of MediaDecoderException. This Exception is thrown by the when a decoder mode is attempted to be set to an invalid value (by the Resource Manager).

Member Summary	
Constructors	InvalidDecoderTypeException() InvalidDecoderTypeException(String)

V.3.12.3 Constructors

InvalidDecoderTypeException()
`public InvalidDecoderTypeException()`

Constructs an InvalidDecoderTypeException with no specified detail message.

InvalidDecoderTypeException(String)
`public InvalidDecoderTypeException(java.lang.String)`

Constructs an InvalidDecoderTypeException with a detailed message.

V.3.13 org.iso.mpeg.mpegj.decoder.MediaDecoderException

V.3.13.1 Syntax

```
public class MediaDecoderException extends org.iso.mpeg.mpegj.MPEGJException
```

java.lang.Object

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000463

```
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--org.iso.mpeg.mpegj.decoder.MediaDecoderException
```

Direct Known Subclasses:
org.iso.mpeg.mpegj.decoder.DecoderAlreadyAttachedException,
org.iso.mpeg.mpegj.decoder.DecoderAlreadyRunningException,
org.iso.mpeg.mpegj.decoder.DecoderNotAttachedException,
org.iso.mpeg.mpegj.decoder.DecoderNotFoundException,
org.iso.mpeg.mpegj.decoder.DecoderNotRunningException,
org.iso.mpeg.mpegj.decoder.InvalidDecoderLevelException,
org.iso.mpeg.mpegj.decoder.InvalidDecoderModeException,
org.iso.mpeg.mpegj.decoder.InvalidDecoderPriorityException,
org.iso.mpeg.mpegj.decoder.InvalidDecoderSpeedException,
org.iso.mpeg.mpegj.decoder.InvalidDecoderTypeException

All Implemented Interfaces:
java.io.Serializable

V.3.13.2 Description

The class MediaDecoderException is a subclass of MPEGJException and is the parent of all ScenManagerExceptions. This is thrown by the MediaDecoder APIs.

Member Summary	
Constructors	MediaDecoderException() MediaDecoderException(String)

V.3.13.3 Constructors

MediaDecoderException()
```
public  MediaDecoderException()
```
Constructs an MediaDecoderException with no specified detail message.

MediaDecoderException(String)
```
public  MediaDecoderException(java.lang.String)
```
Constructs an MediaDecoderException with a detailed message.

V.3.14 org.iso.mpeg.mpegj.decoder.MPDecoder

V.3.14.1 Syntax

```
public interface MPDecoder extends
   org.iso.mpeg.mpegj.resource.MPDecoderEventGenerator
```

All Superinterfaces:
org.iso.mpeg.mpegj.resource.MPDecoderEventGenerator

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000464

V.3.14.2 Description

This is the base interface for media decoders.

Member Summary	
Methods	
void	start()
void	stop()
void	pause()
void	resume()
void	attach(ESDescriptor)
void	detach()
DecoderType	getType()
String	getVendor()
int	getInstance()
boolean	isPauseable()
ESDescriptor	getESDescriptor()

V.3.14.3 Methods

attach(ESDescriptor)
`public void attach(org.iso.mpeg.mpegj.ESDescriptor)`

Attach a decoder for decoding of data.

detach()
`public void detach()`

Detach a decoder already decoding data.

getESDescriptor()
`public org.iso.mpeg.mpegj.ESDescriptor getESDescriptor()`

Get the attached ESDescriptor

getInstance()
`public int getInstance()`

Get instance number.

getType()
`public org.iso.mpeg.mpegj.decoder.DecoderType getType()`

Get decoder type.

getVendor()
`public java.lang.String getVendor()`

Get decoder vendor name.

isPauseable()
`public boolean isPauseable()`

Determines if the decoder is pauseable.

pause()
`public void pause()`

Pause decoding of data.

resume()
`public void resume()`

Resume decoding of data from paused instant.

start()
`public void start()`

Start decoding of data. This method should not be called for normal decoding. Because normal decoding starts automatically from the underlying BIFS construct

stop()
`public void stop()`

Stop decoding of data.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000465

V.4 package org.iso.mpeg.mpegj.net

V.4.1 Description

Class Summary	
Interfaces	
NetworkManager	
ChannelDescriptor	This interface allows to retrieve all the information about a channel in a specified service session.
DMIFMonitor	
ChannelController	
ServiceSessionDe-scriptor	This interface allows to retrieve information about a specific service session.
QoSDescriptor	
Classes	
Qualifier	
ChannelsDescriptor	
QoSData	
QualifierTag	This class contains all the constants that may be used to describe a Qualifier metric (see the DMIF specification).
Exceptions	
AccessLayerException	

V.4.2 org.iso.mpeg.mpegj.net.AccessLayerException

V.4.2.1 Syntax

```
public class AccessLayerException extends org.iso.mpeg.mpegj.MPEGJException
```

java.lang.Object
 |
 +--java.lang.Throwable
    |
    +--java.lang.Exception
       |
       +--org.iso.mpeg.mpegj.MPEGJException
          |
          +--**org.iso.mpeg.mpegj.net.AccessLayerException**

All Implemented Interfaces:
java.io.Serializable

V.4.2.2 Description

Member Summary		
Fields		
	int	INVALID_SERVICE_SESSION_ID
	int	INVALID_STREAMPRIORITY
	int	INVALID_QOSDATA
	int	INVALID_QUALIFIERTAG
	int	BAD_URL_STRING
	int	INVALID_QUALIFIER_VALUE_TYPE
	int	ACCESS_LAYER_NOT_FOUND
Constructors		
		AccessLayerException()
		AccessLayerException(int)
Methods		
	int	getValue()

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000466

V.4.2.3 Fields

ACCESS_LAYER_NOT_FOUND
`public static final int ACCESS_LAYER_NOT_FOUND`

BAD_URL_STRING
`public static final int BAD_URL_STRING`

INVALID_QOSDATA
`public static final int INVALID_QOSDATA`

INVALID_QUALIFIER_VALUE_TYPE
`public static final int INVALID_QUALIFIER_VALUE_TYPE`

INVALID_QUALIFIERTAG
`public static final int INVALID_QUALIFIERTAG`

INVALID_SERVICE_SESSION_ID
`public static final int INVALID_SERVICE_SESSION_ID`

INVALID_STREAMPRIORITY
`public static final int INVALID_STREAMPRIORITY`

V.4.2.4 Constructors

AccessLayerException()
`public AccessLayerException()`

This constructor allows a method to throw an exception related to the underlying Access Layer.

AccessLayerException(int)
`public AccessLayerException(int)`

This constructor allows a method to throw an exception related to the underlying Access Layer.

Parameters:
`Gripe` - input parameter int, allows the application to know the reason of the exception. It can assume a constant value defined in this class.

V.4.2.5 Methods

getValue()
`public int getValue()`

This method enables the application to know the illegal parameter that caused the AccessLayerException

Returns:
An integer value defined in the class, indicating the wrong input parameter

V.4.3 org.iso.mpeg.mpegj.net.ChannelController

V.4.3.1 Syntax

```
public interface ChannelController
```

V.4.3.2 Description

Member Summary	
Methods	
void	enable(long, long)
void	disable(long, long)

V.4.3.3 Methods

disable(long, long)
`public void disable(long, long)`

This method allows to disable a specified channel in a specified service session. This means that the particular channel won't be able to accept data from the Access Layer.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000467

Parameters:
`serviceSessionID:` - specifies the service session identifier

`channelID:` - specifies a particular channel (by gining a channel identifier) within the given service session.

enable(long, long)
`public void enable(long, long)`

This method allows to enable a specified channel in a specified service session. This means that the particular channel will be able to accept data from the Access Layer.

Parameters:
`serviceSessionID:` - specifies the service session identifier

`channelID:` - specifies a particular channel (by gining a channel identifier) within the given service session.

V.4.4 org.iso.mpeg.mpegj.net.ChannelDescriptor

V.4.4.1 Syntax

```
public interface ChannelDescriptor
```

V.4.4.2 Description

This interface allows to retrieve all the information about a channel in a specified service session.

Member Summary	
Methods	
long	getServiceSessionID()
long	getChannelID()
int	getAURate()
int	getError()
int	getStatus()
QoSDescriptor	getQoS()

V.4.4.3 Methods

getAURate()
`public int getAURate()`

This method retrieves the actual Access Unit rate of the channel. This value may change over the time.

getChannelID()
`public long getChannelID()`

This method retrieves the channel identifier of the channel associated to the descriptor. Using this value it is possible to obtain both the Object Descriptor Identifier and the Elementary stream identifier associated to the channel: OD ID: (channelID >> 6) ES ID: (channelId & 0x0000003F)

getError()
`public int getError()`

This method retrieves information about error conditions associated to the channel (see DMIF specification). This value may change over the time.

getQoS()
`public org.iso.mpeg.mpegj.net.QoSDescriptor getQoS()`

This method retrieves information about the Quality of service associated to the channel (see QualifierTag).

getServiceSessionID()
`public long getServiceSessionID()`

This method retrieves the service session identifier associated to the specific channel

getStatus()
`public int getStatus()`

This method retrieves information about the status of the channel (see DMIF specification). This value may change over the time.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000468

V.4.5 org.iso.mpeg.mpegj.net.ChannelsDescriptor

V.4.5.1 Syntax

```
public class ChannelsDescriptor extends java.lang.Object implements
   java.io.Serializable
```

java.lang.Object
 |
 +--**org.iso.mpeg.mpegj.net.ChannelsDescriptor**

All Implemented Interfaces:
java.io.Serializable

V.4.5.2 Description

Member Summary	
Fields	
int	auRate
long	channelId
long	serviceSessionId
int	error
int	status
QoSData	qos
Constructors	
	ChannelsDescriptor(long)

V.4.5.3 Fields

auRate
`public int auRate`

channelId
`public long channelId`

error
`public int error`

qos
`public org.iso.mpeg.mpegj.net.QoSData qos`

serviceSessionId
`public long serviceSessionId`

status
`public int status`

V.4.5.4 Constructors

ChannelsDescriptor(long)
`public  ChannelsDescriptor(long)`

V.4.6 org.iso.mpeg.mpegj.net.DMIFMonitor

V.4.6.1 Syntax

```
public interface DMIFMonitor
```

V.4.6.2 Description

Member Summary	
Methods	
Vector	getServiceSessionDescriptors()
Vector	getChannelDescriptors(long)

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000469

V.4.6.3 Methods

getChannelDescriptors(long)
`public java.util.Vector getChannelDescriptors(long)`

This method returns a Vector of object ChannelsDescriptors that contains informations about the channels existing in a specified session. The returned value refers to entities that may change over the time.

getServiceSessionDescriptors()
`public java.util.Vector getServiceSessionDescriptors()`

This method returns a Vector of object SessionDescriptors that contains information about the existing sessions. The returned value refers to entities that may change over the time.

V.4.7 org.iso.mpeg.mpegj.net.NetworkManager

V.4.7.1 Syntax

```
public interface NetworkManager
```

V.4.7.2 Description

Member Summary	
Methods	
DMIFMonitor	getDMIFMonitor()
ChannelController	getChannelController()

V.4.7.3 Methods

getChannelController()
`public org.iso.mpeg.mpegj.net.ChannelController getChannelController()`

This method returns the ChannelController that is used to enable and disable channels

Returns:
the ChannelControllerif available

getDMIFMonitor()
`public org.iso.mpeg.mpegj.net.DMIFMonitor getDMIFMonitor()`

This method returns the DMIFMonitor that is used to retrieve all the data associated with the DMIF

Returns:
the DMIFMonitor if available

V.4.8 org.iso.mpeg.mpegj.net.QoSData

V.4.8.1 Syntax

```
public class QoSData extends java.lang.Object implements java.io.Serializable
```

java.lang.Object
|
+--**org.iso.mpeg.mpegj.net.QoSData**

All Implemented Interfaces:
java.io.Serializable

V.4.8.2 Description

Member Summary	
Fields	
byte	qualifierCount
Vector	qualifiers
Constructors	
	QoSData()

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000470

V.4.8.3 Fields

qualifierCount
`public byte qualifierCount`

qualifiers
`public java.util.Vector qualifiers`

This attributes is an array of Qualifiers objects

V.4.8.4 Constructors

QoSData()
`public  QoSData()`

V.4.9 org.iso.mpeg.mpegj.net.QoSDescriptor

V.4.9.1 Syntax

`public interface QoSDescriptor`

V.4.9.2 Description

Member Summary	
Methods	
byte	getQualifierCount()
Vector	getQualifiers()

V.4.9.3 Methods

getQualifierCount()
`public byte getQualifierCount()`

getQualifiers()
`public java.util.Vector getQualifiers()`

This attributes is an array of Qualifiers objects

V.4.10 org.iso.mpeg.mpegj.net.Qualifier

V.4.10.1 Syntax

`public class Qualifier extends java.lang.Object implements java.io.Serializable`

java.lang.Object
 |
 +--**org.iso.mpeg.mpegj.net.Qualifier**

All Implemented Interfaces:
java.io.Serializable

V.4.10.2 Description

Member Summary	
Fields	
byte	metric
Number	tagValue
Constructors	
	Qualifier()

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000471

V.4.10.3 Fields

metric
`public byte metric`

This attribute identifies the QoS metric type.

tagValue
`public java.lang.Number tagValue`

This attribute contains the value of the specified metric type

V.4.10.4 Constructors

Qualifier()
`public  Qualifier()`

V.4.11 org.iso.mpeg.mpegj.net.QualifierTag

V.4.11.1 Syntax

```
public class QualifierTag extends java.lang.Object
```

java.lang.Object
 |
 +--**org.iso.mpeg.mpegj.net.QualifierTag**

V.4.11.2 Description

This class contains all the constants that may be used to describe a Qualifier metric (see the DMIF specification).

Member Summary	
Fields	
byte	PRIORITY
byte	MAX_DELAY
byte	AVG_DELAY
byte	LOSS_PROB
byte	MAX_GAP_LOSS
byte	MAX_AU_SIZE
byte	MAX_AU_RATE
byte	AVG_AU_SIZE
Constructors	
	QualifierTag()

V.4.11.3 Fields

AVG_AU_SIZE
`public static final byte AVG_AU_SIZE`

AVG_AU_SIZE = 0x43

AVG_DELAY
`public static final byte AVG_DELAY`

AVG_DELAY = 0x12

LOSS_PROB
`public static final byte LOSS_PROB`

LOSS_PROB = 0x13

MAX_AU_RATE
`public static final byte MAX_AU_RATE`

MAX_AU_RATE = 0x42

MAX_AU_SIZE
`public static final byte MAX_AU_SIZE`

MAX_AU_SIZE = 0x41

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000472

MAX_DELAY
`public static final byte MAX_DELAY`

MAX_DELAY = 0x11

MAX_GAP_LOSS
`public static final byte MAX_GAP_LOSS`

MAX_GAP_LOSS = 0x14

PRIORITY
`public static final byte PRIORITY`

PRIORITY = 0x01

V.4.11.4 Constructors

QualifierTag()
`public  QualifierTag()`

V.4.12 org.iso.mpeg.mpegj.net.ServiceSessionDescriptor

V.4.12.1 Syntax

```
public interface ServiceSessionDescriptor
```

V.4.12.2 Description

This interface allows to retrieve information about a specific service session.

Member Summary	
Methods	
String	getURL()
long	getServiceSessionID()
int	getNumberOfChannels()

V.4.12.3 Methods

getNumberOfChannels()
`public int getNumberOfChannels()`

This method returns the number of the opened channels associated to the service session.

getServiceSessionID()
`public long getServiceSessionID()`

This method returns the service session identifier associated to the service session.

getURL()
`public java.lang.String getURL()`

This method returns a string describing the URL associated to the service session.

V.5 package org.iso.mpeg.mpegj.scene

V.5.1 Description

Class Summary	
Interfaces	
Scene	An interface that acts as a proxy for a BIFS scene.
Node	An interface that acts as a proxy for a BIFS node in the scene graph.
SceneListener	An interface that allows monitoring of changes to a BIFS scene graph.
SceneListener.Message	The message numbers used in the notify method.
MFStringFieldValue	An interface used for obtaining MFString values.
MFFloatFieldValue	An interface used for obtaining MFFloat values.
MFFieldValue	A tagging interface used to classify FieldValue objects that return compound values.
SFStringFieldValue	An interface used for obtaining SFString values.
SFVec2fFieldValue	An interface used for obtaining SFVec2f values.
NodeType	An interface defining constants for node types.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000473

MFTimeFieldValue	An interface used for obtaining MFTime values.
SFNodeFieldValue	An interface used for obtaining SFNode values.
NodeValue	An interface for node values that either act as a proxy for a node in the BIFS scene, or specify the creation of a new node.
FieldValue	A tagging interface used to identify objects that can return field values.
Field	An interface with inner classes defining the constants for the field defIDs of each node.
Field.Anchor	An interface defining constants for the field defIDs of the Anchor node.
Field.AnimationStream	An interface defining constants for the field defIDs of the AnimationStream node.
Field.Appearance	An interface defining constants for the field defIDs of the Appearance node.
Field.AudioBuffer	An interface defining constants for the field defIDs of the AudioBuffer node.
Field.AudioClip	An interface defining constants for the field defIDs of the AudioClip node.
Field.AudioDelay	An interface defining constants for the field defIDs of the AudioDelay node.
Field.AudioFX	An interface defining constants for the field defIDs of the AudioFX node.
Field.AudioMix	An interface defining constants for the field defIDs of the AudioMix node.
Field.AudioSource	An interface defining constants for the field defIDs of the AudioSource node.
Field.AudioSwitch	An interface defining constants for the field defIDs of the AudioSwitch node.
Field.Background	An interface defining constants for the field defIDs of the Background node.
Field.Background2D	An interface defining constants for the field defIDs of the Background2D node.
Field.Billboard	An interface defining constants for the field defIDs of the Billboard node.
Field.Bitmap	An interface defining constants for the field defIDs of the Bitmap node.
Field.Box	An interface defining constants for the field defIDs of the Box node.
Field.Circle	An interface defining constants for the field defIDs of the Circle node.
Field.Collision	An interface defining constants for the field defIDs of the Collision node.
Field.Color	An interface defining constants for the field defIDs of the Color node.
Field.ColorInterpolator	An interface defining constants for the field defIDs of the ColorInterpolator node.
Field.CompositeTexture2D	An interface defining constants for the field defIDs of the CompositeTexture2D node.
Field.CompositeTexture3D	An interface defining constants for the field defIDs of the CompositeTexture3D node.
Field.Conditional	An interface defining constants for the field defIDs of the Conditional node.
Field.Cone	An interface defining constants for the field defIDs of the Cone node.
Field.Coordinate	An interface defining constants for the field defIDs of the Coordinate node.
Field.Coordinate2D	An interface defining constants for the field defIDs of the Coordinate2D node.
Field.CoordinateInterpolator	An interface defining constants for the field defIDs of the CoordinateInterpolator node.
Field.CoordinateInterpolator2D	An interface defining constants for the field defIDs of the CoordinateInterpolator2D node.
Field.Curve2D	An interface defining constants for the field defIDs of the Curve2D node.
Field.Cylinder	An interface defining constants for the field defIDs of the Cylinder node.
Field.CylinderSensor	An interface defining constants for the field defIDs of the CylinderSensor node.
Field.DirectionalLight	An interface defining constants for the field defIDs of the DirectionalLight node.
Field.DiscSensor	An interface defining constants for the field defIDs of the DiscSensor node.
Field.ElevationGrid	An interface defining constants for the field defIDs of the ElevationGrid node.
Field.Expression	An interface defining constants for the field defIDs of the Expression node.
Field.Extrusion	An interface defining constants for the field defIDs of the Extrusion node.
Field.Face	An interface defining constants for the field defIDs of the Face node.
Field.FaceDefMesh	An interface defining constants for the field defIDs of the FaceDefMesh node.
Field.FaceDefTables	An interface defining constants for the field defIDs of the FaceDefTables node.
Field.FaceDefTransform	An interface defining constants for the field defIDs of the FaceDefTransform node.
Field.FAP	An interface defining constants for the field defIDs of the FAP node.
Field.FDP	An interface defining constants for the field defIDs of the FDP node.
Field.FIT	An interface defining constants for the field defIDs of the FIT node.
Field.Fog	An interface defining constants for the field defIDs of the Fog node.
Field.FontStyle	An interface defining constants for the field defIDs of the FontStyle node.
Field.Form	An interface defining constants for the field defIDs of the Form node.
Field.Group	An interface defining constants for the field defIDs of the Group node.
Field.ImageTexture	An interface defining constants for the field defIDs of the ImageTexture node.
Field.IndexedFaceSet	An interface defining constants for the field defIDs of the IndexedFaceSet node.
Field.IndexedFaceSet2D	An interface defining constants for the field defIDs of the IndexedFaceSet2D node.
Field.IndexedLineSet	An interface defining constants for the field defIDs of the IndexedLineSet node.
Field.IndexedLineSet2	An interface defining constants for the field defIDs of the IndexedLineSet2D

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000474

D	node.
Field.Inline	An interface defining constants for the field defIDs of the Inline node.
Field.LOD	An interface defining constants for the field defIDs of the LOD node.
Field.Layer2D	An interface defining constants for the field defIDs of the Layer2D node.
Field.Layer3D	An interface defining constants for the field defIDs of the Layer3D node.
Field.Layout	An interface defining constants for the field defIDs of the Layout node.
Field.LineProperties	An interface defining constants for the field defIDs of the LineProperties node.
Field.ListeningPoint	An interface defining constants for the field defIDs of the ListeningPoint node.
Field.Material	An interface defining constants for the field defIDs of the Material node.
Field.Material2D	An interface defining constants for the field defIDs of the Material2D node.
Field.MovieTexture	An interface defining constants for the field defIDs of the MovieTexture node.
Field.NavigationInfo	An interface defining constants for the field defIDs of the NavigationInfo node.
Field.Normal	An interface defining constants for the field defIDs of the Normal node.
Field.NormalInterpolator	An interface defining constants for the field defIDs of the NormalInterpolator node.
Field.OrderedGroup	An interface defining constants for the field defIDs of the OrderedGroup node.
Field.OrientationInterpolator	An interface defining constants for the field defIDs of the OrientationInterpolator node.
Field.PixelTexture	An interface defining constants for the field defIDs of the PixelTexture node.
Field.PlaneSensor	An interface defining constants for the field defIDs of the PlaneSensor node.
Field.PlaneSensor2D	An interface defining constants for the field defIDs of the PlaneSensor2D node.
Field.PointLight	An interface defining constants for the field defIDs of the PointLight node.
Field.PointSet	An interface defining constants for the field defIDs of the PointSet node.
Field.PointSet2D	An interface defining constants for the field defIDs of the PointSet2D node.
Field.PositionInterpolator	An interface defining constants for the field defIDs of the PositionInterpolator node.
Field.PositionInterpolator2D	An interface defining constants for the field defIDs of the PositionInterpolator2D node.
Field.ProximitySensor2D	An interface defining constants for the field defIDs of the ProximitySensor2D node.
Field.ProximitySensor	An interface defining constants for the field defIDs of the ProximitySensor node.
Field.QuantizationParameter	An interface defining constants for the field defIDs of the QuantizationParameter node.
Field.Rectangle	An interface defining constants for the field defIDs of the Rectangle node.
Field.ScalarInterpolator	An interface defining constants for the field defIDs of the ScalarInterpolator node.
Field.Script	An interface defining constants for the field defIDs of the Script node.
Field.Shape	An interface defining constants for the field defIDs of the Shape node.
Field.Sound	An interface defining constants for the field defIDs of the Sound node.
Field.Sound2D	An interface defining constants for the field defIDs of the Sound2D node.
Field.Sphere	An interface defining constants for the field defIDs of the Sphere node.
Field.SphereSensor	An interface defining constants for the field defIDs of the SphereSensor node.
Field.SpotLight	An interface defining constants for the field defIDs of the SpotLight node.
Field.Switch	An interface defining constants for the field defIDs of the Switch node.
Field.TermCap	An interface defining constants for the field defIDs of the TermCap node.
Field.Text	An interface defining constants for the field defIDs of the Text node.
Field.TextureCoordinate	An interface defining constants for the field defIDs of the TextureCoordinate node.
Field.TextureTransform	An interface defining constants for the field defIDs of the TextureTransform node.
Field.TimeSensor	An interface defining constants for the field defIDs of the TimeSensor node.
Field.TouchSensor	An interface defining constants for the field defIDs of the TouchSensor node.
Field.Transform	An interface defining constants for the field defIDs of the Transform node.
Field.Transform2D	An interface defining constants for the field defIDs of the Transform2D node.
Field.Valuator	An interface defining constants for the field defIDs of the Valuator node.
Field.WorldInfo	An interface defining constants for the field defIDs of the WorldInfo node.
NewNode	An interface used to specify the creation of a new node.
SFColorFieldValue	An interface used for obtaining SFColor values.
MFVec2fFieldValue	An interface used for obtaining MFVec2f values.
EventOut	
EventOut.Anchor	
EventOut.AnimationStream	
EventOut.Appearance	
EventOut.AudioBuffer	

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000475

EventOut.AudioClip		
EventOut.AudioDelay		
EventOut.AudioFX		
EventOut.AudioMix		
EventOut.AudioSource		
EventOut.AudioSwitch		
EventOut.Background		
EventOut.Background2D		
EventOut.Billboard		
EventOut.Bitmap		
EventOut.Box		
EventOut.Circle		
EventOut.Collision		
EventOut.Color		
EventOut.ColorInterpolator		
EventOut.CompositeTexture2D		
EventOut.CompositeTexture3D		
EventOut.Conditional		
EventOut.Cone		
EventOut.Coordinate		
EventOut.Coordinate2D		
EventOut.CoordinateInterpolator		
EventOut.CoordinateInterpolator2D		
EventOut.Curve2D		
EventOut.Cylinder		
EventOut.CylinderSensor		
EventOut.DirectionalLight		
EventOut.DiscSensor		
EventOut.ElevationGrid		
EventOut.Expression		
EventOut.Extrusion		
EventOut.Face		
EventOut.FaceDefMesh		
EventOut.FaceDefTables		
EventOut.FaceDefTransform		
EventOut.FAP		
EventOut.FDP		
EventOut.FIT		
EventOut.Fog		
EventOut.FontStyle		
EventOut.Form		
EventOut.Group		
EventOut.ImageTexture		
EventOut.IndexedFaceSet		
EventOut.IndexedFaceSet2D		
EventOut.IndexedLineSet		
EventOut.IndexedLineSet2D		
EventOut.Inline		
EventOut.LOD		
EventOut.Layer2D		
EventOut.Layer3D		
EventOut.Layout		

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000476

EventOut.LineProperties	
EventOut.ListeningPoint	
EventOut.Material	
EventOut.Material2D	
EventOut.MovieTexture	
EventOut.NavigationInfo	
EventOut.Normal	
EventOut.NormalInterpolator	
EventOut.OrderedGroup	
EventOut.OrientationInterpolator	
EventOut.PixelTexture	
EventOut.PlaneSensor	
EventOut.PlaneSensor2D	
EventOut.PointLight	
EventOut.PointSet	
EventOut.PointSet2D	
EventOut.PositionInterpolator	
EventOut.PositionInterpolator2D	
EventOut.ProximitySensor2D	
EventOut.ProximitySensor	
EventOut.QuantizationParameter	
EventOut.Rectangle	
EventOut.ScalarInterpolator	
EventOut.Script	
EventOut.Shape	
EventOut.Sound	
EventOut.Sound2D	
EventOut.Sphere	
EventOut.SphereSensor	
EventOut.SpotLight	
EventOut.Switch	
EventOut.TermCap	
EventOut.Text	
EventOut.TextureCoordinate	
EventOut.TextureTransform	
EventOut.TimeSensor	
EventOut.TouchSensor	
EventOut.Transform	
EventOut.Transform2D	
EventOut.Valuator	
EventOut.Viewpoint	
EventOut.VisibilitySensor	
EventOut.Viseme	
EventOut.WorldInfo	
SFRotationFieldValue	An interface used for obtaining SFRotation values.
MFColorFieldValue	An interface used for obtaining MFColor values.
SFVec3fFieldValue	An interface used for obtaining SFVec3f values.
SFInt32FieldValue	An interface used for obtaining SFInt32 values.
SceneManager	An interface that allows access to the MPEG-4 terminal's native scene.
MFNodeFieldValue	An interface used for obtaining MFNode values.
SFBoolFieldValue	An interface used for obtaining SFBool values.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000477

EventOutListener EventIn EventIn.Anchor EventIn.Animation-Stream EventIn.Appearance EventIn.AudioBuffer EventIn.AudioClip EventIn.AudioDelay EventIn.AudioFX EventIn.AudioMix EventIn.AudioSource EventIn.AudioSwitch EventIn.Background EventIn.Background2D EventIn.Billboard EventIn.Bitmap EventIn.Box EventIn.Circle EventIn.Collision EventIn.Color EventIn.ColorInterpo-lator Even-tIn.CompositeTexture2D Even-tIn.CompositeTexture3D EventIn.Conditional EventIn.Cone EventIn.Coordinate EventIn.Coordinate2D EventIn.CoordinateIn-terpolator Even-tIn.CoordinateInterpolator2D EventIn.Curve2D EventIn.Cylinder EventIn.CylinderSen-sor EventIn.Directional-Light EventIn.DiscSensor EventIn.ElevationGrid EventIn.Expression EventIn.Extrusion EventIn.Face EventIn.FaceDefMesh EventIn.FaceDefTables EventIn.FaceDefTrans-form EventIn.FAP EventIn.FDP EventIn.FIT EventIn.Fog EventIn.FontStyle EventIn.Form EventIn.Group EventIn.ImageTexture EventIn.IndexedFac-eSet Even-	An interface to be implemented by objects that wish to receive eventOut notification from fields in the scene.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000478

tIn.IndexedFaceSet2D EventIn.Indexed- LineSet Even- tIn.IndexedLineSet2D EventIn.Inline EventIn.LOD EventIn.Layer2D EventIn.Layer3D EventIn.Layout EventIn.LineProper- ties EventIn.Listening- Point EventIn.Material EventIn.Material2D EventIn.MovieTexture EventIn.Navigation- Info EventIn.Normal EventIn.NormalInter- polator EventIn.OrderedGroup EventIn.Orientation- Interpolator EventIn.PixelTexture EventIn.PlaneSensor EventIn.PlaneSensor2D EventIn.PointLight EventIn.PointSet EventIn.PointSet2D EventIn.PositionIn- terpolator Even- tIn.PositionInterpola tor2D Even- tIn.ProximitySensor2D EventIn.ProximitySen- sor EventIn.Quantization- Parameter EventIn.Rectangle EventIn.ScalarInter- polator EventIn.Script EventIn.Shape EventIn.Sound EventIn.Sound2D EventIn.Sphere EventIn.SphereSensor EventIn.SpotLight EventIn.Switch EventIn.TermCap EventIn.Text EventIn.TextureCoor- dinate EventIn.TextureTrans- form EventIn.TimeSensor EventIn.TouchSensor EventIn.Transform EventIn.Transform2D EventIn.Valuator EventIn.Viewpoint EventIn.Visibility-		

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000479

Sensor EventIn.Viseme EventIn.WorldInfo	
MFRotationFieldValue	An interface used for obtaining MFRotation values.
SFTimeFieldValue	An interface used for obtaining SFTime values.
SFFloatFieldValue	An interface used for obtaining SFFloat values.
MFVec3fFieldValue	An interface used for obtaining MFVec3f values.
MFInt32FieldValue	An interface used for obtaining MFInt32 values.
Exceptions	
InvalidSceneException	An exception used by this package to indicate that an operation was attempted on a Scene that is no longer valid.
BadNodeException	The class BadNodeException is a subclass of MPEGJException.
BadParameterException	An exception used by this package to indicate that a parameter to a member function is not valid.
InvalidNodeException	An exception used to indicate that an operation was attempted on a node that is no longer valid.

V.5.2 org.iso.mpeg.mpegj.scene.BadNodeException

V.5.2.1 Syntax

```
public class BadNodeException extends org.iso.mpeg.mpegj.MPEGJException
```

java.lang.Object
 |
 +--java.lang.Throwable
  |
  +--java.lang.Exception
  |
  +--org.iso.mpeg.mpegj.MPEGJException
  |
  +--**org.iso.mpeg.mpegj.scene.BadNodeException**

All Implemented Interfaces:
java.io.Serializable

V.5.2.2 Description

The class BadNodeException is a subclass of MPEGJException. This Exception is thrown by the method getRenderer() of Resource Manager.

Member Summary	
Constructors	BadNodeException() BadNodeException(String)

V.5.2.3 Constructors

BadNodeException()
```
public  BadNodeException()
```
Constructs an BadNodeException with no specified detail message.

BadNodeException(String)
```
public  BadNodeException(java.lang.String)
```
Constructs an RendererFoundException with a detailed message.

V.5.3 org.iso.mpeg.mpegj.scene.BadParameterException

V.5.3.1 Syntax

```
public class BadParameterException extends org.iso.mpeg.mpegj.MPEGJException
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000480

java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--**org.iso.mpeg.mpegj.scene.BadParameterException**

All Implemented Interfaces:
java.io.Serializable

V.5.3.2 Description

An exception used by this package to indicate that a parameter to a member function is not valid.

Member Summary	
Constructors	BadParameterException() BadParameterException(String)

V.5.3.3 Constructors

BadParameterException()
`public BadParameterException()`

Default constructor.

BadParameterException(String)
`public BadParameterException(java.lang.String)`

String constructor.

V.5.4 org.iso.mpeg.mpegj.scene.EventIn

V.5.4.1 Syntax

```
public interface EventIn
```

V.5.4.2 Description

Member Summary	
Inner Classes	
static interface	EventIn.Anchor
static interface	EventIn.AnimationStream
static interface	EventIn.Appearance
static interface	EventIn.AudioBuffer
static interface	EventIn.AudioClip
static interface	EventIn.AudioDelay
static interface	EventIn.AudioFX
static interface	EventIn.AudioMix
static interface	EventIn.AudioSource
static interface	EventIn.AudioSwitch
static interface	EventIn.Background
static interface	EventIn.Background2D
static interface	EventIn.Billboard
static interface	EventIn.Bitmap
static interface	EventIn.Box
static interface	EventIn.Circle
static interface	EventIn.Collision
static interface	EventIn.Color
static interface	EventIn.ColorInterpolator

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000481

static interface	EventIn.CompositeTexture2D
static interface	EventIn.CompositeTexture3D
static interface	EventIn.Conditional
static interface	EventIn.Cone
static interface	EventIn.Coordinate
static interface	EventIn.Coordinate2D
static interface	EventIn.CoordinateInterpolator
static interface	EventIn.CoordinateInterpolator2D
static interface	EventIn.Curve2D
static interface	EventIn.Cylinder
static interface	EventIn.CylinderSensor
static interface	EventIn.DirectionalLight
static interface	EventIn.DiscSensor
static interface	EventIn.ElevationGrid
static interface	EventIn.Expression
static interface	EventIn.Extrusion
static interface	EventIn.Face
static interface	EventIn.FaceDefMesh
static interface	EventIn.FaceDefTables
static interface	EventIn.FaceDefTransform
static interface	EventIn.FAP
static interface	EventIn.FDP
static interface	EventIn.FIT
static interface	EventIn.Fog
static interface	EventIn.FontStyle
static interface	EventIn.Form
static interface	EventIn.Group
static interface	EventIn.ImageTexture
static interface	EventIn.IndexedFaceSet
static interface	EventIn.IndexedFaceSet2D
static interface	EventIn.IndexedLineSet
static interface	EventIn.IndexedLineSet2D
static interface	EventIn.Inline
static interface	EventIn.LOD
static interface	EventIn.Layer2D
static interface	EventIn.Layer3D
static interface	EventIn.Layout
static interface	EventIn.LineProperties
static interface	EventIn.ListeningPoint
static interface	EventIn.Material
static interface	EventIn.Material2D
static interface	EventIn.MovieTexture
static interface	EventIn.NavigationInfo
static interface	EventIn.Normal
static interface	EventIn.NormalInterpolator
static interface	EventIn.OrderedGroup
static interface	EventIn.OrientationInterpolator
static interface	EventIn.PixelTexture
static interface	EventIn.PlaneSensor
static interface	EventIn.PlaneSensor2D
static interface	EventIn.PointLight
static interface	EventIn.PointSet
static interface	EventIn.PointSet2D
static interface	EventIn.PositionInterpolator
static interface	EventIn.PositionInterpolator2D
static interface	EventIn.ProximitySensor2D
static interface	EventIn.ProximitySensor
static interface	EventIn.QuantizationParameter
static interface	EventIn.Rectangle
static interface	EventIn.ScalarInterpolator
static interface	EventIn.Script
static interface	EventIn.Shape
static interface	EventIn.Sound
static interface	EventIn.Sound2D
static interface	EventIn.Sphere
static interface	EventIn.SphereSensor
static interface	EventIn.SpotLight
static interface	EventIn.Switch
static interface	EventIn.TermCap
static interface	EventIn.Text
static interface	EventIn.TextureCoordinate

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000482

static interface	EventIn.TextureTransform
static interface	EventIn.TimeSensor
static interface	EventIn.TouchSensor
static interface	EventIn.Transform
static interface	EventIn.Transform2D
static interface	EventIn.Valuator
static interface	EventIn.Viewpoint
static interface	EventIn.VisibilitySensor
static interface	EventIn.Viseme
static interface	EventIn.WorldInfo

V.5.5 org.iso.mpeg.mpegj.scene.EventIn.Anchor

V.5.5.1 Syntax

```
public static interface EventIn.Anchor
```

V.5.5.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	children
int	description
int	parameter
int	url

V.5.5.3 Fields

addChildren
```
public static final int addChildren
```
addChildren = 0

children
```
public static final int children
```
children = 2

description
```
public static final int description
```
description = 3

parameter
```
public static final int parameter
```
parameter = 4

removeChildren
```
public static final int removeChildren
```
removeChildren = 1

url
```
public static final int url
```
url = 5

V.5.6 org.iso.mpeg.mpegj.scene.EventIn.AnimationStream

V.5.6.1 Syntax

```
public static interface EventIn.AnimationStream
```
V.5.6.2 Description

Member Summary	
Fields	
int	loop
int	speed

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000483

int	startTime
int	stopTime
int	url

V.5.6.3 Fields

loop
`public static final int loop`

loop = 0

speed
`public static final int speed`

speed = 1

startTime
`public static final int startTime`

startTime = 2

stopTime
`public static final int stopTime`

stopTime = 3

url
`public static final int url`

url = 4

V.5.7 org.iso.mpeg.mpegj.scene.EventIn.Appearance

V.5.7.1 Syntax

```
public static interface EventIn.Appearance
```

V.5.7.2 Description

Member Summary	
Fields	
int	material
int	texture
int	textureTransform

V.5.7.3 Fields

material
`public static final int material`

material = 0

texture
`public static final int texture`

texture = 1

textureTransform
`public static final int textureTransform`

textureTransform = 2

V.5.8 org.iso.mpeg.mpegj.scene.EventIn.AudioBuffer

V.5.8.1 Syntax

```
public static interface EventIn.AudioBuffer
```

V.5.8.2 Description

Member Summary	
Fields	

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000484

int	loop
int	pitch
int	startTime
int	stopTime
int	children
int	numChan
int	phaseGroup

V.5.8.3 Fields

children
`public static final int children`

children = 4

loop
`public static final int loop`

loop = 0

numChan
`public static final int numChan`

numChan = 5

phaseGroup
`public static final int phaseGroup`

phaseGroup = 6

pitch
`public static final int pitch`

pitch = 1

startTime
`public static final int startTime`

startTime = 2

stopTime
`public static final int stopTime`

stopTime = 3

V.5.9 org.iso.mpeg.mpegj.scene.EventIn.AudioClip

V.5.9.1 Syntax

```
public static interface EventIn.AudioClip
```

V.5.9.2 Description

Member Summary	
Fields	
int	description
int	loop
int	pitch
int	startTime
int	stopTime
int	url

V.5.9.3 Fields

description
`public static final int description`

description = 0

loop
`public static final int loop`

loop = 1

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000485

pitch
`public static final int pitch`

pitch = 2

startTime
`public static final int startTime`

startTime = 3

stopTime
`public static final int stopTime`

stopTime = 4

url
`public static final int url`

url = 5

V.5.10 org.iso.mpeg.mpegj.scene.EventIn.AudioDelay

V.5.10.1 Syntax

```
public static interface EventIn.AudioDelay
```

V.5.10.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	children
int	delay

V.5.10.3 Fields

addChildren
`public static final int addChildren`

addChildren = 0

children
`public static final int children`

children = 2

delay
`public static final int delay`

delay = 3

removeChildren
`public static final int removeChildren`

removeChildren = 1

V.5.11 org.iso.mpeg.mpegj.scene.EventIn.AudioFX

V.5.11.1 Syntax

```
public static interface EventIn.AudioFX
```

V.5.11.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	children
int	orch
int	score
int	params

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000486

V.5.11.3 Fields

addChildren
`public static final int addChildren`

addChildren = 0

children
`public static final int children`

children = 2

orch
`public static final int orch`

orch = 3

params
`public static final int params`

params = 5

removeChildren
`public static final int removeChildren`

removeChildren = 1

score
`public static final int score`

score = 4

V.5.12 org.iso.mpeg.mpegj.scene.EventIn.AudioMix

V.5.12.1 Syntax

```
public static interface EventIn.AudioMix
```

V.5.12.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	children
int	numInputs
int	matrix

V.5.12.3 Fields

addChildren
`public static final int addChildren`

addChildren = 0

children
`public static final int children`

children = 2

matrix
`public static final int matrix`

matrix = 4

numInputs
`public static final int numInputs`

numInputs = 3

removeChildren
`public static final int removeChildren`

removeChildren = 1

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000487

V.5.13 org.iso.mpeg.mpegj.scene.EventIn.AudioSource

V.5.13.1 Syntax

```
public static interface EventIn.AudioSource
```

V.5.13.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	children
int	url
int	pitch
int	speed
int	startTime
int	stopTime

V.5.13.3 Fields

addChildren
```
public static final int addChildren
```
addChildren = 0

children
```
public static final int children
```
children = 2

pitch
```
public static final int pitch
```
pitch = 4

removeChildren
```
public static final int removeChildren
```
removeChildren = 1

speed
```
public static final int speed
```
speed = 5

startTime
```
public static final int startTime
```
startTime = 6

stopTime
```
public static final int stopTime
```
stopTime = 7

url
```
public static final int url
```
url = 3

V.5.14 org.iso.mpeg.mpegj.scene.EventIn.AudioSwitch

V.5.14.1 Syntax

```
public static interface EventIn.AudioSwitch
```

V.5.14.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	children

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000488

int	whichChoice

V.5.14.3 Fields

addChildren
`public static final int addChildren`

addChildren = 0

children
`public static final int children`

children = 2

removeChildren
`public static final int removeChildren`

removeChildren = 1

whichChoice
`public static final int whichChoice`

whichChoice = 3

V.5.15 org.iso.mpeg.mpegj.scene.EventIn.Background

V.5.15.1 Syntax

```
public static interface EventIn.Background
```

V.5.15.2 Description

Member Summary	
Fields	
int	set_bind
int	groundAngle
int	groundColor
int	backUrl
int	bottomUrl
int	frontUrl
int	leftUrl
int	rightUrl
int	topUrl
int	skyAngle
int	skyColor

V.5.15.3 Fields

backUrl
`public static final int backUrl`

backUrl = 3

bottomUrl
`public static final int bottomUrl`

bottomUrl = 4

frontUrl
`public static final int frontUrl`

frontUrl = 5

groundAngle
`public static final int groundAngle`

groundAngle = 1

groundColor
`public static final int groundColor`

groundColor = 2

leftUrl

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000489

```
public static final int leftUrl
```

leftUrl = 6

rightUrl
```
public static final int rightUrl
```

rightUrl = 7

set_bind
```
public static final int set_bind
```

set_bind = 0

skyAngle
```
public static final int skyAngle
```

skyAngle = 9

skyColor
```
public static final int skyColor
```

skyColor = 10

topUrl
```
public static final int topUrl
```

topUrl = 8

V.5.16 org.iso.mpeg.mpegj.scene.EventIn.Background2D

V.5.16.1 Syntax

```
public static interface EventIn.Background2D
```

V.5.16.2 Description

Member Summary	
Fields	
int	set_bind
int	backColor
int	url

V.5.16.3 Fields

backColor
```
public static final int backColor
```

backColor = 1

set_bind
```
public static final int set_bind
```

set_bind = 0

url
```
public static final int url
```

url = 2

V.5.17 org.iso.mpeg.mpegj.scene.EventIn.Billboard

V.5.17.1 Syntax

```
public static interface EventIn.Billboard
```

V.5.17.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	children

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000490

int	axisOfRotation

V.5.17.3 Fields

addChildren
`public static final int addChildren`

addChildren = 0

axisOfRotation
`public static final int axisOfRotation`

axisOfRotation = 3

children
`public static final int children`

children = 2

removeChildren
`public static final int removeChildren`

removeChildren = 1

V.5.18 org.iso.mpeg.mpegj.scene.EventIn.Bitmap

V.5.18.1 Syntax

```
public static interface EventIn.Bitmap
```

V.5.18.2 Description

Member Summary	
Fields	
int	scale

V.5.18.3 Fields

scale
`public static final int scale`

scale = 0

V.5.19 org.iso.mpeg.mpegj.scene.EventIn.Box

V.5.19.1 Syntax

```
public static interface EventIn.Box
```

V.5.19.2 Description

V.5.20 org.iso.mpeg.mpegj.scene.EventIn.Circle

V.5.20.1 Syntax

```
public static interface EventIn.Circle
```

V.5.20.2 Description

Member Summary	
Fields	
int	radius

V.5.20.3 Fields

radius
`public static final int radius`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000491

radius = 0

V.5.21 org.iso.mpeg.mpegj.scene.EventIn.Collision

V.5.21.1 Syntax

```
public static interface EventIn.Collision
```

V.5.21.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	children
int	collide

V.5.21.3 Fields

addChildren
```
public static final int addChildren
```
addChildren = 0

children
```
public static final int children
```
children = 2

collide
```
public static final int collide
```
collide = 3

removeChildren
```
public static final int removeChildren
```
removeChildren = 1

V.5.22 org.iso.mpeg.mpegj.scene.EventIn.Color

V.5.22.1 Syntax

```
public static interface EventIn.Color
```

V.5.22.2 Description

Member Summary	
Fields	
int	color

V.5.22.3 Fields

color
```
public static final int color
```
color = 0

V.5.23 org.iso.mpeg.mpegj.scene.EventIn.ColorInterpolator

V.5.23.1 Syntax

```
public static interface EventIn.ColorInterpolator
```

V.5.23.2 Description

Member Summary	
Fields	
int	set_fraction

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000492

int	key
int	keyValue

V.5.23.3 Fields

key
`public static final int key`

key = 1

keyValue
`public static final int keyValue`

keyValue = 2

set_fraction
`public static final int set_fraction`

set_fraction = 0

V.5.24 org.iso.mpeg.mpegj.scene.EventIn.CompositeTexture2D

V.5.24.1 Syntax

```
public static interface EventIn.CompositeTexture2D
```

V.5.24.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	children
int	pixelWidth
int	pixelHeight
int	background
int	viewport

V.5.24.3 Fields

addChildren
`public static final int addChildren`

addChildren = 0

background
`public static final int background`

background = 5

children
`public static final int children`

children = 2

pixelHeight
`public static final int pixelHeight`

pixelHeight = 4

pixelWidth
`public static final int pixelWidth`

pixelWidth = 3

removeChildren
`public static final int removeChildren`

removeChildren = 1

viewport
`public static final int viewport`

viewport = 6

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000493

V.5.25 org.iso.mpeg.mpegj.scene.EventIn.CompositeTexture3D

V.5.25.1 Syntax

```
public static interface EventIn.CompositeTexture3D
```

V.5.25.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	children
int	pixelWidth
int	pixelHeight
int	background
int	fog
int	navigationInfo
int	viewpoint

V.5.25.3 Fields

addChildren
```
public static final int addChildren
```
addChildren = 0

background
```
public static final int background
```
background = 5

children
```
public static final int children
```
children = 2

fog
```
public static final int fog
```
fog = 6

navigationInfo
```
public static final int navigationInfo
```
navigationInfo = 7

pixelHeight
```
public static final int pixelHeight
```
pixelHeight = 4

pixelWidth
```
public static final int pixelWidth
```
pixelWidth = 3

removeChildren
```
public static final int removeChildren
```
removeChildren = 1

viewpoint
```
public static final int viewpoint
```
viewpoint = 8

V.5.26 org.iso.mpeg.mpegj.scene.EventIn.Conditional

V.5.26.1 Syntax

```
public static interface EventIn.Conditional
```

V.5.26.2 Description

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000494

Member Summary	
Fields	
int	activate
int	reverseActivate
int	buffer

V.5.26.3 Fields

activate
`public static final int activate`

activate = 0

buffer
`public static final int buffer`

buffer = 2

reverseActivate
`public static final int reverseActivate`

reverseActivate = 1

V.5.27 org.iso.mpeg.mpegj.scene.EventIn.Cone

V.5.27.1 Syntax

```
public static interface EventIn.Cone
```

V.5.27.2 Description

V.5.28 org.iso.mpeg.mpegj.scene.EventIn.Coordinate

V.5.28.1 Syntax

```
public static interface EventIn.Coordinate
```

V.5.28.2 Description

Member Summary	
Fields	
int	point

V.5.28.3 Fields

point
`public static final int point`

point = 0

V.5.29 org.iso.mpeg.mpegj.scene.EventIn.Coordinate2D

V.5.29.1 Syntax

```
public static interface EventIn.Coordinate2D
```

V.5.29.2 Description

Member Summary	
Fields	
int	point

V.5.29.3 Fields

point
`public static final int point`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000495

point = 0

V.5.30 org.iso.mpeg.mpegj.scene.EventIn.CoordinateInterpolator

V.5.30.1 Syntax

```
public static interface EventIn.CoordinateInterpolator
```

V.5.30.2 Description

Member Summary	
Fields	
int	set_fraction
int	key
int	keyValue

V.5.30.3 Fields

key
```
public static final int key
```
key = 1

keyValue
```
public static final int keyValue
```
keyValue = 2

set_fraction
```
public static final int set_fraction
```
set_fraction = 0

V.5.31 org.iso.mpeg.mpegj.scene.EventIn.CoordinateInterpolator2D

V.5.31.1 Syntax

```
public static interface EventIn.CoordinateInterpolator2D
```

V.5.31.2 Description

Member Summary	
Fields	
int	set_fraction
int	key
int	keyValue

V.5.31.3 Fields

key
```
public static final int key
```
key = 1

keyValue
```
public static final int keyValue
```
keyValue = 2

set_fraction
```
public static final int set_fraction
```
set_fraction = 0

V.5.32 org.iso.mpeg.mpegj.scene.EventIn.Curve2D

V.5.32.1 Syntax

```
public static interface EventIn.Curve2D
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000496

V.5.32.2 Description

Member Summary	
Fields	
int	point
int	fineness
int	type

V.5.32.3 Fields

fineness
`public static final int fineness`

fineness = 1

point
`public static final int point`

point = 0

type
`public static final int type`

type = 2

V.5.33 org.iso.mpeg.mpegj.scene.EventIn.Cylinder

V.5.33.1 Syntax

```
public static interface EventIn.Cylinder
```

V.5.33.2 Description

V.5.34 org.iso.mpeg.mpegj.scene.EventIn.CylinderSensor

V.5.34.1 Syntax

```
public static interface EventIn.CylinderSensor
```

V.5.34.2 Description

Member Summary	
Fields	
int	autoOffset
int	diskAngle
int	enabled
int	maxAngle
int	minAngle
int	offset

V.5.34.3 Fields

autoOffset
`public static final int autoOffset`

autoOffset = 0

diskAngle
`public static final int diskAngle`

diskAngle = 1

enabled
`public static final int enabled`

enabled = 2

maxAngle
`public static final int maxAngle`

maxAngle = 3

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000497

minAngle
`public static final int minAngle`

minAngle = 4

offset
`public static final int offset`

offset = 5

V.5.35 org.iso.mpeg.mpegj.scene.EventIn.DirectionalLight

V.5.35.1 Syntax

```
public static interface EventIn.DirectionalLight
```

V.5.35.2 Description

Member Summary	
Fields	
int	ambientIntensity
int	color
int	direction
int	intensity
int	on

V.5.35.3 Fields

ambientIntensity
`public static final int ambientIntensity`

ambientIntensity = 0

color
`public static final int color`

color = 1

direction
`public static final int direction`

direction = 2

intensity
`public static final int intensity`

intensity = 3

on
`public static final int on`

on = 4

V.5.36 org.iso.mpeg.mpegj.scene.EventIn.DiscSensor

V.5.36.1 Syntax

```
public static interface EventIn.DiscSensor
```

V.5.36.2 Description

Member Summary	
Fields	
int	autoOffset
int	enabled
int	maxAngle
int	minAngle
int	offset

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000498

V.5.36.3 Fields

autoOffset
`public static final int autoOffset`

enabled
`public static final int enabled`

enabled = 1

maxAngle
`public static final int maxAngle`

maxAngle = 2

minAngle
`public static final int minAngle`

minAngle = 3

offset
`public static final int offset`

offset = 4

V.5.37 org.iso.mpeg.mpegj.scene.EventIn.ElevationGrid

V.5.37.1 Syntax

```
public static interface EventIn.ElevationGrid
```

V.5.37.2 Description

Member Summary	
Fields	
int	set_height
int	color
int	normal
int	texCoord

V.5.37.3 Fields

color
`public static final int color`

color = 1

normal
`public static final int normal`

normal = 2

set_height
`public static final int set_height`

set_height = 0

texCoord
`public static final int texCoord`

texCoord = 3

V.5.38 org.iso.mpeg.mpegj.scene.EventIn.Expression

V.5.38.1 Syntax

```
public static interface EventIn.Expression
```

V.5.38.2 Description

Member Summary	
Fields	
int	expression_select1

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000499

int	expression_intensity1
int	expression_select2
int	expression_intensity2
int	init_face
int	expression_def

V.5.38.3 Fields

expression_def
`public static final int expression_def`

expression_def = 5

expression_intensity1
`public static final int expression_intensity1`

expression_intensity1 = 1

expression_intensity2
`public static final int expression_intensity2`

expression_intensity2 = 3

expression_select1
`public static final int expression_select1`

expression_select1 = 0

expression_select2
`public static final int expression_select2`

expression_select2 = 2

init_face
`public static final int init_face`

init_face = 4

V.5.39 org.iso.mpeg.mpegj.scene.EventIn.Extrusion

V.5.39.1 Syntax

```
public static interface EventIn.Extrusion
```

V.5.39.2 Description

Member Summary	
Fields	
int	set_crossSection
int	set_orientation
int	set_scale
int	set_spine

V.5.39.3 Fields

set_crossSection
`public static final int set_crossSection`

set_crossSection = 0

set_orientation
`public static final int set_orientation`

set_orientation = 1

set_scale
`public static final int set_scale`

set_scale = 2

set_spine
`public static final int set_spine`

set_spine = 3

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000500

V.5.40 org.iso.mpeg.mpegj.scene.EventIn.Face

V.5.40.1 Syntax

```
public static interface EventIn.Face
```

V.5.40.2 Description

Member Summary	
Fields	
int	fap
int	fdp
int	fit
int	ttsSource
int	renderedFace

V.5.40.3 Fields

fap
```
public static final int fap
```
fap = 0

fdp
```
public static final int fdp
```
fdp = 1

fit
```
public static final int fit
```
fit = 2

renderedFace
```
public static final int renderedFace
```
renderedFace = 4

ttsSource
```
public static final int ttsSource
```
ttsSource = 3

V.5.41 org.iso.mpeg.mpegj.scene.EventIn.FaceDefMesh

V.5.41.1 Syntax

```
public static interface EventIn.FaceDefMesh
```

V.5.41.2 Description

V.5.42 org.iso.mpeg.mpegj.scene.EventIn.FaceDefTables

V.5.42.1 Syntax

```
public static interface EventIn.FaceDefTables
```

V.5.42.2 Description

Member Summary	
Fields	
int	faceDefMesh
int	faceDefTransform

V.5.42.3 Fields

faceDefMesh
```
public static final int faceDefMesh
```
faceDefMesh = 0

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000501

faceDefTransform
`public static final int faceDefTransform`

faceDefTransform = 1

V.5.43 org.iso.mpeg.mpegj.scene.EventIn.FaceDefTransform

V.5.43.1 Syntax

`public static interface EventIn.FaceDefTransform`

V.5.43.2 Description

V.5.44 org.iso.mpeg.mpegj.scene.EventIn.FAP

V.5.44.1 Syntax

`public static interface EventIn.FAP`

V.5.44.2 Description

Member Summary	
Fields	
int	viseme
int	expression
int	open_jaw
int	lower_t_midlip
int	raise_b_midlip
int	stretch_l_corner
int	stretch_r_corner
int	lower_t_lip_lm
int	lower_t_lip_rm
int	lower_b_lip_lm
int	lower_b_lip_rm
int	raise_l_cornerlip
int	raise_r_cornerlip
int	thrust_jaw
int	shift_jaw
int	push_b_lip
int	push_t_lip
int	depress_chin
int	close_t_l_eyelid
int	close_t_r_eyelid
int	close_b_l_eyelid
int	close_b_r_eyelid
int	yaw_l_eyeball
int	yaw_r_eyeball
int	pitch_l_eyeball
int	pitch_r_eyeball
int	thrust_l_eyeball
int	thrust_r_eyeball
int	dilate_l_pupil
int	dilate_r_pupil
int	raise_l_i_eyebrow
int	raise_r_i_eyebrow
int	raise_l_m_eyebrow
int	raise_r_m_eyebrow
int	raise_l_o_eyebrow
int	raise_r_o_eyebrow
int	squeeze_l_eyebrow
int	squeeze_r_eyebrow
int	puff_l_cheek
int	puff_r_cheek
int	lift_l_cheek
int	lift_r_cheek
int	shift_tongue_tip
int	raise_tongue_tip
int	thrust_tongue_tip
int	raise_tongue

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000502

int	tongue_roll
int	head_pitch
int	head_yaw
int	head_roll
int	lower_t_midlip_o
int	raise_b_midlip_o
int	stretch_l_cornerlip
int	stretch_r_cornerlip
int	lower_t_lip_lm_o
int	lower_t_lip_rm_o
int	raise_b_lip_lm_o
int	raise_b_lip_rm_o
int	raise_l_cornerlip_o
int	raise_r_cornerlip_o
int	stretch_l_nose
int	stretch_r_nose
int	raise_nose
int	bend_nose
int	raise_l_ear
int	raise_r_ear
int	pull_l_ear
int	pull_r_ear

V.5.44.3 Fields

bend_nose
`public static final int bend_nose`

bend_nose = 63

close_b_l_eyelid
`public static final int close_b_l_eyelid`

close_b_l_eyelid = 20

close_b_r_eyelid
`public static final int close_b_r_eyelid`

close_b_r_eyelid = 21

close_t_l_eyelid
`public static final int close_t_l_eyelid`

close_t_l_eyelid = 18

close_t_r_eyelid
`public static final int close_t_r_eyelid`

close_t_r_eyelid = 19

depress_chin
`public static final int depress_chin`

depress_chin = 17

dilate_l_pupil
`public static final int dilate_l_pupil`

dilate_l_pupil = 28

dilate_r_pupil
`public static final int dilate_r_pupil`

dilate_r_pupil = 29

expression
`public static final int expression`

expression = 1

head_pitch
`public static final int head_pitch`

head_pitch = 47

head_roll
`public static final int head_roll`

head_roll = 49

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000503

head_yaw
`public static final int head_yaw`

head_yaw = 48

lift_l_cheek
`public static final int lift_l_cheek`

lift_l_cheek = 40

lift_r_cheek
`public static final int lift_r_cheek`

lift_r_cheek = 41

lower_b_lip_lm
`public static final int lower_b_lip_lm`

lower_b_lip_lm = 9

lower_b_lip_rm
`public static final int lower_b_lip_rm`

lower_b_lip_rm = 10

lower_t_lip_lm
`public static final int lower_t_lip_lm`

lower_t_lm = 7

lower_t_lip_lm_o
`public static final int lower_t_lip_lm_o`

lower_t_lip_lm_o = 54

lower_t_lip_rm
`public static final int lower_t_lip_rm`

lower_t_lip_rm = 8

lower_t_lip_rm_o
`public static final int lower_t_lip_rm_o`

lower_t_lip_rm_o = 55

lower_t_midlip
`public static final int lower_t_midlip`

lower_t_midlip = 3

lower_t_midlip_o
`public static final int lower_t_midlip_o`

lower_t_midlip_o = 50

open_jaw
`public static final int open_jaw`

open_jaw = 2

pitch_l_eyeball
`public static final int pitch_l_eyeball`

pitch_l_eyeball = 24

pitch_r_eyeball
`public static final int pitch_r_eyeball`

pitch_r_eyeball = 25

puff_l_cheek
`public static final int puff_l_cheek`

puff_l_cheek = 38

puff_r_cheek
`public static final int puff_r_cheek`

puff_r_cheek = 39

pull_l_ear
`public static final int pull_l_ear`

pull_l_ear = 66

pull_r_ear

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000504

```
public static final int pull_r_ear
```

pull_r_ear = 67

push_b_lip

```
public static final int push_b_lip
```

push_b_lip = 15

push_t_lip

```
public static final int push_t_lip
```

push_t_lip = 16

raise_b_lip_lm_o

```
public static final int raise_b_lip_lm_o
```

raise_b_lip_lm_o = 56

raise_b_lip_rm_o

```
public static final int raise_b_lip_rm_o
```

raise_b_lip_rm_o = 57

raise_b_midlip

```
public static final int raise_b_midlip
```

raise_b_midlip = 4

raise_b_midlip_o

```
public static final int raise_b_midlip_o
```

raise_b_midlip_o = 51

raise_l_cornerlip

```
public static final int raise_l_cornerlip
```

raise_l_cornerlip = 11

raise_l_cornerlip_o

```
public static final int raise_l_cornerlip_o
```

raise_l_cornerlip_o = 58

raise_l_ear

```
public static final int raise_l_ear
```

raise_l_ear = 64

raise_l_i_eyebrow

```
public static final int raise_l_i_eyebrow
```

raise_l_i_eyebrow = 30

raise_l_m_eyebrow

```
public static final int raise_l_m_eyebrow
```

raise_l_m_eyebrow = 32

raise_l_o_eyebrow

```
public static final int raise_l_o_eyebrow
```

raise_l_o_eyebrow = 34

raise_nose

```
public static final int raise_nose
```

raise_nose = 62

raise_r_cornerlip

```
public static final int raise_r_cornerlip
```

raise_r_cornerlip = 12

raise_r_cornerlip_o

```
public static final int raise_r_cornerlip_o
```

raise_r_cornerlip_o = 59

raise_r_ear

```
public static final int raise_r_ear
```

raise_r_ear = 65

raise_r_i_eyebrow

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000505

```
public static final int raise_r_i_eyebrow
```

raise_r_i_eyebrow = 31

raise_r_m_eyebrow
```
public static final int raise_r_m_eyebrow
```

raise_r_m_eyebrow = 33

raise_r_o_eyebrow
```
public static final int raise_r_o_eyebrow
```

raise_r_o_eyebrow = 35

raise_tongue
```
public static final int raise_tongue
```

raise_tongue = 45

raise_tongue_tip
```
public static final int raise_tongue_tip
```

raise_tongue_tip = 43

shift_jaw
```
public static final int shift_jaw
```

shift_jaw = 14

shift_tongue_tip
```
public static final int shift_tongue_tip
```

shift_tongue_tip = 42

squeeze_l_eyebrow
```
public static final int squeeze_l_eyebrow
```

squeeze_l_eyebrow = 36

squeeze_r_eyebrow
```
public static final int squeeze_r_eyebrow
```

squeeze_r_eyebrow = 37

stretch_l_corner
```
public static final int stretch_l_corner
```

stretch_l_corner = 5

stretch_l_cornerlip
```
public static final int stretch_l_cornerlip
```

stretch_l_cornerlip = 52

stretch_l_nose
```
public static final int stretch_l_nose
```

stretch_l_nose = 60

stretch_r_corner
```
public static final int stretch_r_corner
```

stretch_r_corner = 6

stretch_r_cornerlip
```
public static final int stretch_r_cornerlip
```

stretch_r_cornerlip = 53

stretch_r_nose
```
public static final int stretch_r_nose
```

stretch_r_nose = 61

thrust_jaw
```
public static final int thrust_jaw
```

thrust_jaw = 13

thrust_l_eyeball
```
public static final int thrust_l_eyeball
```

thrust_l_eyeball = 26

thrust_r_eyeball

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000506

```
public static final int thrust_r_eyeball
```

thrust_r_eyeball = 27

thrust_tongue_tip
```
public static final int thrust_tongue_tip
```

thrust_tongue_tip = 44

tongue_roll
```
public static final int tongue_roll
```

tongue_roll = 46

viseme
```
public static final int viseme
```

viseme = 0

yaw_l_eyeball
```
public static final int yaw_l_eyeball
```

yaw_l_eyeball = 22

yaw_r_eyeball
```
public static final int yaw_r_eyeball
```

yaw_r_eyeball = 23

V.5.45 org.iso.mpeg.mpegj.scene.EventIn.FDP

V.5.45.1 Syntax

```
public static interface EventIn.FDP
```

V.5.45.2 Description

Member Summary	
Fields	
int	featurePointsCoord
int	textureCoord
int	faceDefTables
int	faceSceneGraph

V.5.45.3 Fields

faceDefTables
```
public static final int faceDefTables
```

faceDefTables = 2

faceSceneGraph
```
public static final int faceSceneGraph
```

faceSceneGraph = 3

featurePointsCoord
```
public static final int featurePointsCoord
```

featurePointsCoord = 0

textureCoord
```
public static final int textureCoord
```

textureCoord = 1

V.5.46 org.iso.mpeg.mpegj.scene.EventIn.FIT

V.5.46.1 Syntax

```
public static interface EventIn.FIT
```

V.5.46.2 Description

Member Summary	

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000507

Fields	
int	FAPs
int	Graph
int	numeratorExp
int	denominatorExp
int	numeratorImpulse
int	numeratorTerms
int	denominatorTerms
int	numeratorCoefs
int	denominatorCoefs

V.5.46.3 Fields

denominatorCoefs
`public static final int denominatorCoefs`

denominatorCoefs = 8

denominatorExp
`public static final int denominatorExp`

denominatorExp = 3

denominatorTerms
`public static final int denominatorTerms`

denominatorTerms = 6

FAPs
`public static final int FAPs`

FAPs = 0

Graph
`public static final int Graph`

Graph = 1

numeratorCoefs
`public static final int numeratorCoefs`

numeratorCoefs = 7

numeratorExp
`public static final int numeratorExp`

numeratorExp = 2

numeratorImpulse
`public static final int numeratorImpulse`

numeratorImpulse = 4

numeratorTerms
`public static final int numeratorTerms`

numeratorTerms = 5

V.5.47 org.iso.mpeg.mpegj.scene.EventIn.Fog

V.5.47.1 Syntax

```
public static interface EventIn.Fog
```

V.5.47.2 Description

Member Summary	
Fields	
int	color
int	fogType
int	visibilityRange
int	set_bind

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000508

V.5.47.3 Fields

color
`public static final int color`

color = 0

fogType
`public static final int fogType`

fogType = 1

set_bind
`public static final int set_bind`

set_bind = 3

visibilityRange
`public static final int visibilityRange`

visibilityRange = 2

V.5.48 org.iso.mpeg.mpegj.scene.EventIn.FontStyle

V.5.48.1 Syntax

```
public static interface EventIn.FontStyle
```

V.5.48.2 Description

V.5.49 org.iso.mpeg.mpegj.scene.EventIn.Form

V.5.49.1 Syntax

```
public static interface EventIn.Form
```

V.5.49.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	children
int	size
int	groups
int	constraints
int	groupsIndex

V.5.49.3 Fields

addChildren
`public static final int addChildren`

addChildren = 0

children
`public static final int children`

children = 2

constraints
`public static final int constraints`

constraints = 5

groups
`public static final int groups`

groups = 4

groupsIndex
`public static final int groupsIndex`

groupsIndex = 6

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000509

removeChildren
`public static final int removeChildren`

removeChildren = 1

size
`public static final int size`

size = 3

V.5.50 org.iso.mpeg.mpegj.scene.EventIn.Group

V.5.50.1 Syntax

`public static interface EventIn.Group`

V.5.50.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	children

V.5.50.3 Fields

addChildren
`public static final int addChildren`

addChildren = 0

children
`public static final int children`

children = 2

removeChildren
`public static final int removeChildren`

removeChildren = 1

V.5.51 org.iso.mpeg.mpegj.scene.EventIn.ImageTexture

V.5.51.1 Syntax

`public static interface EventIn.ImageTexture`

V.5.51.2 Description

Member Summary	
Fields	
int	url

V.5.51.3 Fields

url
`public static final int url`

url = 0

V.5.52 org.iso.mpeg.mpegj.scene.EventIn.IndexedFaceSet

V.5.52.1 Syntax

`public static interface EventIn.IndexedFaceSet`

V.5.52.2 Description

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000510

Member Summary	
Fields	
int	set_colorIndex
int	set_coordIndex
int	set_normalIndex
int	set_texCoordIndex
int	color
int	coord
int	normal
int	texCoord

V.5.52.3 Fields

color
`public static final int color`

color = 4

coord
`public static final int coord`

coord = 5

normal
`public static final int normal`

normal = 6

set_colorIndex
`public static final int set_colorIndex`

set_colorIndex = 0

set_coordIndex
`public static final int set_coordIndex`

set_coordIndex = 1

set_normalIndex
`public static final int set_normalIndex`

set_normalIndex = 2

set_texCoordIndex
`public static final int set_texCoordIndex`

set_texCoordIndex = 3

texCoord
`public static final int texCoord`

texCoord = 7

V.5.53 org.iso.mpeg.mpegj.scene.EventIn.IndexedFaceSet2D

V.5.53.1 Syntax

```
public static interface EventIn.IndexedFaceSet2D
```

V.5.53.2 Description

Member Summary	
Fields	
int	set_colorIndex
int	set_coordIndex
int	set_texCoordIndex
int	color
int	coord
int	texCoord

V.5.53.3 Fields

color
`public static final int color`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000511

color = 3

coord
`public static final int coord`

coord = 4

set_colorIndex
`public static final int set_colorIndex`

set_colorIndex = 0

set_coordIndex
`public static final int set_coordIndex`

set_coordIndex = 1

set_texCoordIndex
`public static final int set_texCoordIndex`

set_texCoordIndex = 2

texCoord
`public static final int texCoord`

texCoord = 5

V.5.54 org.iso.mpeg.mpegj.scene.EventIn.IndexedLineSet

V.5.54.1 Syntax

```
public static interface EventIn.IndexedLineSet
```

V.5.54.2 Description

Member Summary	
Fields	
int	set_colorIndex
int	set_coordIndex
int	color
int	coord

V.5.54.3 Fields

color
`public static final int color`

color = 2

coord
`public static final int coord`

coord = 3

set_colorIndex
`public static final int set_colorIndex`

set_colorIndex = 0

set_coordIndex
`public static final int set_coordIndex`

set_coordIndex = 1

V.5.55 org.iso.mpeg.mpegj.scene.EventIn.IndexedLineSet2D

V.5.55.1 Syntax

```
public static interface EventIn.IndexedLineSet2D
```

V.5.55.2 Description

Member Summary	
Fields	

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000512

int	set_colorIndex
int	set_coordIndex
int	color
int	coord

V.5.55.3 Fields

color
`public static final int color`

color = 2

coord
`public static final int coord`

coord = 3

set_colorIndex
`public static final int set_colorIndex`

set_colorIndex = 0

set_coordIndex
`public static final int set_coordIndex`

set_coordIndex = 1

V.5.56 org.iso.mpeg.mpegj.scene.EventIn.Inline

V.5.56.1 Syntax

`public static interface EventIn.Inline`

V.5.56.2 Description

Member Summary	
Fields	
int	url

V.5.56.3 Fields

url
`public static final int url`

url = 0

V.5.57 org.iso.mpeg.mpegj.scene.EventIn.Layer2D

V.5.57.1 Syntax

`public static interface EventIn.Layer2D`

V.5.57.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	children
int	size
int	background
int	viewport

V.5.57.3 Fields

addChildren
`public static final int addChildren`

addChildren = 0

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000513

background
`public static final int background`

background = 4

children
`public static final int children`

children = 2

removeChildren
`public static final int removeChildren`

removeChildren = 1

size
`public static final int size`

size = 3

viewport
`public static final int viewport`

viewport = 5

V.5.58 org.iso.mpeg.mpegj.scene.EventIn.Layer3D

V.5.58.1 Syntax

```
public static interface EventIn.Layer3D
```

V.5.58.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	children
int	size
int	background
int	fog
int	navigationInfo
int	viewpoint

V.5.58.3 Fields

addChildren
`public static final int addChildren`

addChildren = 0

background
`public static final int background`

background = 4

children
`public static final int children`

children = 2

fog
`public static final int fog`

fog = 5

navigationInfo
`public static final int navigationInfo`

navigationInfo = 6

removeChildren
`public static final int removeChildren`

removeChildren = 1

size

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000514

```
public static final int size
```

size = 3

viewpoint

```
public static final int viewpoint
```

viewpoint = 7

V.5.59 org.iso.mpeg.mpegj.scene.EventIn.Layout

V.5.59.1 Syntax

```
public static interface EventIn.Layout
```

V.5.59.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	children
int	wrap
int	size
int	horizontal
int	justify
int	leftToRight
int	topToBottom
int	spacing
int	smoothScroll
int	loop
int	scrollVertical
int	scrollRate

V.5.59.3 Fields

addChildren

```
public static final int addChildren
```

addChildren = 0

children

```
public static final int children
```

children = 2

horizontal

```
public static final int horizontal
```

horizontal = 5

justify

```
public static final int justify
```

justify = 6

leftToRight

```
public static final int leftToRight
```

leftToRight = 7

loop

```
public static final int loop
```

loop = 11

removeChildren

```
public static final int removeChildren
```

removeChildren = 1

scrollRate

```
public static final int scrollRate
```

scrollRate = 13

scrollVertical

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000515

```
public static final int scrollVertical
```

scrollVertical = 12

size

```
public static final int size
```

size = 4

smoothScroll

```
public static final int smoothScroll
```

smoothScroll = 10

spacing

```
public static final int spacing
```

spacing = 9

topToBottom

```
public static final int topToBottom
```

topToBottom = 8

wrap

```
public static final int wrap
```

wrap = 3

V.5.60 org.iso.mpeg.mpegj.scene.EventIn.LineProperties

V.5.60.1 Syntax

```
public static interface EventIn.LineProperties
```

V.5.60.2 Description

Member Summary	
Fields	
int	lineColor
int	lineStyle
int	width

V.5.60.3 Fields

lineColor

```
public static final int lineColor
```

lineColor = 0

lineStyle

```
public static final int lineStyle
```

lineStyle = 1

width

```
public static final int width
```

width = 2

V.5.61 org.iso.mpeg.mpegj.scene.EventIn.ListeningPoint

V.5.61.1 Syntax

```
public static interface EventIn.ListeningPoint
```

V.5.61.2 Description

Member Summary	
Fields	
int	set_bind
int	jump
int	orientation
int	position

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000516

V.5.61.3 Fields

jump
`public static final int jump`

jump = 1

orientation
`public static final int orientation`

orientation = 2

position
`public static final int position`

position = 3

set_bind
`public static final int set_bind`

set_bind = 0

V.5.62 org.iso.mpeg.mpegj.scene.EventIn.LOD

V.5.62.1 Syntax

```
public static interface EventIn.LOD
```

V.5.62.2 Description

Member Summary	
Fields	
int	level

V.5.62.3 Fields

level
`public static final int level`

level = 0

V.5.63 org.iso.mpeg.mpegj.scene.EventIn.Material

V.5.63.1 Syntax

```
public static interface EventIn.Material
```

V.5.63.2 Description

Member Summary	
Fields	
int	ambientIntensity
int	diffuseColor
int	emissiveColor
int	shininess
int	specularColor
int	transparency

V.5.63.3 Fields

ambientIntensity
`public static final int ambientIntensity`

ambientIntensity = 0

diffuseColor
`public static final int diffuseColor`

diffuseColor = 1

emissiveColor

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000517

```
public static final int emissiveColor
```

emissiveColor = 2

shininess
```
public static final int shininess
```

shininess = 3

specularColor
```
public static final int specularColor
```

specularColor = 4

transparency
```
public static final int transparency
```

transparency = 5

V.5.64 org.iso.mpeg.mpegj.scene.EventIn.Material2D

V.5.64.1 Syntax

```
public static interface EventIn.Material2D
```

V.5.64.2 Description

Member Summary	
Fields	
int	emissiveColor
int	filled
int	lineProps
int	transparency

V.5.64.3 Fields

emissiveColor
```
public static final int emissiveColor
```

emissiveColor = 0

filled
```
public static final int filled
```

filled = 1

lineProps
```
public static final int lineProps
```

lineProps = 2

transparency
```
public static final int transparency
```

transparency = 3

V.5.65 org.iso.mpeg.mpegj.scene.EventIn.MovieTexture

V.5.65.1 Syntax

```
public static interface EventIn.MovieTexture
```

V.5.65.2 Description

Member Summary	
Fields	
int	loop
int	speed
int	startTime
int	stopTime
int	url

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000518

V.5.65.3 Fields

loop
`public static final int loop`

loop = 0

speed
`public static final int speed`

speed = 1

startTime
`public static final int startTime`

startTime = 2

stopTime
`public static final int stopTime`

stopTime = 3

url
`public static final int url`

url = 4

V.5.66 org.iso.mpeg.mpegj.scene.EventIn.NavigationInfo

V.5.66.1 Syntax

```
public static interface EventIn.NavigationInfo
```

V.5.66.2 Description

Member Summary	
Fields	
int	set_bind
int	avatarSize
int	headlight
int	speed
int	type
int	visibilityLimit

V.5.66.3 Fields

avatarSize
`public static final int avatarSize`

avatarSize = 1

headlight
`public static final int headlight`

headlight = 2

set_bind
`public static final int set_bind`

set_bind = 0

speed
`public static final int speed`

speed = 3

type
`public static final int type`

type = 4

visibilityLimit
`public static final int visibilityLimit`

visibilityLimit = 5

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000519

V.5.67 org.iso.mpeg.mpegj.scene.EventIn.Normal

V.5.67.1 Syntax

```
public static interface EventIn.Normal
```

V.5.67.2 Description

Member Summary	
Fields	
int	vector

V.5.67.3 Fields

vector
```
public static final int vector
```
vector = 0

V.5.68 org.iso.mpeg.mpegj.scene.EventIn.NormalInterpolator

V.5.68.1 Syntax

```
public static interface EventIn.NormalInterpolator
```

V.5.68.2 Description

Member Summary	
Fields	
int	set_fraction
int	key
int	keyValue

V.5.68.3 Fields

key
```
public static final int key
```
key = 1

keyValue
```
public static final int keyValue
```
keyValue = 2

set_fraction
```
public static final int set_fraction
```
set_fraction = 0

V.5.69 org.iso.mpeg.mpegj.scene.EventIn.OrderedGroup

V.5.69.1 Syntax

```
public static interface EventIn.OrderedGroup
```

V.5.69.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	children
int	order

V.5.69.3 Fields

addChildren

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000520

```
public static final int addChildren
```

addChildren = 0

children
```
public static final int children
```

children = 2

order
```
public static final int order
```

order = 3

removeChildren
```
public static final int removeChildren
```

removeChildren = 1

V.5.70 org.iso.mpeg.mpegj.scene.EventIn.OrientationInterpolator

V.5.70.1 Syntax

```
public static interface EventIn.OrientationInterpolator
```

V.5.70.2 Description

Member Summary	
Fields	
int	set_fraction
int	key
int	keyValue

V.5.70.3 Fields

key
```
public static final int key
```

key = 1

keyValue
```
public static final int keyValue
```

keyValue = 2

set_fraction
```
public static final int set_fraction
```

set_fraction = 0

V.5.71 org.iso.mpeg.mpegj.scene.EventIn.PixelTexture

V.5.71.1 Syntax

```
public static interface EventIn.PixelTexture
```

V.5.71.2 Description

Member Summary	
Fields	
int	image

V.5.71.3 Fields

image
```
public static final int image
```

image = 0

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000521

V.5.72 org.iso.mpeg.mpegj.scene.EventIn.PlaneSensor

V.5.72.1 Syntax

```
public static interface EventIn.PlaneSensor
```

V.5.72.2 Description

Member Summary	
Fields	
int	autoOffset
int	enabled
int	maxPosition
int	minPosition
int	offset

V.5.72.3 Fields

autoOffset
```
public static final int autoOffset
```
autoOffset = 0

enabled
```
public static final int enabled
```
enabled = 1

maxPosition
```
public static final int maxPosition
```
maxPosition = 2

minPosition
```
public static final int minPosition
```
minPosition = 3

offset
```
public static final int offset
```
offset = 4

V.5.73 org.iso.mpeg.mpegj.scene.EventIn.PlaneSensor2D

V.5.73.1 Syntax

```
public static interface EventIn.PlaneSensor2D
```

V.5.73.2 Description

Member Summary	
Fields	
int	autoOffset
int	enabled
int	maxPosition
int	minPosition
int	offset

V.5.73.3 Fields

autoOffset
```
public static final int autoOffset
```
autoOffset = 0

enabled
```
public static final int enabled
```
enabled = 1

maxPosition

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000522

```
public static final int maxPosition
```

maxPosition = 2

minPosition

```
public static final int minPosition
```

minPosition = 3

offset

```
public static final int offset
```

offset = 4

V.5.74 org.iso.mpeg.mpegj.scene.EventIn.PointLight

V.5.74.1 Syntax

```
public static interface EventIn.PointLight
```

V.5.74.2 Description

Member Summary	
Fields	
int	ambientIntensity
int	attenuation
int	color
int	intensity
int	location
int	on
int	radius

V.5.74.3 Fields

ambientIntensity

```
public static final int ambientIntensity
```

ambientIntensity = 0

attenuation

```
public static final int attenuation
```

attenuation = 1

color

```
public static final int color
```

color = 2

intensity

```
public static final int intensity
```

intensity = 3

location

```
public static final int location
```

location = 4

on

```
public static final int on
```

on = 5

radius

```
public static final int radius
```

radius = 6

V.5.75 org.iso.mpeg.mpegj.scene.EventIn.PointSet

V.5.75.1 Syntax

```
public static interface EventIn.PointSet
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000523

V.5.75.2 Description

Member Summary	
Fields	
int	color
int	coord

V.5.75.3 Fields

color
`public static final int color`

color = 0

coord
`public static final int coord`

coord = 1

V.5.76 org.iso.mpeg.mpegj.scene.EventIn.PointSet2D

V.5.76.1 Syntax

```
public static interface EventIn.PointSet2D
```

V.5.76.2 Description

Member Summary	
Fields	
int	color
int	coord

V.5.76.3 Fields

color
`public static final int color`

color = 0

coord
`public static final int coord`

coord = 1

V.5.77 org.iso.mpeg.mpegj.scene.EventIn.PositionInterpolator

V.5.77.1 Syntax

```
public static interface EventIn.PositionInterpolator
```

V.5.77.2 Description

Member Summary	
Fields	
int	set_fraction
int	key
int	keyValue

V.5.77.3 Fields

key
`public static final int key`

key = 1

keyValue
`public static final int keyValue`

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000524

keyValue = 2

set_fraction
`public static final int set_fraction`

set_fraction = 0

V.5.78 org.iso.mpeg.mpegj.scene.EventIn.PositionInterpolator2D

V.5.78.1 Syntax

```
public static interface EventIn.PositionInterpolator2D
```

V.5.78.2 Description

Member Summary	
Fields	
int	set_fraction
int	key
int	keyValue

V.5.78.3 Fields

key
`public static final int key`

key = 1

keyValue
`public static final int keyValue`

keyValue = 2

set_fraction
`public static final int set_fraction`

set_fraction = 0

V.5.79 org.iso.mpeg.mpegj.scene.EventIn.ProximitySensor

V.5.79.1 Syntax

```
public static interface EventIn.ProximitySensor
```

V.5.79.2 Description

Member Summary	
Fields	
int	center
int	size
int	enabled

V.5.79.3 Fields

center
`public static final int center`

center = 0

enabled
`public static final int enabled`

enabled = 2

size
`public static final int size`

size = 1

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000525

V.5.80 org.iso.mpeg.mpegj.scene.EventIn.ProximitySensor2D

V.5.80.1 Syntax

```
public static interface EventIn.ProximitySensor2D
```

V.5.80.2 Description

Member Summary	
Fields	
int	center
int	size
int	enabled

V.5.80.3 Fields

center
```
public static final int center
```
center = 0

enabled
```
public static final int enabled
```
enabled = 2

size
```
public static final int size
```
size = 1

V.5.81 org.iso.mpeg.mpegj.scene.EventIn.QuantizationParameter

V.5.81.1 Syntax

```
public static interface EventIn.QuantizationParameter
```

V.5.81.2 Description

V.5.82 org.iso.mpeg.mpegj.scene.EventIn.Rectangle

V.5.82.1 Syntax

```
public static interface EventIn.Rectangle
```

V.5.82.2 Description

Member Summary	
Fields	
int	size

V.5.82.3 Fields

size
```
public static final int size
```
size = 0

V.5.83 org.iso.mpeg.mpegj.scene.EventIn.ScalarInterpolator

V.5.83.1 Syntax

```
public static interface EventIn.ScalarInterpolator
```

V.5.83.2 Description

Member Summary	

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000526

Fields		
	int	set_fraction
	int	key
	int	keyValue

V.5.83.3 Fields

key
`public static final int key`

key = 1

keyValue
`public static final int keyValue`

keyValue = 2

set_fraction
`public static final int set_fraction`

set_fraction = 0

V.5.84 org.iso.mpeg.mpegj.scene.EventIn.Script

V.5.84.1 Syntax

```
public static interface EventIn.Script
```

V.5.84.2 Description

Member Summary	
Fields	
int	url

V.5.84.3 Fields

url
`public static final int url`

url = 0

V.5.85 org.iso.mpeg.mpegj.scene.EventIn.Shape

V.5.85.1 Syntax

```
public static interface EventIn.Shape
```

V.5.85.2 Description

Member Summary	
Fields	
int	appearance
int	geometry

V.5.85.3 Fields

appearance
`public static final int appearance`

appearance = 0

geometry
`public static final int geometry`

geometry = 1

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000527

V.5.86 org.iso.mpeg.mpegj.scene.EventIn.Sound

V.5.86.1 Syntax

```
public static interface EventIn.Sound
```

V.5.86.2 Description

Member Summary	
Fields	
int	direction
int	intensity
int	location
int	maxBack
int	maxFront
int	minBack
int	minFront
int	priority
int	source

V.5.86.3 Fields

direction
```
public static final int direction
```
direction = 0

intensity
```
public static final int intensity
```
intensity = 1

location
```
public static final int location
```
location = 2

maxBack
```
public static final int maxBack
```
maxBack = 3

maxFront
```
public static final int maxFront
```
maxFront = 4

minBack
```
public static final int minBack
```
minBack = 5

minFront
```
public static final int minFront
```
minFront = 6

priority
```
public static final int priority
```
priority = 7

source
```
public static final int source
```
source = 8

V.5.87 org.iso.mpeg.mpegj.scene.EventIn.Sound2D

V.5.87.1 Syntax

```
public static interface EventIn.Sound2D
```

V.5.87.2 Description

Member Summary	

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000528

Fields	
int	intensity
int	location
int	source

V.5.87.3 Fields

intensity
`public static final int intensity`

intensity = 0

location
`public static final int location`

location = 1

source
`public static final int source`

source = 2

V.5.88 org.iso.mpeg.mpegj.scene.EventIn.Sphere

V.5.88.1 Syntax

```
public static interface EventIn.Sphere
```

V.5.88.2 Description

V.5.89 org.iso.mpeg.mpegj.scene.EventIn.SphereSensor

V.5.89.1 Syntax

```
public static interface EventIn.SphereSensor
```

V.5.89.2 Description

Member Summary	
Fields	
int	autoOffset
int	enabled
int	offset

V.5.89.3 Fields

autoOffset
`public static final int autoOffset`

autoOffset = 0

enabled
`public static final int enabled`

enabled = 1

offset
`public static final int offset`

offset = 2

V.5.90 org.iso.mpeg.mpegj.scene.EventIn.SpotLight

V.5.90.1 Syntax

```
public static interface EventIn.SpotLight
```

V.5.90.2 Description

Member Summary	

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000529

Fields	
int	ambientIntensity
int	attenuation
int	beamWidth
int	color
int	cutOffAngle
int	direction
int	intensity
int	location
int	on
int	radius

V.5.90.3 Fields

ambientIntensity
`public static final int ambientIntensity`

ambientIntensity = 0

attenuation
`public static final int attenuation`

attenuation = 1

beamWidth
`public static final int beamWidth`

beamWidth = 2

color
`public static final int color`

color = 3

cutOffAngle
`public static final int cutOffAngle`

cutOffAngle = 4

direction
`public static final int direction`

direction = 5

intensity
`public static final int intensity`

intensity = 6

location
`public static final int location`

location = 7

on
`public static final int on`

on = 8

radius
`public static final int radius`

radius = 9

V.5.91 org.iso.mpeg.mpegj.scene.EventIn.Switch

V.5.91.1 Syntax

```
public static interface EventIn.Switch
```

V.5.91.2 Description

Member Summary	
Fields	
int	choice
int	whichChoice

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000530

V.5.91.3 Fields

choice
`public static final int choice`

choice = 0

whichChoice
`public static final int whichChoice`

whichChoice = 1

V.5.92 org.iso.mpeg.mpegj.scene.EventIn.TermCap

V.5.92.1 Syntax

```
public static interface EventIn.TermCap
```

V.5.92.2 Description

Member Summary	
Fields	
int	evaluate
int	capability

V.5.92.3 Fields

capability
`public static final int capability`

capability = 1

evaluate
`public static final int evaluate`

evaluate = 0

V.5.93 org.iso.mpeg.mpegj.scene.EventIn.Text

V.5.93.1 Syntax

```
public static interface EventIn.Text
```

V.5.93.2 Description

Member Summary	
Fields	
int	string
int	length
int	fontStyle
int	maxExtent

V.5.93.3 Fields

fontStyle
`public static final int fontStyle`

fontStyle = 2

length
`public static final int length`

length = 1

maxExtent
`public static final int maxExtent`

maxExtent = 3

string
`public static final int string`

string = 0

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000531

V.5.94 org.iso.mpeg.mpegj.scene.EventIn.TextureCoordinate

V.5.94.1 Syntax

```
public static interface EventIn.TextureCoordinate
```

V.5.94.2 Description

Member Summary	
Fields	
int	point

V.5.94.3 Fields

point
```
public static final int point
```
point = 0

V.5.95 org.iso.mpeg.mpegj.scene.EventIn.TextureTransform

V.5.95.1 Syntax

```
public static interface EventIn.TextureTransform
```

V.5.95.2 Description

Member Summary	
Fields	
int	center
int	rotation
int	scale
int	translation

V.5.95.3 Fields

center
```
public static final int center
```
center = 0

rotation
```
public static final int rotation
```
rotation = 1

scale
```
public static final int scale
```
scale = 2

translation
```
public static final int translation
```
translation = 3

V.5.96 org.iso.mpeg.mpegj.scene.EventIn.TimeSensor

V.5.96.1 Syntax

```
public static interface EventIn.TimeSensor
```

V.5.96.2 Description

Member Summary	
Fields	
int	cycleInterval
int	enabled
int	loop

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000532

int	startTime
int	stopTime

V.5.96.3 Fields

cycleInterval
`public static final int cycleInterval`

cycleInterval = 0

enabled
`public static final int enabled`

enabled = 1

loop
`public static final int loop`

loop = 2

startTime
`public static final int startTime`

startTime = 3

stopTime
`public static final int stopTime`

stopTime = 4

V.5.97 org.iso.mpeg.mpegj.scene.EventIn.TouchSensor

V.5.97.1 Syntax

```
public static interface EventIn.TouchSensor
```

V.5.97.2 Description

Member Summary	
Fields	
int	enabled

V.5.97.3 Fields

enabled
`public static final int enabled`

enabled = 0

V.5.98 org.iso.mpeg.mpegj.scene.EventIn.Transform

V.5.98.1 Syntax

```
public static interface EventIn.Transform
```

V.5.98.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	center
int	children
int	rotation
int	scale
int	scaleOrientation
int	translation

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000533

V.5.98.3 Fields

addChildren
`public static final int addChildren`

addChildren = 0

center
`public static final int center`

center = 2

children
`public static final int children`

children = 3

removeChildren
`public static final int removeChildren`

removeChildren = 1

rotation
`public static final int rotation`

rotation = 4

scale
`public static final int scale`

scale = 5

scaleOrientation
`public static final int scaleOrientation`

scaleOrientation = 6

translation
`public static final int translation`

translation = 7

V.5.99 org.iso.mpeg.mpegj.scene.EventIn.Transform2D

V.5.99.1 Syntax

```
public static interface EventIn.Transform2D
```

V.5.99.2 Description

Member Summary	
Fields	
int	addChildren
int	removeChildren
int	children
int	center
int	rotationAngle
int	scale
int	scaleOrientation
int	translation

V.5.99.3 Fields

addChildren
`public static final int addChildren`

addChildren = 0

center
`public static final int center`

center = 3

children
`public static final int children`

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000534

children = 2

removeChildren
`public static final int removeChildren`

removeChildren = 1

rotationAngle
`public static final int rotationAngle`

rotationAngle = 4

scale
`public static final int scale`

scale = 5

scaleOrientation
`public static final int scaleOrientation`

scaleOrientation = 6

translation
`public static final int translation`

translation = 7

V.5.100 org.iso.mpeg.mpegj.scene.EventIn.Valuator

V.5.100.1 Syntax

```
public static interface EventIn.Valuator
```

V.5.100.2 Description

Member Summary	
Fields	
int	inSFBool
int	inSFColor
int	inMFColor
int	inSFFloat
int	inMFFloat
int	inSFInt32
int	inMFInt32
int	inSFRotation
int	inMFRotation
int	inSFString
int	inMFString
int	inSFTime
int	inSFVec2f
int	inMFVec2f
int	inSFVec3f
int	inMFVec3f
int	Factor1
int	Factor2
int	Factor3
int	Factor4
int	Offset1
int	Offset2
int	Offset3
int	Offset4
int	Sum

V.5.100.3 Fields

Factor1
`public static final int Factor1`

Factor1 = 16

Factor2
`public static final int Factor2`

Factor2 = 17

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000535

Factor3
`public static final int Factor3`

Factor3 = 18

Factor4
`public static final int Factor4`

Factor4 = 19

inMFColor
`public static final int inMFColor`

inMFColor = 2

inMFFloat
`public static final int inMFFloat`

inMFFloat = 4

inMFInt32
`public static final int inMFInt32`

inMFInt32 = 6

inMFRotation
`public static final int inMFRotation`

inMFRotation = 8

inMFString
`public static final int inMFString`

inMFString = 10

inMFVec2f
`public static final int inMFVec2f`

inMFVec2f = 13

inMFVec3f
`public static final int inMFVec3f`

inMFVec3f = 15

inSFBool
`public static final int inSFBool`

inSFBool = 0

inSFColor
`public static final int inSFColor`

inSFColor = 1

inSFFloat
`public static final int inSFFloat`

inSFFloat = 3

inSFInt32
`public static final int inSFInt32`

inSFInt32 = 5

inSFRotation
`public static final int inSFRotation`

inSFRotation = 7

inSFString
`public static final int inSFString`

inSFString = 9

inSFTime
`public static final int inSFTime`

inSFTime = 11

inSFVec2f
`public static final int inSFVec2f`

inSFVec2f = 12

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000536

inSFVec3f
`public static final int inSFVec3f`

inSFVec3f = 14

Offset1
`public static final int Offset1`

Offset1 = 20

Offset2
`public static final int Offset2`

Offset2 = 21

Offset3
`public static final int Offset3`

Offset3 = 22

Offset4
`public static final int Offset4`

Offset4 = 23

Sum
`public static final int Sum`

Sum = 24

V.5.101 org.iso.mpeg.mpegj.scene.EventIn.Viewpoint

V.5.101.1 Syntax

```
public static interface EventIn.Viewpoint
```

V.5.101.2 Description

Member Summary	
Fields	
int	set_bind
int	fieldOfView
int	jump
int	orientation
int	position

V.5.101.3 Fields

fieldOfView
`public static final int fieldOfView`

fieldOfView = 1

jump
`public static final int jump`

jump = 2

orientation
`public static final int orientation`

orientation = 3

position
`public static final int position`

position = 4

set_bind
`public static final int set_bind`

set_bind = 0

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000537

V.5.102 org.iso.mpeg.mpegj.scene.EventIn.Viseme

V.5.102.1 Syntax

```
public static interface EventIn.Viseme
```

V.5.102.2 Description

Member Summary	
Fields	
int	viseme_select1
int	viseme_select2
int	viseme_blend
int	viseme_def

V.5.102.3 Fields

viseme_blend
```
public static final int viseme_blend
```
viseme_blend = 2

viseme_def
```
public static final int viseme_def
```
viseme_def = 3

viseme_select1
```
public static final int viseme_select1
```
viseme_select1 = 0

viseme_select2
```
public static final int viseme_select2
```
viseme_select2 = 1

V.5.103 org.iso.mpeg.mpegj.scene.EventIn.VisibilitySensor

V.5.103.1 Syntax

```
public static interface EventIn.VisibilitySensor
```

V.5.103.2 Description

Member Summary	
Fields	
int	center
int	enabled
int	size

V.5.103.3 Fields

center
```
public static final int center
```
center = 0

enabled
```
public static final int enabled
```
enabled = 1

size
```
public static final int size
```
size = 2

V.5.104 org.iso.mpeg.mpegj.scene.EventIn.WorldInfo

V.5.104.1 Syntax

```
public static interface EventIn.WorldInfo
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000538

V.5.104.2 Description

V.5.105 org.iso.mpeg.mpegj.scene.EventOut

V.5.105.1 Syntax

```
public interface EventOut
```

V.5.105.2 Description

Member Summary	
Inner Classes	
static interface	EventOut.Anchor
static interface	EventOut.AnimationStream
static interface	EventOut.Appearance
static interface	EventOut.AudioBuffer
static interface	EventOut.AudioClip
static interface	EventOut.AudioDelay
static interface	EventOut.AudioFX
static interface	EventOut.AudioMix
static interface	EventOut.AudioSource
static interface	EventOut.AudioSwitch
static interface	EventOut.Background
static interface	EventOut.Background2D
static interface	EventOut.Billboard
static interface	EventOut.Bitmap
static interface	EventOut.Box
static interface	EventOut.Circle
static interface	EventOut.Collision
static interface	EventOut.Color
static interface	EventOut.ColorInterpolator
static interface	EventOut.CompositeTexture2D
static interface	EventOut.CompositeTexture3D
static interface	EventOut.Conditional
static interface	EventOut.Cone
static interface	EventOut.Coordinate
static interface	EventOut.Coordinate2D
static interface	EventOut.CoordinateInterpolator
static interface	EventOut.CoordinateInterpolator2D
static interface	EventOut.Curve2D
static interface	EventOut.Cylinder
static interface	EventOut.CylinderSensor
static interface	EventOut.DirectionalLight
static interface	EventOut.DiscSensor
static interface	EventOut.ElevationGrid
static interface	EventOut.Expression
static interface	EventOut.Extrusion
static interface	EventOut.Face
static interface	EventOut.FaceDefMesh
static interface	EventOut.FaceDefTables
static interface	EventOut.FaceDefTransform
static interface	EventOut.FAP
static interface	EventOut.FDP
static interface	EventOut.FIT
static interface	EventOut.Fog
static interface	EventOut.FontStyle
static interface	EventOut.Form
static interface	EventOut.Group
static interface	EventOut.ImageTexture
static interface	EventOut.IndexedFaceSet
static interface	EventOut.IndexedFaceSet2D
static interface	EventOut.IndexedLineSet
static interface	EventOut.IndexedLineSet2D
static interface	EventOut.Inline
static interface	EventOut.LOD
static interface	EventOut.Layer2D
static interface	EventOut.Layer3D
static interface	EventOut.Layout
static interface	EventOut.LineProperties
static interface	EventOut.ListeningPoint
static interface	EventOut.Material

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000539

static interface	EventOut.Material2D
static interface	EventOut.MovieTexture
static interface	EventOut.NavigationInfo
static interface	EventOut.Normal
static interface	EventOut.NormalInterpolator
static interface	EventOut.OrderedGroup
static interface	EventOut.OrientationInterpolator
static interface	EventOut.PixelTexture
static interface	EventOut.PlaneSensor
static interface	EventOut.PlaneSensor2D
static interface	EventOut.PointLight
static interface	EventOut.PointSet
static interface	EventOut.PointSet2D
static interface	EventOut.PositionInterpolator
static interface	EventOut.PositionInterpolator2D
static interface	EventOut.ProximitySensor2D
static interface	EventOut.ProximitySensor
static interface	EventOut.QuantizationParameter
static interface	EventOut.Rectangle
static interface	EventOut.ScalarInterpolator
static interface	EventOut.Script
static interface	EventOut.Shape
static interface	EventOut.Sound
static interface	EventOut.Sound2D
static interface	EventOut.Sphere
static interface	EventOut.SphereSensor
static interface	EventOut.SpotLight
static interface	EventOut.Switch
static interface	EventOut.TermCap
static interface	EventOut.Text
static interface	EventOut.TextureCoordinate
static interface	EventOut.TextureTransform
static interface	EventOut.TimeSensor
static interface	EventOut.TouchSensor
static interface	EventOut.Transform
static interface	EventOut.Transform2D
static interface	EventOut.Valuator
static interface	EventOut.Viewpoint
static interface	EventOut.VisibilitySensor
static interface	EventOut.Viseme
static interface	EventOut.WorldInfo

V.5.106 org.iso.mpeg.mpegj.scene.EventOut.Anchor

V.5.106.1 Syntax

```
public static interface EventOut.Anchor
```

V.5.106.2 Description

Member Summary	
Fields	
int	children
int	description
int	parameter
int	url

V.5.106.3 Fields

children
```
public static final int children
```

children = 0

description
```
public static final int description
```

description = 1

parameter
```
public static final int parameter
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000540

parameter = 2

url
`public static final int url`

url = 3

V.5.107 org.iso.mpeg.mpegj.scene.EventOut.AnimationStream

V.5.107.1 Syntax

```
public static interface EventOut.AnimationStream
```

V.5.107.2 Description

Member Summary	
Fields	
int	loop
int	speed
int	startTime
int	stopTime
int	url
int	duration_changed
int	isActive

V.5.107.3 Fields

duration_changed
`public static final int duration_changed`

duration_changed = 5

isActive
`public static final int isActive`

isActive = 6

loop
`public static final int loop`

loop = 0

speed
`public static final int speed`

speed = 1

startTime
`public static final int startTime`

startTime = 2

stopTime
`public static final int stopTime`

stopTime = 3

url
`public static final int url`

url = 4

V.5.108 org.iso.mpeg.mpegj.scene.EventOut.Appearance

V.5.108.1 Syntax

```
public static interface EventOut.Appearance
```

V.5.108.2 Description

Member Summary	
Fields	
int	material

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000541

int	texture
int	textureTransform

V.5.108.3 Fields

material
`public static final int material`

material = 0

texture
`public static final int texture`

texture = 1

textureTransform
`public static final int textureTransform`

textureTransform = 2

V.5.109 org.iso.mpeg.mpegj.scene.EventOut.AudioBuffer

V.5.109.1 Syntax

```
public static interface EventOut.AudioBuffer
```

V.5.109.2 Description

Member Summary	
Fields	
int	loop
int	pitch
int	startTime
int	stopTime
int	children
int	numChan
int	phaseGroup
int	duration_changed
int	isActive

V.5.109.3 Fields

children
`public static final int children`

children = 4

duration_changed
`public static final int duration_changed`

duration_changed = 7

isActive
`public static final int isActive`

isActive = 8

loop
`public static final int loop`

loop = 0

numChan
`public static final int numChan`

numChan = 5

phaseGroup
`public static final int phaseGroup`

phaseGroup = 6

pitch
`public static final int pitch`

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000542

pitch = 1

startTime
`public static final int startTime`

startTime = 2

stopTime
`public static final int stopTime`

stopTime = 3

V.5.110 org.iso.mpeg.mpegj.scene.EventOut.AudioClip

V.5.110.1 Syntax

```
public static interface EventOut.AudioClip
```

V.5.110.2 Description

Member Summary	
Fields	
int	description
int	loop
int	pitch
int	startTime
int	stopTime
int	url
int	duration_changed
int	isActive

V.5.110.3 Fields

description
`public static final int description`

description = 0

duration_changed
`public static final int duration_changed`

duration_changed = 6

isActive
`public static final int isActive`

isActive = 7

loop
`public static final int loop`

loop = 1

pitch
`public static final int pitch`

pitch = 2

startTime
`public static final int startTime`

startTime = 3

stopTime
`public static final int stopTime`

stopTime = 4

url
`public static final int url`

url = 5

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000543

V.5.111 org.iso.mpeg.mpegj.scene.EventOut.AudioDelay

V.5.111.1 Syntax

```
public static interface EventOut.AudioDelay
```

V.5.111.2 Description

Member Summary	
Fields	
int	children
int	delay

V.5.111.3 Fields

children
```
public static final int children
```
children = 0

delay
```
public static final int delay
```
delay = 1

V.5.112 org.iso.mpeg.mpegj.scene.EventOut.AudioFX

V.5.112.1 Syntax

```
public static interface EventOut.AudioFX
```

V.5.112.2 Description

Member Summary	
Fields	
int	children
int	orch
int	score
int	params

V.5.112.3 Fields

children
```
public static final int children
```
children = 0

orch
```
public static final int orch
```
orch = 1

params
```
public static final int params
```
params = 3

score
```
public static final int score
```
score = 2

V.5.113 org.iso.mpeg.mpegj.scene.EventOut.AudioMix

V.5.113.1 Syntax

```
public static interface EventOut.AudioMix
```

V.5.113.2 Description

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000544

Member Summary	
Fields	
int	children
int	numInputs
int	matrix

V.5.113.3 Fields

children
`public static final int children`

children = 0

matrix
`public static final int matrix`

matrix = 2

numInputs
`public static final int numInputs`

numInputs = 1

V.5.114 org.iso.mpeg.mpegj.scene.EventOut.AudioSource

V.5.114.1 Syntax

```
public static interface EventOut.AudioSource
```

V.5.114.2 Description

Member Summary	
Fields	
int	children
int	url
int	pitch
int	speed
int	startTime
int	stopTime

V.5.114.3 Fields

children
`public static final int children`

children = 0

pitch
`public static final int pitch`

pitch = 2

speed
`public static final int speed`

speed = 3

startTime
`public static final int startTime`

startTime = 4

stopTime
`public static final int stopTime`

stopTime = 5

url
`public static final int url`

url = 1

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000545

V.5.115 org.iso.mpeg.mpegj.scene.EventOut.AudioSwitch

V.5.115.1 Syntax

```
public static interface EventOut.AudioSwitch
```

V.5.115.2 Description

Member Summary	
Fields	
int	children
int	whichChoice

V.5.115.3 Fields

children
```
public static final int children
```
children = 0

whichChoice
```
public static final int whichChoice
```
whichChoice = 1

V.5.116 org.iso.mpeg.mpegj.scene.EventOut.Background

V.5.116.1 Syntax

```
public static interface EventOut.Background
```

V.5.116.2 Description

Member Summary	
Fields	
int	groundAngle
int	groundColor
int	backUrl
int	bottomUrl
int	frontUrl
int	leftUrl
int	rightUrl
int	topUrl
int	skyAngle
int	skyColor
int	isBound

V.5.116.3 Fields

backUrl
```
public static final int backUrl
```
backUrl = 2

bottomUrl
```
public static final int bottomUrl
```
bottomUrl = 3

frontUrl
```
public static final int frontUrl
```
frontUrl = 4

groundAngle
```
public static final int groundAngle
```
groundAngle = 0

groundColor
```
public static final int groundColor
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000546

groundColor = 1

isBound
`public static final int isBound`

isBound = 10

leftUrl
`public static final int leftUrl`

leftUrl = 5

rightUrl
`public static final int rightUrl`

rightUrl = 6

skyAngle
`public static final int skyAngle`

skyAngle = 8

skyColor
`public static final int skyColor`

skyColor = 9

topUrl
`public static final int topUrl`

topUrl = 7

V.5.117 org.iso.mpeg.mpegj.scene.EventOut.Background2D

V.5.117.1 Syntax

```
public static interface EventOut.Background2D
```

V.5.117.2 Description

Member Summary	
Fields	
int	backColor
int	url
int	isBound

V.5.117.3 Fields

backColor
`public static final int backColor`

backColor = 0

isBound
`public static final int isBound`

isBound = 2

url
`public static final int url`

url = 1

V.5.118 org.iso.mpeg.mpegj.scene.EventOut.Billboard

V.5.118.1 Syntax

```
public static interface EventOut.Billboard
```

V.5.118.2 Description

Member Summary	
Fields	
int	children

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000547

int	axisOfRotation

V.5.118.3 Fields

axisOfRotation
`public static final int axisOfRotation`

axisOfRotation = 1

children
`public static final int children`

children = 0

V.5.119 org.iso.mpeg.mpegj.scene.EventOut.Bitmap

V.5.119.1 Syntax

```
public static interface EventOut.Bitmap
```

V.5.119.2 Description

Member Summary	
Fields	
int	scale

V.5.119.3 Fields

scale
`public static final int scale`

scale = 0

V.5.120 org.iso.mpeg.mpegj.scene.EventOut.Box

V.5.120.1 Syntax

```
public static interface EventOut.Box
```

V.5.120.2 Description

V.5.121 org.iso.mpeg.mpegj.scene.EventOut.Circle

V.5.121.1 Syntax

```
public static interface EventOut.Circle
```

V.5.121.2 Description

Member Summary	
Fields	
int	radius

V.5.121.3 Fields

radius
`public static final int radius`

radius = 0

V.5.122 org.iso.mpeg.mpegj.scene.EventOut.Collision

V.5.122.1 Syntax

```
public static interface EventOut.Collision
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000548

V.5.122.2 Description

Member Summary	
Fields	
int	children
int	collide
int	collideTime

V.5.122.3 Fields

children
`public static final int children`

children = 0

collide
`public static final int collide`

collide = 1

collideTime
`public static final int collideTime`

collideTime = 2

V.5.123 org.iso.mpeg.mpegj.scene.EventOut.Color

V.5.123.1 Syntax

```
public static interface EventOut.Color
```

V.5.123.2 Description

Member Summary	
Fields	
int	color

V.5.123.3 Fields

color
`public static final int color`

color = 0

V.5.124 org.iso.mpeg.mpegj.scene.EventOut.ColorInterpolator

V.5.124.1 Syntax

```
public static interface EventOut.ColorInterpolator
```

V.5.124.2 Description

Member Summary	
Fields	
int	key
int	keyValue
int	value_changed

V.5.124.3 Fields

key
`public static final int key`

key = 0

keyValue
`public static final int keyValue`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000549

keyValue = 1

value_changed
`public static final int value_changed`

value_changed = 2

V.5.125 org.iso.mpeg.mpegj.scene.EventOut.CompositeTexture2D

V.5.125.1 Syntax

```
public static interface EventOut.CompositeTexture2D
```

V.5.125.2 Description

Member Summary	
Fields	
int	children
int	pixelWidth
int	pixelHeight
int	background
int	viewport

V.5.125.3 Fields

background
`public static final int background`

background = 3

children
`public static final int children`

children = 0

pixelHeight
`public static final int pixelHeight`

pixelHeight = 2

pixelWidth
`public static final int pixelWidth`

pixelWidth = 1

viewport
`public static final int viewport`

viewport = 4

V.5.126 org.iso.mpeg.mpegj.scene.EventOut.CompositeTexture3D

V.5.126.1 Syntax

```
public static interface EventOut.CompositeTexture3D
```

V.5.126.2 Description

Member Summary	
Fields	
int	children
int	pixelWidth
int	pixelHeight
int	background
int	fog
int	navigationInfo
int	viewpoint

V.5.126.3 Fields

background

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000550

```
public static final int background
```

background = 3

children

```
public static final int children
```

children = 0

fog

```
public static final int fog
```

fog = 4

navigationInfo

```
public static final int navigationInfo
```

navigationInfo = 5

pixelHeight

```
public static final int pixelHeight
```

pixelHeight = 2

pixelWidth

```
public static final int pixelWidth
```

pixelWidth = 1

viewpoint

```
public static final int viewpoint
```

viewpoint = 6

V.5.127 org.iso.mpeg.mpegj.scene.EventOut.Conditional

V.5.127.1 Syntax

```
public static interface EventOut.Conditional
```

V.5.127.2 Description

Member Summary	
Fields	
int	buffer
int	isActive

V.5.127.3 Fields

buffer

```
public static final int buffer
```

buffer = 0

isActive

```
public static final int isActive
```

isActive = 1

V.5.128 org.iso.mpeg.mpegj.scene.EventOut.Cone

V.5.128.1 Syntax

```
public static interface EventOut.Cone
```

V.5.128.2 Description

V.5.129 org.iso.mpeg.mpegj.scene.EventOut.Coordinate

V.5.129.1 Syntax

```
public static interface EventOut.Coordinate
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000551

V.5.129.2 Description

Member Summary	
Fields	
int	point

V.5.129.3 Fields

point
`public static final int point`

point = 0

V.5.130 org.iso.mpeg.mpegj.scene.EventOut.Coordinate2D

V.5.130.1 Syntax

```
public static interface EventOut.Coordinate2D
```

V.5.130.2 Description

Member Summary	
Fields	
int	point

V.5.130.3 Fields

point
`public static final int point`

point = 0

V.5.131 org.iso.mpeg.mpegj.scene.EventOut.CoordinateInterpolator

V.5.131.1 Syntax

```
public static interface EventOut.CoordinateInterpolator
```

V.5.131.2 Description

Member Summary	
Fields	
int	key
int	keyValue
int	value_changed

V.5.131.3 Fields

key
`public static final int key`

key = 0

keyValue
`public static final int keyValue`

keyValue = 1

value_changed
`public static final int value_changed`

value_changed = 2

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000552

V.5.132 org.iso.mpeg.mpegj.scene.EventOut.CoordinateInterpolator2D

V.5.132.1 Syntax

```
public static interface EventOut.CoordinateInterpolator2D
```

V.5.132.2 Description

Member Summary	
Fields	
int	key
int	keyValue
int	value_changed

V.5.132.3 Fields

key
```
public static final int key
```
key = 0

keyValue
```
public static final int keyValue
```
keyValue = 1

value_changed
```
public static final int value_changed
```
value_changed = 2

V.5.133 org.iso.mpeg.mpegj.scene.EventOut.Curve2D

V.5.133.1 Syntax

```
public static interface EventOut.Curve2D
```

V.5.133.2 Description

Member Summary	
Fields	
int	point
int	fineness
int	type

V.5.133.3 Fields

fineness
```
public static final int fineness
```
fineness = 1

point
```
public static final int point
```
point = 0

type
```
public static final int type
```
type = 2

V.5.134 org.iso.mpeg.mpegj.scene.EventOut.Cylinder

V.5.134.1 Syntax

```
public static interface EventOut.Cylinder
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000553

V.5.134.2 Description

V.5.135 org.iso.mpeg.mpegj.scene.EventOut.CylinderSensor

V.5.135.1 Syntax

```
public static interface EventOut.CylinderSensor
```

V.5.135.2 Description

Member Summary	
Fields	
int	autoOffset
int	diskAngle
int	enabled
int	maxAngle
int	minAngle
int	offset
int	isActive
int	rotation_changed
int	trackPoint_changed

V.5.135.3 Fields

autoOffset
```
public static final int autoOffset
```
autoOffset = 0

diskAngle
```
public static final int diskAngle
```
diskAngle = 1

enabled
```
public static final int enabled
```
enabled = 2

isActive
```
public static final int isActive
```
isActive = 6

maxAngle
```
public static final int maxAngle
```
maxAngle = 3

minAngle
```
public static final int minAngle
```
minAngle = 4

offset
```
public static final int offset
```
offset = 5

rotation_changed
```
public static final int rotation_changed
```
rotation_changed = 7

trackPoint_changed
```
public static final int trackPoint_changed
```
trackPoint_changed = 8

V.5.136 org.iso.mpeg.mpegj.scene.EventOut.DirectionalLight

V.5.136.1 Syntax

```
public static interface EventOut.DirectionalLight
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000554

V.5.136.2 Description

Member Summary	
Fields	
int	ambientIntensity
int	color
int	direction
int	intensity
int	on

V.5.136.3 Fields

ambientIntensity
`public static final int ambientIntensity`

ambientIntensity = 0

color
`public static final int color`

color = 1

direction
`public static final int direction`

direction = 2

intensity
`public static final int intensity`

intensity = 3

on
`public static final int on`

on = 4

V.5.137 org.iso.mpeg.mpegj.scene.EventOut.DiscSensor

V.5.137.1 Syntax

```
public static interface EventOut.DiscSensor
```

V.5.137.2 Description

Member Summary	
Fields	
int	autoOffset
int	enabled
int	maxAngle
int	minAngle
int	offset
int	isActive
int	rotation_changed
int	trackPoint_changed

V.5.137.3 Fields

autoOffset
`public static final int autoOffset`

autoOffset = 0

enabled
`public static final int enabled`

enabled = 1

isActive
`public static final int isActive`

isActive = 5

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000555

maxAngle
`public static final int maxAngle`

maxAngle = 2

minAngle
`public static final int minAngle`

minAngle = 3

offset
`public static final int offset`

offset = 4

rotation_changed
`public static final int rotation_changed`

rotation_changed = 6

trackPoint_changed
`public static final int trackPoint_changed`

trackPoint_changed = 7

V.5.138 org.iso.mpeg.mpegj.scene.EventOut.ElevationGrid

V.5.138.1 Syntax

```
public static interface EventOut.ElevationGrid
```

V.5.138.2 Description

Member Summary	
Fields	
int	color
int	normal
int	texCoord

V.5.138.3 Fields

color
`public static final int color`

color = 0

normal
`public static final int normal`

normal = 1

texCoord
`public static final int texCoord`

texCoord = 2

V.5.139 org.iso.mpeg.mpegj.scene.EventOut.Expression

V.5.139.1 Syntax

```
public static interface EventOut.Expression
```

V.5.139.2 Description

Member Summary	
Fields	
int	expression_select1
int	expression_intensity1
int	expression_select2
int	expression_intensity2
int	init_face
int	expression_def

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000556

V.5.139.3 Fields

expression_def
`public static final int expression_def`

expression_def = 5

expression_intensity1
`public static final int expression_intensity1`

expression_intensity1 = 1

expression_intensity2
`public static final int expression_intensity2`

expression_intensity2 = 3

expression_select1
`public static final int expression_select1`

expression_select1 = 0

expression_select2
`public static final int expression_select2`

expression_select2 = 2

init_face
`public static final int init_face`

init_face = 4

V.5.140 org.iso.mpeg.mpegj.scene.EventOut.Extrusion

V.5.140.1 Syntax

```
public static interface EventOut.Extrusion
```

V.5.140.2 Description

V.5.141 org.iso.mpeg.mpegj.scene.EventOut.Face

V.5.141.1 Syntax

```
public static interface EventOut.Face
```

V.5.141.2 Description

Member Summary	
Fields	
int	fap
int	fdp
int	fit
int	ttsSource
int	renderedFace

V.5.141.3 Fields

fap
`public static final int fap`

fap = 0

fdp
`public static final int fdp`

fdp = 1

fit
`public static final int fit`

fit = 2

renderedFace

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000557

```
public static final int renderedFace
```

renderedFace = 4

ttsSource

```
public static final int ttsSource
```

ttsSource = 3

V.5.142 org.iso.mpeg.mpegj.scene.EventOut.FaceDefMesh

V.5.142.1 Syntax

```
public static interface EventOut.FaceDefMesh
```

V.5.142.2 Description

V.5.143 org.iso.mpeg.mpegj.scene.EventOut.FaceDefTables

V.5.143.1 Syntax

```
public static interface EventOut.FaceDefTables
```

V.5.143.2 Description

Member Summary	
Fields	
int	faceDefMesh
int	faceDefTransform

V.5.143.3 Fields

faceDefMesh

```
public static final int faceDefMesh
```

faceDefMesh = 0

faceDefTransform

```
public static final int faceDefTransform
```

faceDefTransform = 1

V.5.144 org.iso.mpeg.mpegj.scene.EventOut.FaceDefTransform

V.5.144.1 Syntax

```
public static interface EventOut.FaceDefTransform
```

V.5.144.2 Description

V.5.145 org.iso.mpeg.mpegj.scene.EventOut.FAP

V.5.145.1 Syntax

```
public static interface EventOut.FAP
```

V.5.145.2 Description

Member Summary	
Fields	
int	viseme
int	expression
int	open_jaw
int	lower_t_midlip
int	raise_b_midlip
int	stretch_l_corner
int	stretch_r_corner
int	lower_t_lip_lm

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000558

int	lower_t_lip_rm
int	lower_b_lip_lm
int	lower_b_lip_rm
int	raise_l_cornerlip
int	raise_r_cornerlip
int	thrust_jaw
int	shift_jaw
int	push_b_lip
int	push_t_lip
int	depress_chin
int	close_t_l_eyelid
int	close_t_r_eyelid
int	close_b_l_eyelid
int	close_b_r_eyelid
int	yaw_l_eyeball
int	yaw_r_eyeball
int	pitch_l_eyeball
int	pitch_r_eyeball
int	thrust_l_eyeball
int	thrust_r_eyeball
int	dilate_l_pupil
int	dilate_r_pupil
int	raise_l_i_eyebrow
int	raise_r_i_eyebrow
int	raise_l_m_eyebrow
int	raise_r_m_eyebrow
int	raise_l_o_eyebrow
int	raise_r_o_eyebrow
int	squeeze_l_eyebrow
int	squeeze_r_eyebrow
int	puff_l_cheek
int	puff_r_cheek
int	lift_l_cheek
int	lift_r_cheek
int	shift_tongue_tip
int	raise_tongue_tip
int	thrust_tongue_tip
int	raise_tongue
int	tongue_roll
int	head_pitch
int	head_yaw
int	head_roll
int	lower_t_midlip_o
int	raise_b_midlip_o
int	stretch_l_cornerlip
int	stretch_r_cornerlip
int	lower_t_lip_lm_o
int	lower_t_lip_rm_o
int	raise_b_lip_lm_o
int	raise_b_lip_rm_o
int	raise_l_cornerlip_o
int	raise_r_cornerlip_o
int	stretch_l_nose
int	stretch_r_nose
int	raise_nose
int	bend_nose
int	raise_l_ear
int	raise_r_ear
int	pull_l_ear
int	pull_r_ear

V.5.145.3 Fields

bend_nose
`public static final int bend_nose`

bend_nose = 63

close_b_l_eyelid
`public static final int close_b_l_eyelid`

close_b_l_eyelid = 20

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000559

close_b_r_eyelid
`public static final int close_b_r_eyelid`

close_b_r_eyelid = 21

close_t_l_eyelid
`public static final int close_t_l_eyelid`

close_t_l_eyelid = 18

close_t_r_eyelid
`public static final int close_t_r_eyelid`

close_t_r_eyelid = 19

depress_chin
`public static final int depress_chin`

depress_chin = 17

dilate_l_pupil
`public static final int dilate_l_pupil`

dilate_l_pupil = 28

dilate_r_pupil
`public static final int dilate_r_pupil`

dilate_r_pupil = 29

expression
`public static final int expression`

expression = 1

head_pitch
`public static final int head_pitch`

head_pitch = 47

head_roll
`public static final int head_roll`

head_roll = 49

head_yaw
`public static final int head_yaw`

head_yaw = 48

lift_l_cheek
`public static final int lift_l_cheek`

lift_l_cheek = 40

lift_r_cheek
`public static final int lift_r_cheek`

lift_r_cheek = 41

lower_b_lip_lm
`public static final int lower_b_lip_lm`

lower_b_lip_lm = 9

lower_b_lip_rm
`public static final int lower_b_lip_rm`

lower_b_lip_rm = 10

lower_t_lip_lm
`public static final int lower_t_lip_lm`

lower_t_lip_lm = 7

lower_t_lip_lm_o
`public static final int lower_t_lip_lm_o`

lower_t_lip_lm_o = 54

lower_t_lip_rm
`public static final int lower_t_lip_rm`

lower_t_lip_rm = 8

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000560

lower_t_lip_rm_o
`public static final int lower_t_lip_rm_o`

lower_t_lip_rm_o = 55

lower_t_midlip
`public static final int lower_t_midlip`

lower_t_midlip = 3

lower_t_midlip_o
`public static final int lower_t_midlip_o`

lower_t_midlip_o = 50

open_jaw
`public static final int open_jaw`

open_jaw = 2

pitch_l_eyeball
`public static final int pitch_l_eyeball`

pitch_l_eyeball = 24

pitch_r_eyeball
`public static final int pitch_r_eyeball`

pitch_r_eyeball = 25

puff_l_cheek
`public static final int puff_l_cheek`

puff_l_cheek = 38

puff_r_cheek
`public static final int puff_r_cheek`

puff_r_cheek = 39

pull_l_ear
`public static final int pull_l_ear`

pull_l_ear = 66

pull_r_ear
`public static final int pull_r_ear`

pull_r_ear = 67

push_b_lip
`public static final int push_b_lip`

push_b_lip = 15

push_t_lip
`public static final int push_t_lip`

push_t_lip = 16

raise_b_lip_lm_o
`public static final int raise_b_lip_lm_o`

raise_b_lip_lm_o = 56

raise_b_lip_rm_o
`public static final int raise_b_lip_rm_o`

raise_b_lip_rm_o = 57

raise_b_midlip
`public static final int raise_b_midlip`

raise_b_midlip = 4

raise_b_midlip_o
`public static final int raise_b_midlip_o`

raise_b_midlip_o = 51

raise_l_cornerlip
`public static final int raise_l_cornerlip`

raise_l_cornerlip = 11

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000561

raise_l_cornerlip_o
`public static final int raise_l_cornerlip_o`

raise_l_cornerlip_o = 58

raise_l_ear
`public static final int raise_l_ear`

raise_l_ear = 64

raise_l_i_eyebrow
`public static final int raise_l_i_eyebrow`

raise_l_i_eyebrow = 30

raise_l_m_eyebrow
`public static final int raise_l_m_eyebrow`

raise_l_m_eyebrow = 32

raise_l_o_eyebrow
`public static final int raise_l_o_eyebrow`

raise_l_o_eyebrow = 34

raise_nose
`public static final int raise_nose`

raise_nose = 62

raise_r_cornerlip
`public static final int raise_r_cornerlip`

raise_r_cornerlip = 12

raise_r_cornerlip_o
`public static final int raise_r_cornerlip_o`

raise_r_cornerlip_o = 59

raise_r_ear
`public static final int raise_r_ear`

raise_r_ear = 65

raise_r_i_eyebrow
`public static final int raise_r_i_eyebrow`

raise_r_i_eyebrow = 31

raise_r_m_eyebrow
`public static final int raise_r_m_eyebrow`

raise_r_m_eyebrow = 33

raise_r_o_eyebrow
`public static final int raise_r_o_eyebrow`

raise_r_o_eyebrow = 35

raise_tongue
`public static final int raise_tongue`

raise_tongue = 45

raise_tongue_tip
`public static final int raise_tongue_tip`

raise_tongue_tip = 43

shift_jaw
`public static final int shift_jaw`

shift_jaw = 14

shift_tongue_tip
`public static final int shift_tongue_tip`

shift_tongue_tip = 42

squeeze_l_eyebrow
`public static final int squeeze_l_eyebrow`

squeeze_l_eyebrow = 36

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000562

squeeze_r_eyebrow
`public static final int squeeze_r_eyebrow`

squeeze_r_eyebrow = 37

stretch_l_corner
`public static final int stretch_l_corner`

stretch_l_corner = 5

stretch_l_cornerlip
`public static final int stretch_l_cornerlip`

stretch_l_cornerlip = 52

stretch_l_nose
`public static final int stretch_l_nose`

stretch_l_nose = 60

stretch_r_corner
`public static final int stretch_r_corner`

stretch_r_corner = 6

stretch_r_cornerlip
`public static final int stretch_r_cornerlip`

stretch_r_cornerlip = 53

stretch_r_nose
`public static final int stretch_r_nose`

stretch_r_nose = 61

thrust_jaw
`public static final int thrust_jaw`

thrust_jaw = 13

thrust_l_eyeball
`public static final int thrust_l_eyeball`

thrust_l_eyeball = 26

thrust_r_eyeball
`public static final int thrust_r_eyeball`

thrust_r_eyeball = 27

thrust_tongue_tip
`public static final int thrust_tongue_tip`

thrust_tongue_tip = 44

tongue_roll
`public static final int tongue_roll`

tongue_roll = 46

viseme
`public static final int viseme`

viseme = 0

yaw_l_eyeball
`public static final int yaw_l_eyeball`

yaw_l_eyeball = 22

yaw_r_eyeball
`public static final int yaw_r_eyeball`

yaw_r_eyeball = 23

V.5.146 org.iso.mpeg.mpegj.scene.EventOut.FDP

V.5.146.1 Syntax

```
public static interface EventOut.FDP
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000563

V.5.146.2 Description

Member Summary	
Fields	
int	featurePointsCoord
int	textureCoord
int	faceDefTables
int	faceSceneGraph

V.5.146.3 Fields

faceDefTables
`public static final int faceDefTables`

faceDefTables = 2

faceSceneGraph
`public static final int faceSceneGraph`

faceSceneGraph = 3

featurePointsCoord
`public static final int featurePointsCoord`

featurePointsCoord = 0

textureCoord
`public static final int textureCoord`

textureCoord = 1

V.5.147 org.iso.mpeg.mpegj.scene.EventOut.FIT

V.5.147.1 Syntax

```
public static interface EventOut.FIT
```

V.5.147.2 Description

Member Summary	
Fields	
int	FAPs
int	Graph
int	numeratorExp
int	denominatorExp
int	numeratorImpulse
int	numeratorTerms
int	denominatorTerms
int	numeratorCoefs
int	denominatorCoefs

V.5.147.3 Fields

denominatorCoefs
`public static final int denominatorCoefs`

denominatorCoefs = 8

denominatorExp
`public static final int denominatorExp`

denominatorExp = 3

denominatorTerms
`public static final int denominatorTerms`

denominatorTerms = 6

FAPs
`public static final int FAPs`

FAPs = 0

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000564

Graph
`public static final int Graph`

Graph = 1

numeratorCoefs
`public static final int numeratorCoefs`

numeratorCoefs = 7

numeratorExp
`public static final int numeratorExp`

numeratorExp = 2

numeratorImpulse
`public static final int numeratorImpulse`

numeratorImpulse = 4

numeratorTerms
`public static final int numeratorTerms`

numeratorTerms = 5

V.5.148 org.iso.mpeg.mpegj.scene.EventOut.Fog

V.5.148.1 Syntax

```
public static interface EventOut.Fog
```

V.5.148.2 Description

Member Summary	
Fields	
int	color
int	fogType
int	visibilityRange
int	isBound

V.5.148.3 Fields

color
`public static final int color`

color = 0

fogType
`public static final int fogType`

fogType = 1

isBound
`public static final int isBound`

isBound = 3

visibilityRange
`public static final int visibilityRange`

visibilityRange = 2

V.5.149 org.iso.mpeg.mpegj.scene.EventOut.FontStyle

V.5.149.1 Syntax

```
public static interface EventOut.FontStyle
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000565

V.5.149.2 Description

V.5.150 org.iso.mpeg.mpegj.scene.EventOut.Form

V.5.150.1 Syntax

```
public static interface EventOut.Form
```

V.5.150.2 Description

Member Summary	
Fields	
int	children
int	size
int	groups
int	constraints
int	groupsIndex

V.5.150.3 Fields

children
```
public static final int children
```
children = 0

constraints
```
public static final int constraints
```
constraints = 3

groups
```
public static final int groups
```
groups = 2

groupsIndex
```
public static final int groupsIndex
```
groupsIndex = 4

size
```
public static final int size
```
size = 1

V.5.151 org.iso.mpeg.mpegj.scene.EventOut.Group

V.5.151.1 Syntax

```
public static interface EventOut.Group
```

V.5.151.2 Description

Member Summary	
Fields	
int	children

V.5.151.3 Fields

children
```
public static final int children
```
children = 0

V.5.152 org.iso.mpeg.mpegj.scene.EventOut.ImageTexture

V.5.152.1 Syntax

```
public static interface EventOut.ImageTexture
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000566

V.5.152.2 Description

Member Summary	
Fields	
int	url

V.5.152.3 Fields

url
```
public static final int url
```
url = 0

V.5.153 org.iso.mpeg.mpegj.scene.EventOut.IndexedFaceSet

V.5.153.1 Syntax

```
public static interface EventOut.IndexedFaceSet
```

V.5.153.2 Description

Member Summary	
Fields	
int	color
int	coord
int	normal
int	texCoord

V.5.153.3 Fields

color
```
public static final int color
```
color = 0

coord
```
public static final int coord
```
coord = 1

normal
```
public static final int normal
```
normal = 2

texCoord
```
public static final int texCoord
```
texCoord = 3

V.5.154 org.iso.mpeg.mpegj.scene.EventOut.IndexedFaceSet2D

V.5.154.1 Syntax

```
public static interface EventOut.IndexedFaceSet2D
```

V.5.154.2 Description

Member Summary	
Fields	
int	color
int	coord
int	texCoord

V.5.154.3 Fields

color
```
public static final int color
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000567

color = 0

coord
`public static final int coord`

coord = 1

texCoord
`public static final int texCoord`

texCoord = 2

V.5.155 org.iso.mpeg.mpegj.scene.EventOut.IndexedLineSet

V.5.155.1 Syntax

```
public static interface EventOut.IndexedLineSet
```

V.5.155.2 Description

Member Summary	
Fields	
int	color
int	coord

V.5.155.3 Fields

color
`public static final int color`

color = 0

coord
`public static final int coord`

coord = 1

V.5.156 org.iso.mpeg.mpegj.scene.EventOut.IndexedLineSet2D

V.5.156.1 Syntax

```
public static interface EventOut.IndexedLineSet2D
```

V.5.156.2 Description

Member Summary	
Fields	
int	color
int	coord

V.5.156.3 Fields

color
`public static final int color`

color = 0

coord
`public static final int coord`

coord = 1

V.5.157 org.iso.mpeg.mpegj.scene.EventOut.Inline

V.5.157.1 Syntax

```
public static interface EventOut.Inline
```

V.5.157.2 Description

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000568

Member Summary	
Fields	
int	url

V.5.157.3 Fields

url
`public static final int url`

url = 0

V.5.158 org.iso.mpeg.mpegj.scene.EventOut.Layer2D

V.5.158.1 Syntax

```
public static interface EventOut.Layer2D
```

V.5.158.2 Description

Member Summary	
Fields	
int	children
int	size
int	background
int	viewport

V.5.158.3 Fields

background
`public static final int background`

background = 2

children
`public static final int children`

children = 0

size
`public static final int size`

size = 1

viewport
`public static final int viewport`

viewport = 3

V.5.159 org.iso.mpeg.mpegj.scene.EventOut.Layer3D

V.5.159.1 Syntax

```
public static interface EventOut.Layer3D
```

V.5.159.2 Description

Member Summary	
Fields	
int	children
int	size
int	background
int	fog
int	navigationInfo
int	viewpoint

V.5.159.3 Fields

background

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000569

```
public static final int background
```

background = 2

children
```
public static final int children
```

children = 0

fog
```
public static final int fog
```

fog = 3

navigationInfo
```
public static final int navigationInfo
```

navigationInfo = 4

size
```
public static final int size
```

size = 1

viewpoint
```
public static final int viewpoint
```

viewpoint = 5

V.5.160 org.iso.mpeg.mpegj.scene.EventOut.Layout

V.5.160.1 Syntax

```
public static interface EventOut.Layout
```

V.5.160.2 Description

Member Summary	
Fields	
int	children
int	wrap
int	size
int	horizontal
int	justify
int	leftToRight
int	topToBottom
int	spacing
int	smoothScroll
int	loop
int	scrollVertical
int	scrollRate

V.5.160.3 Fields

children
```
public static final int children
```

children = 0

horizontal
```
public static final int horizontal
```

horizontal = 3

justify
```
public static final int justify
```

justify = 4

leftToRight
```
public static final int leftToRight
```

leftToRight = 5

loop
```
public static final int loop
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000570

loop = 9

scrollRate
`public static final int scrollRate`

scrollRate = 11

scrollVertical
`public static final int scrollVertical`

scrollVertical = 10

size
`public static final int size`

size = 2

smoothScroll
`public static final int smoothScroll`

smoothScroll = 8

spacing
`public static final int spacing`

spacing = 7

topToBottom
`public static final int topToBottom`

topToBottom = 6

wrap
`public static final int wrap`

wrap = 1

V.5.161 org.iso.mpeg.mpegj.scene.EventOut.LineProperties

V.5.161.1 Syntax

```
public static interface EventOut.LineProperties
```

V.5.161.2 Description

Member Summary	
Fields	
int	lineColor
int	lineStyle
int	width

V.5.161.3 Fields

lineColor
`public static final int lineColor`

lineColor = 0

lineStyle
`public static final int lineStyle`

lineStyle = 1

width
`public static final int width`

width = 2

V.5.162 org.iso.mpeg.mpegj.scene.EventOut.ListeningPoint

V.5.162.1 Syntax

```
public static interface EventOut.ListeningPoint
```

V.5.162.2 Description

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000571

Member Summary	
Fields	
int	jump
int	orientation
int	position
int	bindTime
int	isBound

V.5.162.3 Fields

bindTime
`public static final int bindTime`

bindTime = 3

isBound
`public static final int isBound`

isBound = 4

jump
`public static final int jump`

jump = 0

orientation
`public static final int orientation`

orientation = 1

position
`public static final int position`

position = 2

V.5.163 org.iso.mpeg.mpegj.scene.EventOut.LOD

V.5.163.1 Syntax

```
public static interface EventOut.LOD
```

V.5.163.2 Description

Member Summary	
Fields	
int	level

V.5.163.3 Fields

level
`public static final int level`

level = 0

V.5.164 org.iso.mpeg.mpegj.scene.EventOut.Material

V.5.164.1 Syntax

```
public static interface EventOut.Material
```

V.5.164.2 Description

Member Summary	
Fields	
int	ambientIntensity
int	diffuseColor
int	emissiveColor
int	shininess
int	specularColor
int	transparency

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000572

V.5.164.3 Fields

ambientIntensity
`public static final int ambientIntensity`

ambientIntensity = 0

diffuseColor
`public static final int diffuseColor`

diffuseColor = 1

emissiveColor
`public static final int emissiveColor`

emissiveColor = 2

shininess
`public static final int shininess`

shininess = 3

specularColor
`public static final int specularColor`

specularColor = 4

transparency
`public static final int transparency`

transparency = 5

V.5.165 org.iso.mpeg.mpegj.scene.EventOut.Material2D

V.5.165.1 Syntax

```
public static interface EventOut.Material2D
```

V.5.165.2 Description

Member Summary	
Fields	
int	emissiveColor
int	filled
int	lineProps
int	transparency

V.5.165.3 Fields

emissiveColor
`public static final int emissiveColor`

emissiveColor = 0

filled
`public static final int filled`

filled = 1

lineProps
`public static final int lineProps`

lineProps = 2

transparency
`public static final int transparency`

transparency = 3

V.5.166 org.iso.mpeg.mpegj.scene.EventOut.MovieTexture

V.5.166.1 Syntax

```
public static interface EventOut.MovieTexture
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000573

V.5.166.2 Description

Member Summary	
Fields	
int	loop
int	speed
int	startTime
int	stopTime
int	url
int	duration_changed
int	isActive

V.5.166.3 Fields

duration_changed
`public static final int duration_changed`

duration_changed = 5

isActive
`public static final int isActive`

isActive = 6

loop
`public static final int loop`

loop = 0

speed
`public static final int speed`

speed = 1

startTime
`public static final int startTime`

startTime = 2

stopTime
`public static final int stopTime`

stopTime = 3

url
`public static final int url`

url = 4

V.5.167 org.iso.mpeg.mpegj.scene.EventOut.NavigationInfo

V.5.167.1 Syntax

`public static interface EventOut.NavigationInfo`

V.5.167.2 Description

Member Summary	
Fields	
int	avatarSize
int	headlight
int	speed
int	type
int	visibilityLimit
int	isBound

V.5.167.3 Fields

avatarSize
`public static final int avatarSize`

avatarSize = 0

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000574

headlight
`public static final int headlight`

headlight = 1

isBound
`public static final int isBound`

isBound = 5

speed
`public static final int speed`

speed = 2

type
`public static final int type`

type = 3

visibilityLimit
`public static final int visibilityLimit`

visibilityLimit = 4

V.5.168 org.iso.mpeg.mpegj.scene.EventOut.Normal

V.5.168.1 Syntax

```
public static interface EventOut.Normal
```

V.5.168.2 Description

Member Summary	
Fields	
int	vector

V.5.168.3 Fields

vector
`public static final int vector`

vector = 0

V.5.169 org.iso.mpeg.mpegj.scene.EventOut.NormalInterpolator

V.5.169.1 Syntax

```
public static interface EventOut.NormalInterpolator
```

V.5.169.2 Description

Member Summary	
Fields	
int	key
int	keyValue
int	value_changed

V.5.169.3 Fields

key
`public static final int key`

key = 0

keyValue
`public static final int keyValue`

keyValue = 1

value_changed
`public static final int value_changed`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000575

value_changed = 2

V.5.170 org.iso.mpeg.mpegj.scene.EventOut.OrderedGroup

V.5.170.1 Syntax

```
public static interface EventOut.OrderedGroup
```

V.5.170.2 Description

Member Summary	
Fields	
int	children
int	order

V.5.170.3 Fields

children
```
public static final int children
```
children = 0

order
```
public static final int order
```
order = 1

V.5.171 org.iso.mpeg.mpegj.scene.EventOut.OrientationInterpolator

V.5.171.1 Syntax

```
public static interface EventOut.OrientationInterpolator
```

V.5.171.2 Description

Member Summary	
Fields	
int	key
int	keyValue
int	value_changed

V.5.171.3 Fields

key
```
public static final int key
```
key = 0

keyValue
```
public static final int keyValue
```
keyValue = 1

value_changed
```
public static final int value_changed
```
value_changed = 2

V.5.172 org.iso.mpeg.mpegj.scene.EventOut.PixelTexture

V.5.172.1 Syntax

```
public static interface EventOut.PixelTexture
```

V.5.172.2 Description

Member Summary	
Fields	

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000576

int	image

V.5.172.3 Fields

image
`public static final int image`

image = 0

V.5.173 org.iso.mpeg.mpegj.scene.EventOut.PlaneSensor

V.5.173.1 Syntax

```
public static interface EventOut.PlaneSensor
```

V.5.173.2 Description

Member Summary	
Fields	
int	autoOffset
int	enabled
int	maxPosition
int	minPosition
int	offset
int	isActive
int	trackPoint_changed
int	translation_changed

V.5.173.3 Fields

autoOffset
`public static final int autoOffset`

autoOffset = 0

enabled
`public static final int enabled`

enabled = 1

isActive
`public static final int isActive`

isActive = 5

maxPosition
`public static final int maxPosition`

maxPosition = 2

minPosition
`public static final int minPosition`

minPosition = 3

offset
`public static final int offset`

offset = 4

trackPoint_changed
`public static final int trackPoint_changed`

trackPoint_changed = 6

translation_changed
`public static final int translation_changed`

translation_changed = 7

V.5.174 org.iso.mpeg.mpegj.scene.EventOut.PlaneSensor2D

V.5.174.1 Syntax

```
public static interface EventOut.PlaneSensor2D
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000577

V.5.174.2 Description

Member Summary	
Fields	
int	autoOffset
int	enabled
int	maxPosition
int	minPosition
int	offset
int	isActive
int	trackPoint_changed
int	translation_changed

V.5.174.3 Fields

autoOffset
`public static final int autoOffset`

autoOffset = 0

enabled
`public static final int enabled`

enabled = 1

isActive
`public static final int isActive`

isActive = 5

maxPosition
`public static final int maxPosition`

maxPosition = 2

minPosition
`public static final int minPosition`

minPosition = 3

offset
`public static final int offset`

offset = 4

trackPoint_changed
`public static final int trackPoint_changed`

trackPoint_changed = 6

translation_changed
`public static final int translation_changed`

translation_changed = 7

V.5.175 org.iso.mpeg.mpegj.scene.EventOut.PointLight

V.5.175.1 Syntax

```
public static interface EventOut.PointLight
```

V.5.175.2 Description

Member Summary	
Fields	
int	ambientIntensity
int	attenuation
int	color
int	intensity
int	location
int	on
int	radius

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000578

V.5.175.3 Fields

ambientIntensity
`public static final int ambientIntensity`

ambientIntensity = 0

attenuation
`public static final int attenuation`

attenuation = 1

color
`public static final int color`

color = 2

intensity
`public static final int intensity`

intensity = 3

location
`public static final int location`

location = 4

on
`public static final int on`

on = 5

radius
`public static final int radius`

radius = 6

V.5.176 org.iso.mpeg.mpegj.scene.EventOut.PointSet

V.5.176.1 Syntax

```
public static interface EventOut.PointSet
```

V.5.176.2 Description

Member Summary	
Fields	
int	color
int	coord

V.5.176.3 Fields

color
`public static final int color`

color = 0

coord
`public static final int coord`

coord = 1

V.5.177 org.iso.mpeg.mpegj.scene.EventOut.PointSet2D

V.5.177.1 Syntax

```
public static interface EventOut.PointSet2D
```

V.5.177.2 Description

Member Summary	
Fields	
int	color
int	coord

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000579

V.5.177.3 Fields

color
`public static final int color`

color = 0

coord
`public static final int coord`

coord = 1

V.5.178 org.iso.mpeg.mpegj.scene.EventOut.PositionInterpolator

V.5.178.1 Syntax

```
public static interface EventOut.PositionInterpolator
```

V.5.178.2 Description

Member Summary	
Fields	
int	key
int	keyValue
int	value_changed

V.5.178.3 Fields

key
`public static final int key`

key = 0

keyValue
`public static final int keyValue`

keyValue = 1

value_changed
`public static final int value_changed`

value_changed = 2

V.5.179 org.iso.mpeg.mpegj.scene.EventOut.PositionInterpolator2D

V.5.179.1 Syntax

```
public static interface EventOut.PositionInterpolator2D
```

V.5.179.2 Description

Member Summary	
Fields	
int	key
int	keyValue
int	value_changed

V.5.179.3 Fields

key
`public static final int key`

key = 0

keyValue
`public static final int keyValue`

keyValue = 1

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000580

value_changed
`public static final int value_changed`

value_changed = 2

V.5.180 org.iso.mpeg.mpegj.scene.EventOut.ProximitySensor

V.5.180.1 Syntax

```
public static interface EventOut.ProximitySensor
```

V.5.180.2 Description

Member Summary	
Fields	
int	center
int	size
int	enabled
int	isActive
int	position_changed
int	orientation_changed
int	enterTime
int	exitTime

V.5.180.3 Fields

center
`public static final int center`

center = 0

enabled
`public static final int enabled`

enabled = 2

enterTime
`public static final int enterTime`

enterTime = 6

exitTime
`public static final int exitTime`

exitTime = 7

isActive
`public static final int isActive`

isActive = 3

orientation_changed
`public static final int orientation_changed`

orientation_changed = 5

position_changed
`public static final int position_changed`

position_changed = 4

size
`public static final int size`

size = 1

V.5.181 org.iso.mpeg.mpegj.scene.EventOut.ProximitySensor2D

V.5.181.1 Syntax

```
public static interface EventOut.ProximitySensor2D
```

V.5.181.2 Description

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000581

Member Summary	
Fields	
int	center
int	size
int	enabled
int	isActive
int	position_changed
int	orientation_changed
int	enterTime
int	exitTime

V.5.181.3 Fields

center
```
public static final int center
```
center = 0

enabled
```
public static final int enabled
```
enabled = 2

enterTime
```
public static final int enterTime
```
enterTime = 6

exitTime
```
public static final int exitTime
```
exitTime = 7

isActive
```
public static final int isActive
```
isActive = 3

orientation_changed
```
public static final int orientation_changed
```
orientation_changed = 5

position_changed
```
public static final int position_changed
```
position_changed = 4

size
```
public static final int size
```
size = 1

V.5.182 org.iso.mpeg.mpegj.scene.EventOut.QuantizationParameter

V.5.182.1 Syntax

```
public static interface EventOut.QuantizationParameter
```

V.5.182.2 Description

V.5.183 org.iso.mpeg.mpegj.scene.EventOut.Rectangle

V.5.183.1 Syntax

```
public static interface EventOut.Rectangle
```

V.5.183.2 Description

Member Summary	
Fields	
int	size

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000582

V.5.183.3 Fields

size
`public static final int size`

size = 0

V.5.184 org.iso.mpeg.mpegj.scene.EventOut.ScalarInterpolator

V.5.184.1 Syntax

```
public static interface EventOut.ScalarInterpolator
```

V.5.184.2 Description

Member Summary	
Fields	
int	key
int	keyValue
int	value_changed

V.5.184.3 Fields

key
`public static final int key`

key = 0

keyValue
`public static final int keyValue`

keyValue = 1

value_changed
`public static final int value_changed`

value_changed = 2

V.5.185 org.iso.mpeg.mpegj.scene.EventOut.Script

V.5.185.1 Syntax

```
public static interface EventOut.Script
```

V.5.185.2 Description

Member Summary	
Fields	
int	url

V.5.185.3 Fields

url
`public static final int url`

url = 0

V.5.186 org.iso.mpeg.mpegj.scene.EventOut.Shape

V.5.186.1 Syntax

```
public static interface EventOut.Shape
```

V.5.186.2 Description

Member Summary	
Fields	
int	appearance

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000583

int	geometry

V.5.186.3 Fields

appearance
`public static final int appearance`

appearance = 0

geometry
`public static final int geometry`

geometry = 1

V.5.187 org.iso.mpeg.mpegj.scene.EventOut.Sound

V.5.187.1 Syntax

```
public static interface EventOut.Sound
```

V.5.187.2 Description

Member Summary	
Fields	
int	direction
int	intensity
int	location
int	maxBack
int	maxFront
int	minBack
int	minFront
int	priority
int	source

V.5.187.3 Fields

direction
`public static final int direction`

direction = 0

intensity
`public static final int intensity`

intensity = 1

location
`public static final int location`

location = 2

maxBack
`public static final int maxBack`

maxBack = 3

maxFront
`public static final int maxFront`

maxFront = 4

minBack
`public static final int minBack`

minBack = 5

minFront
`public static final int minFront`

minFront = 6

priority
`public static final int priority`

priority = 7

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000584

source
`public static final int source`

source = 8

V.5.188 org.iso.mpeg.mpegj.scene.EventOut.Sound2D

V.5.188.1 Syntax

```
public static interface EventOut.Sound2D
```

V.5.188.2 Description

Member Summary	
Fields	
int	intensity
int	location
int	source

V.5.188.3 Fields

intensity
`public static final int intensity`

intensity = 0

location
`public static final int location`

location = 1

source
`public static final int source`

source = 2

V.5.189 org.iso.mpeg.mpegj.scene.EventOut.Sphere

V.5.189.1 Syntax

```
public static interface EventOut.Sphere
```

V.5.189.2 Description

V.5.190 org.iso.mpeg.mpegj.scene.EventOut.SphereSensor

V.5.190.1 Syntax

```
public static interface EventOut.SphereSensor
```

V.5.190.2 Description

Member Summary	
Fields	
int	autoOffset
int	enabled
int	offset
int	isActive
int	rotation_changed
int	trackPoint_changed

V.5.190.3 Fields

autoOffset
`public static final int autoOffset`

autoOffset = 0

enabled

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000585

```
public static final int enabled
```

enabled = 1

isActive
```
public static final int isActive
```

isActive = 3

offset
```
public static final int offset
```

offset = 2

rotation_changed
```
public static final int rotation_changed
```

rotation_changed = 4

trackPoint_changed
```
public static final int trackPoint_changed
```

trackPoint_changed = 5

V.5.191 org.iso.mpeg.mpegj.scene.EventOut.SpotLight

V.5.191.1 Syntax

```
public static interface EventOut.SpotLight
```

V.5.191.2 Description

Member Summary	
Fields	
int	ambientIntensity
int	attenuation
int	beamWidth
int	color
int	cutOffAngle
int	direction
int	intensity
int	location
int	on
int	radius

V.5.191.3 Fields

ambientIntensity
```
public static final int ambientIntensity
```

ambientIntensity = 0

attenuation
```
public static final int attenuation
```

attenuation = 1

beamWidth
```
public static final int beamWidth
```

beamWidth = 2

color
```
public static final int color
```

color = 3

cutOffAngle
```
public static final int cutOffAngle
```

cutOffAngle = 4

direction
```
public static final int direction
```

direction = 5

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000586

intensity
`public static final int intensity`

intensity = 6

location
`public static final int location`

location = 7

on
`public static final int on`

on = 8

radius
`public static final int radius`

radius = 9

V.5.192 org.iso.mpeg.mpegj.scene.EventOut.Switch

V.5.192.1 Syntax

```
public static interface EventOut.Switch
```

V.5.192.2 Description

Member Summary	
Fields	
int	choice
int	whichChoice

V.5.192.3 Fields

choice
`public static final int choice`

choice = 0

whichChoice
`public static final int whichChoice`

whichChoice = 1

V.5.193 org.iso.mpeg.mpegj.scene.EventOut.TermCap

V.5.193.1 Syntax

```
public static interface EventOut.TermCap
```

V.5.193.2 Description

Member Summary	
Fields	
int	capability
int	value

V.5.193.3 Fields

capability
`public static final int capability`

capability = 0

value
`public static final int value`

value = 1

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000587

V.5.194 org.iso.mpeg.mpegj.scene.EventOut.Text

V.5.194.1 Syntax

```
public static interface EventOut.Text
```

V.5.194.2 Description

Member Summary	
Fields	
int	string
int	length
int	fontStyle
int	maxExtent

V.5.194.3 Fields

fontStyle
```
public static final int fontStyle
```
fontStyle = 2

length
```
public static final int length
```
length = 1

maxExtent
```
public static final int maxExtent
```
maxExtent = 3

string
```
public static final int string
```
string = 0

V.5.195 org.iso.mpeg.mpegj.scene.EventOut.TextureCoordinate

V.5.195.1 Syntax

```
public static interface EventOut.TextureCoordinate
```

V.5.195.2 Description

Member Summary	
Fields	
int	point

V.5.195.3 Fields

point
```
public static final int point
```
point = 0

V.5.196 org.iso.mpeg.mpegj.scene.EventOut.TextureTransform

V.5.196.1 Syntax

```
public static interface EventOut.TextureTransform
```

V.5.196.2 Description

Member Summary	
Fields	
int	center
int	rotation
int	scale

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000588

int	translation

V.5.196.3 Fields

center
`public static final int center`

center = 0

rotation
`public static final int rotation`

rotation = 1

scale
`public static final int scale`

scale = 2

translation
`public static final int translation`

translation = 3

V.5.197 org.iso.mpeg.mpegj.scene.EventOut.TimeSensor

V.5.197.1 Syntax

```
public static interface EventOut.TimeSensor
```

V.5.197.2 Description

Member Summary	
Fields	
int	cycleInterval
int	enabled
int	loop
int	startTime
int	stopTime
int	cycleTime
int	fraction_changed
int	isActive
int	time

V.5.197.3 Fields

cycleInterval
`public static final int cycleInterval`

cycleInterval = 0

cycleTime
`public static final int cycleTime`

cycleTime = 5

enabled
`public static final int enabled`

enabled = 1

fraction_changed
`public static final int fraction_changed`

fraction_changed = 6

isActive
`public static final int isActive`

isActive = 7

loop
`public static final int loop`

loop = 2

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000589

startTime
`public static final int startTime`

startTime = 3

stopTime
`public static final int stopTime`

stopTime = 4

time
`public static final int time`

time = 8

V.5.198 org.iso.mpeg.mpegj.scene.EventOut.TouchSensor

V.5.198.1 Syntax

```
public static interface EventOut.TouchSensor
```

V.5.198.2 Description

Member Summary	
Fields	
int	enabled
int	hitNormal_changed
int	hitPoint_changed
int	hitTexCoord_changed
int	isActive
int	isOver
int	touchTime

V.5.198.3 Fields

enabled
`public static final int enabled`

enabled = 0

hitNormal_changed
`public static final int hitNormal_changed`

hitNormal_changed = 1

hitPoint_changed
`public static final int hitPoint_changed`

hitPoint_changed = 2

hitTexCoord_changed
`public static final int hitTexCoord_changed`

hitTexCoord_changed = 3

isActive
`public static final int isActive`

isActive = 4

isOver
`public static final int isOver`

isOver = 5

touchTime
`public static final int touchTime`

touchTime = 6

V.5.199 org.iso.mpeg.mpegj.scene.EventOut.Transform

V.5.199.1 Syntax

```
public static interface EventOut.Transform
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000590

V.5.199.2 Description

Member Summary	
Fields	
int	center
int	children
int	rotation
int	scale
int	scaleOrientation
int	translation

V.5.199.3 Fields

center
`public static final int center`

center = 0

children
`public static final int children`

children = 1

rotation
`public static final int rotation`

rotation = 2

scale
`public static final int scale`

scale = 3

scaleOrientation
`public static final int scaleOrientation`

scaleOrientation = 4

translation
`public static final int translation`

translation = 5

V.5.200 org.iso.mpeg.mpegj.scene.EventOut.Transform2D

V.5.200.1 Syntax

```
public static interface EventOut.Transform2D
```

V.5.200.2 Description

Member Summary	
Fields	
int	children
int	center
int	rotationAngle
int	scale
int	scaleOrientation
int	translation

V.5.200.3 Fields

center
`public static final int center`

center = 1

children
`public static final int children`

children = 0

rotationAngle

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000591

```
public static final int rotationAngle
```

rotationAngle = 2

scale
```
public static final int scale
```

scale = 3

scaleOrientation
```
public static final int scaleOrientation
```

scaleOrientation = 4

translation
```
public static final int translation
```

translation = 5

V.5.201 org.iso.mpeg.mpegj.scene.EventOut.Valuator

V.5.201.1 Syntax

```
public static interface EventOut.Valuator
```

V.5.201.2 Description

Member Summary	
Fields	
int	outSFBool
int	outSFColor
int	outMFColor
int	outSFFloat
int	outMFFloat
int	outSFInt32
int	outMFInt32
int	outSFRotation
int	outMFRotation
int	outSFString
int	outMFString
int	outSFTime
int	outSFVec2f
int	outMFVec2f
int	outSFVec3f
int	outMFVec3f
int	Factor1
int	Factor2
int	Factor3
int	Factor4
int	Offset1
int	Offset2
int	Offset3
int	Offset4
int	Sum

V.5.201.3 Fields

Factor1
```
public static final int Factor1
```

Factor1 = 16

Factor2
```
public static final int Factor2
```

Factor2 = 17

Factor3
```
public static final int Factor3
```

Factor3 = 18

Factor4
```
public static final int Factor4
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000592

Factor4 = 19

Offset1
`public static final int Offset1`

Offset1 = 20

Offset2
`public static final int Offset2`

Offset2 = 21

Offset3
`public static final int Offset3`

Offset3 = 22

Offset4
`public static final int Offset4`

Offset4 = 23

outMFColor
`public static final int outMFColor`

outMFColor = 2

outMFFloat
`public static final int outMFFloat`

outMFFloat = 4

outMFInt32
`public static final int outMFInt32`

outMFInt32 = 6

outMFRotation
`public static final int outMFRotation`

outMFRotation = 8

outMFString
`public static final int outMFString`

outMFString = 10

outMFVec2f
`public static final int outMFVec2f`

outMFVec2f = 13

outMFVec3f
`public static final int outMFVec3f`

outMFVec3f = 15

outSFBool
`public static final int outSFBool`

outSFBool = 0

outSFColor
`public static final int outSFColor`

outSFColor = 1

outSFFloat
`public static final int outSFFloat`

outSFFloat = 3

outSFInt32
`public static final int outSFInt32`

outSFInt32 = 5

outSFRotation
`public static final int outSFRotation`

outSFRotation = 7

outSFString
`public static final int outSFString`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000593

outSFString = 9

outSFTime
`public static final int outSFTime`

outSFTime = 11

outSFVec2f
`public static final int outSFVec2f`

outSFVec2f = 12

outSFVec3f
`public static final int outSFVec3f`

outSFVec3f = 14

Sum
`public static final int Sum`

Sum = 24

V.5.202 org.iso.mpeg.mpegj.scene.EventOut.Viewpoint

V.5.202.1 Syntax

```
public static interface EventOut.Viewpoint
```

V.5.202.2 Description

Member Summary	
Fields	
int	fieldOfView
int	jump
int	orientation
int	position
int	bindTime
int	isBound

V.5.202.3 Fields

bindTime
`public static final int bindTime`

bindTime = 4

fieldOfView
`public static final int fieldOfView`

fieldOfView = 0

isBound
`public static final int isBound`

isBound = 5

jump
`public static final int jump`

jump = 1

orientation
`public static final int orientation`

orientation = 2

position
`public static final int position`

position = 3

V.5.203 org.iso.mpeg.mpegj.scene.EventOut.Viseme

V.5.203.1 Syntax

```
public static interface EventOut.Viseme
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000594

V.5.203.2 Description

Member Summary	
Fields	
int	viseme_select1
int	viseme_select2
int	viseme_blend
int	viseme_def

V.5.203.3 Fields

viseme_blend
```
public static final int viseme_blend
```
viseme_blend = 2

viseme_def
```
public static final int viseme_def
```
viseme_def = 3

viseme_select1
```
public static final int viseme_select1
```
viseme_select1 = 0

viseme_select2
```
public static final int viseme_select2
```
viseme_select2 = 1

V.5.204 org.iso.mpeg.mpegj.scene.EventOut.VisibilitySensor

V.5.204.1 Syntax

```
public static interface EventOut.VisibilitySensor
```

V.5.204.2 Description

Member Summary	
Fields	
int	center
int	enabled
int	size
int	enterTime
int	exitTime
int	isActive

V.5.204.3 Fields

center
```
public static final int center
```
center = 0

enabled
```
public static final int enabled
```
enabled = 1

enterTime
```
public static final int enterTime
```
enterTime = 3

exitTime
```
public static final int exitTime
```
exitTime = 4

isActive
```
public static final int isActive
```
isActive = 5

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000595

size
`public static final int size`

size = 2

V.5.205 org.iso.mpeg.mpegj.scene.EventOut.WorldInfo

V.5.205.1 Syntax

`public static interface EventOut.WorldInfo`

V.5.205.2 Description

V.5.206 org.iso.mpeg.mpegj.scene.EventOutListener

V.5.206.1 Syntax

`public interface EventOutListener`

V.5.206.2 Description

An interface to be implemented by objects that wish to receive eventOut notification from fields in the scene.

Member Summary	
Methods	
void	notify(int, FieldValue)

V.5.206.3 Methods

notify(int, FieldValue)
`public void notify(int, org.iso.mpeg.mpegj.scene.FieldValue)`

Called by the scene graph manager when a new event out is triggered.

Parameters:
`outID` - contains the outID of the changed field.

`newValue` - contains the new value of the field.

See Also:
`org.iso.mpeg.mpegj.scene.EventOut`

V.5.207 org.iso.mpeg.mpegj.scene.Field

V.5.207.1 Syntax

`public interface Field`

V.5.207.2 Description

An interface with inner classes defining the constants for the field defIDs of each node.

Member Summary	
Inner Classes	
static interface	Field.Anchor
static interface	Field.AnimationStream
static interface	Field.Appearance
static interface	Field.AudioBuffer
static interface	Field.AudioClip
static interface	Field.AudioDelay
static interface	Field.AudioFX
static interface	Field.AudioMix
static interface	Field.AudioSource
static interface	Field.AudioSwitch
static interface	Field.Background
static interface	Field.Background2D
static interface	Field.Billboard
static interface	Field.Bitmap

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000596

static interface	Field.Box
static interface	Field.Circle
static interface	Field.Collision
static interface	Field.Color
static interface	Field.ColorInterpolator
static interface	Field.CompositeTexture2D
static interface	Field.CompositeTexture3D
static interface	Field.Conditional
static interface	Field.Cone
static interface	Field.Coordinate
static interface	Field.Coordinate2D
static interface	Field.CoordinateInterpolator
static interface	Field.CoordinateInterpolator2D
static interface	Field.Curve2D
static interface	Field.Cylinder
static interface	Field.CylinderSensor
static interface	Field.DirectionalLight
static interface	Field.DiscSensor
static interface	Field.ElevationGrid
static interface	Field.Expression
static interface	Field.Extrusion
static interface	Field.Face
static interface	Field.FaceDefMesh
static interface	Field.FaceDefTables
static interface	Field.FaceDefTransform
static interface	Field.FAP
static interface	Field.FDP
static interface	Field.FIT
static interface	Field.Fog
static interface	Field.FontStyle
static interface	Field.Form
static interface	Field.Group
static interface	Field.ImageTexture
static interface	Field.IndexedFaceSet
static interface	Field.IndexedFaceSet2D
static interface	Field.IndexedLineSet
static interface	Field.IndexedLineSet2D
static interface	Field.Inline
static interface	Field.LOD
static interface	Field.Layer2D
static interface	Field.Layer3D
static interface	Field.Layout
static interface	Field.LineProperties
static interface	Field.ListeningPoint
static interface	Field.Material
static interface	Field.Material2D
static interface	Field.MovieTexture
static interface	Field.NavigationInfo
static interface	Field.Normal
static interface	Field.NormalInterpolator
static interface	Field.OrderedGroup
static interface	Field.OrientationInterpolator
static interface	Field.PixelTexture
static interface	Field.PlaneSensor
static interface	Field.PlaneSensor2D
static interface	Field.PointLight
static interface	Field.PointSet
static interface	Field.PointSet2D
static interface	Field.PositionInterpolator
static interface	Field.PositionInterpolator2D
static interface	Field.ProximitySensor2D
static interface	Field.ProximitySensor
static interface	Field.QuantizationParameter
static interface	Field.Rectangle
static interface	Field.ScalarInterpolator
static interface	Field.Script
static interface	Field.Shape
static interface	Field.Sound
static interface	Field.Sound2D
static interface	Field.Sphere
static interface	Field.SphereSensor

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000597

static interface	Field.SpotLight
static interface	Field.Switch
static interface	Field.TermCap
static interface	Field.Text
static interface	Field.TextureCoordinate
static interface	Field.TextureTransform
static interface	Field.TimeSensor
static interface	Field.TouchSensor
static interface	Field.Transform
static interface	Field.Transform2D
static interface	Field.Valuator
static interface	Field.WorldInfo

V.5.208 org.iso.mpeg.mpegj.scene.Field.Anchor

V.5.208.1 Syntax

```
public static interface Field.Anchor
```

V.5.208.2 Description

An interface defining constants for the field defIDs of the Anchor node.

Member Summary	
Fields	
int	children
int	description
int	parameter
int	url

V.5.208.3 Fields

children
```
public static final int children
```
The defID for the children field = 2.

description
```
public static final int description
```
The defID for the description field = 3.

parameter
```
public static final int parameter
```
The defID for the parameter field = 4.

url
```
public static final int url
```
The defID for the url field = 5.

V.5.209 org.iso.mpeg.mpegj.scene.Field.AnimationStream

V.5.209.1 Syntax

```
public static interface Field.AnimationStream
```

V.5.209.2 Description

An interface defining constants for the field defIDs of the AnimationStream node.

Member Summary	
Fields	
int	loop
int	speed
int	startTime
int	stopTime
int	url

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000598

V.5.209.3 Fields

loop
`public static final int loop`

The defID for the loop field = 0.

speed
`public static final int speed`

The defID for the speed field = 1.

startTime
`public static final int startTime`

The defID for the startTime field = 2.

stopTime
`public static final int stopTime`

The defID for the stopTime field = 3.

url
`public static final int url`

The defID for the url field = 4.

V.5.210 org.iso.mpeg.mpegj.scene.Field.Appearance

V.5.210.1 Syntax

```
public static interface Field.Appearance
```

V.5.210.2 Description

An interface defining constants for the field defIDs of the Appearance node.

Member Summary	
Fields	
int	material
int	texture
int	textureTransform

V.5.210.3 Fields

material
`public static final int material`

The defID for the material field = 0.

texture
`public static final int texture`

The defID for the texture field = 1.

textureTransform
`public static final int textureTransform`

The defID for the textureTransform field = 2.

V.5.211 org.iso.mpeg.mpegj.scene.Field.AudioBuffer

V.5.211.1 Syntax

```
public static interface Field.AudioBuffer
```

V.5.211.2 Description

An interface defining constants for the field defIDs of the AudioBuffer node.

Member Summary	
Fields	
int	loop
int	pitch

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000599

int	startTime
int	stopTime
int	children
int	numChan
int	phaseGroup
int	length

V.5.211.3 Fields

children
`public static final int children`

The defID for the children field = 4.

length
`public static final int length`

The defID for the length field = 7.

loop
`public static final int loop`

The defID for the loop field = 0.

numChan
`public static final int numChan`

The defID for the numChan field = 5.

phaseGroup
`public static final int phaseGroup`

The defID for the phaseGroup field = 6.

pitch
`public static final int pitch`

The defID for the pitch field = 1.

startTime
`public static final int startTime`

The defID for the startTime field = 2.

stopTime
`public static final int stopTime`

The defID for the stopTime field = 3.

V.5.212 org.iso.mpeg.mpegj.scene.Field.AudioClip

V.5.212.1 Syntax

```
public static interface Field.AudioClip
```

V.5.212.2 Description

An interface defining constants for the field defIDs of the AudioClip node.

Member Summary	
Fields	
int	description
int	loop
int	pitch
int	startTime
int	stopTime
int	url

V.5.212.3 Fields

description
`public static final int description`

The defID for the description field = 0.

loop

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000600

```
public static final int loop
```

The defID for the loop field = 1.

pitch

```
public static final int pitch
```

The defID for the pitch field = 2.

startTime

```
public static final int startTime
```

The defID for the startTime field = 3.

stopTime

```
public static final int stopTime
```

The defID for the stopTime field = 4.

url

```
public static final int url
```

The defID for the url field = 5.

V.5.213 org.iso.mpeg.mpegj.scene.Field.AudioDelay

V.5.213.1 Syntax

```
public static interface Field.AudioDelay
```

V.5.213.2 Description

An interface defining constants for the field defIDs of the AudioDelay node.

Member Summary	
Fields	
int	children
int	delay
int	numChan
int	phaseGroup

V.5.213.3 Fields

children

```
public static final int children
```

The defID for the children field = 2.

delay

```
public static final int delay
```

The defID for the delay field = 3.

numChan

```
public static final int numChan
```

The defID for the numChan field = 4.

phaseGroup

```
public static final int phaseGroup
```

The defID for the phaseGroup field = 5.

V.5.214 org.iso.mpeg.mpegj.scene.Field.AudioFX

V.5.214.1 Syntax

```
public static interface Field.AudioFX
```

V.5.214.2 Description

An interface defining constants for the field defIDs of the AudioFX node.

Member Summary	

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000601

Fields	
int	children
int	orch
int	score
int	params
int	numChan
int	phaseGroup

V.5.214.3 Fields

children
`public static final int children`

The defID for the children field = 2.

numChan
`public static final int numChan`

The defID for the numChan field = 6.

orch
`public static final int orch`

The defID for the orch field = 3.

params
`public static final int params`

The defID for the params field = 5.

phaseGroup
`public static final int phaseGroup`

The defID for the phaseGroup field = 7.

score
`public static final int score`

The defID for the score field = 4.

V.5.215 org.iso.mpeg.mpegj.scene.Field.AudioMix

V.5.215.1 Syntax

```
public static interface Field.AudioMix
```

V.5.215.2 Description

An interface defining constants for the field defIDs of the AudioMix node.

Member Summary	
Fields	
int	children
int	numInputs
int	matrix
int	numChan
int	phaseGroup

V.5.215.3 Fields

children
`public static final int children`

The defID for the children field = 2.

matrix
`public static final int matrix`

The defID for the matrix field = 4.

numChan
`public static final int numChan`

The defID for the numChan field = 5.

numInputs

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000602

```
public static final int numInputs
```

The defID for the numInputs field = 3.

phaseGroup
```
public static final int phaseGroup
```

The defID for the phaseGroup field = 6.

V.5.216 org.iso.mpeg.mpegj.scene.Field.AudioSource

V.5.216.1 Syntax

```
public static interface Field.AudioSource
```

V.5.216.2 Description

An interface defining constants for the field defIDs of the AudioSource node.

Member Summary	
Fields	
int	children
int	url
int	pitch
int	speed
int	startTime
int	stopTime
int	numChan
int	phaseGroup

V.5.216.3 Fields

children
```
public static final int children
```

The defID for the children field = 2.

numChan
```
public static final int numChan
```

The defID for the numChan field = 8.

phaseGroup
```
public static final int phaseGroup
```

The defID for the phaseGroup field = 9.

pitch
```
public static final int pitch
```

The defID for the pitch field = 4.

speed
```
public static final int speed
```

The defID for the speed field = 5.

startTime
```
public static final int startTime
```

The defID for the startTime field = 6.

stopTime
```
public static final int stopTime
```

The defID for the stopTime field = 7.

url
```
public static final int url
```

The defID for the url field = 3.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000603

V.5.217 org.iso.mpeg.mpegj.scene.Field.AudioSwitch

V.5.217.1 Syntax

```
public static interface Field.AudioSwitch
```

V.5.217.2 Description

An interface defining constants for the field defIDs of the AudioSwitch node.

Member Summary	
Fields	
int	children
int	whichChoice
int	numChan
int	phaseGroup

V.5.217.3 Fields

children
```
public static final int children
```
The defID for the children field = 2.

numChan
```
public static final int numChan
```
The defID for the numChan field = 4.

phaseGroup
```
public static final int phaseGroup
```
The defID for the phaseGroup field = 5.

whichChoice
```
public static final int whichChoice
```
The defID for the whichChoice field = 3.

V.5.218 org.iso.mpeg.mpegj.scene.Field.Background

V.5.218.1 Syntax

```
public static interface Field.Background
```

V.5.218.2 Description

An interface defining constants for the field defIDs of the Background node.

Member Summary	
Fields	
int	groundAngle
int	groundColor
int	backUrl
int	bottomUrl
int	frontUrl
int	leftUrl
int	rightUrl
int	topUrl
int	skyAngle
int	skyColor

V.5.218.3 Fields

backUrl
```
public static final int backUrl
```
The defID for the backUrl field = 3.

bottomUrl
```
public static final int bottomUrl
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000604

The defID for the bottomUrl field = 4.

frontUrl
`public static final int frontUrl`

The defID for the frontUrl field = 5.

groundAngle
`public static final int groundAngle`

The defID for the groundAngle field = 1.

groundColor
`public static final int groundColor`

The defID for the groundColor field = 2.

leftUrl
`public static final int leftUrl`

The defID for the leftUrl field = 6.

rightUrl
`public static final int rightUrl`

The defID for the rightUrl field = 7.

skyAngle
`public static final int skyAngle`

The defID for the skyAngle field = 9.

skyColor
`public static final int skyColor`

The defID for the skyColor field = 10.

topUrl
`public static final int topUrl`

The defID for the topUrl field = 8.

V.5.219 org.iso.mpeg.mpegj.scene.Field.Background2D

V.5.219.1 Syntax

```
public static interface Field.Background2D
```

V.5.219.2 Description

An interface defining constants for the field defIDs of the Background2D node.

Member Summary	
Fields	
int	backColor
int	url

V.5.219.3 Fields

backColor
`public static final int backColor`

The defID for the backColor field = 1.

url
`public static final int url`

The defID for the url field = 2.

V.5.220 org.iso.mpeg.mpegj.scene.Field.Billboard

V.5.220.1 Syntax

```
public static interface Field.Billboard
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000605

V.5.220.2 Description

An interface defining constants for the field defIDs of the Billboard node.

Member Summary	
Fields	
int	children
int	axisOfRotation

V.5.220.3 Fields

axisOfRotation
`public static final int axisOfRotation`

The defID for the axisOfRotation field = 3.

children
`public static final int children`

The defID for the children field = 2.

V.5.221 org.iso.mpeg.mpegj.scene.Field.Bitmap

V.5.221.1 Syntax

```
public static interface Field.Bitmap
```

V.5.221.2 Description

An interface defining constants for the field defIDs of the Bitmap node.

Member Summary	
Fields	
int	scale

V.5.221.3 Fields

scale
`public static final int scale`

The defID for the scale field = 0.

V.5.222 org.iso.mpeg.mpegj.scene.Field.Box

V.5.222.1 Syntax

```
public static interface Field.Box
```

V.5.222.2 Description

An interface defining constants for the field defIDs of the Box node.

Member Summary	
Fields	
int	size

V.5.222.3 Fields

size
`public static final int size`

The defID for the size field = 0.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000606

V.5.223 org.iso.mpeg.mpegj.scene.Field.Circle

V.5.223.1 Syntax

```
public static interface Field.Circle
```

V.5.223.2 Description

An interface defining constants for the field defIDs of the Circle node.

Member Summary	
Fields	
int	radius

V.5.223.3 Fields

radius
`public static final int radius`

The defID for the radius field = 0.

V.5.224 org.iso.mpeg.mpegj.scene.Field.Collision

V.5.224.1 Syntax

```
public static interface Field.Collision
```

V.5.224.2 Description

An interface defining constants for the field defIDs of the Collision node.

Member Summary	
Fields	
int	children
int	collide
int	proxy

V.5.224.3 Fields

children
`public static final int children`

The defID for the children field = 2.

collide
`public static final int collide`

The defID for the collide field = 3.

proxy
`public static final int proxy`

The defID for the proxy field = 4.

V.5.225 org.iso.mpeg.mpegj.scene.Field.Color

V.5.225.1 Syntax

```
public static interface Field.Color
```

V.5.225.2 Description

An interface defining constants for the field defIDs of the Color node.

Member Summary	
Fields	
int	color

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000607

V.5.225.3 Fields

color
`public static final int color`

The defID for the color field = 0.

V.5.226 org.iso.mpeg.mpegj.scene.Field.ColorInterpolator

V.5.226.1 Syntax

```
public static interface Field.ColorInterpolator
```

V.5.226.2 Description

An interface defining constants for the field defIDs of the ColorInterpolator node.

Member Summary	
Fields	
int	key
int	keyValue

V.5.226.3 Fields

key
`public static final int key`

The defID for the key field = 1.

keyValue
`public static final int keyValue`

The defID for the keyValue field = 2.

V.5.227 org.iso.mpeg.mpegj.scene.Field.CompositeTexture2D

V.5.227.1 Syntax

```
public static interface Field.CompositeTexture2D
```

V.5.227.2 Description

An interface defining constants for the field defIDs of the CompositeTexture2D node.

Member Summary	
Fields	
int	children
int	pixelWidth
int	pixelHeight
int	background
int	viewport

V.5.227.3 Fields

background
`public static final int background`

The defID for the background field = 5.

children
`public static final int children`

The defID for the children field = 2.

pixelHeight
`public static final int pixelHeight`

The defID for the pixelHeight field = 4.

pixelWidth

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000608

```
public static final int pixelWidth
```

The defID for the pixelWidth field = 3.

viewport

```
public static final int viewport
```

The defID for the viewport field = 6.

V.5.228 org.iso.mpeg.mpegj.scene.Field.CompositeTexture3D

V.5.228.1 Syntax

```
public static interface Field.CompositeTexture3D
```

V.5.228.2 Description

An interface defining constants for the field defIDs of the CompositeTexture3D node.

Member Summary	
Fields	
int	children
int	pixelWidth
int	pixelHeight
int	background
int	fog
int	navigationInfo
int	viewpoint

V.5.228.3 Fields

background

```
public static final int background
```

The defID for the background field = 5.

children

```
public static final int children
```

The defID for the children field = 2.

fog

```
public static final int fog
```

The defID for the fog field = 6.

navigationInfo

```
public static final int navigationInfo
```

The defID for the navigationInfo field = 7.

pixelHeight

```
public static final int pixelHeight
```

The defID for the pixelHeight field = 4.

pixelWidth

```
public static final int pixelWidth
```

The defID for the pixelWidth field = 3.

viewpoint

```
public static final int viewpoint
```

The defID for the viewpoint field = 8.

V.5.229 org.iso.mpeg.mpegj.scene.Field.Conditional

V.5.229.1 Syntax

```
public static interface Field.Conditional
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000609

V.5.229.2 Description

An interface defining constants for the field defIDs of the Conditional node.

Member Summary	
Fields	
int	buffer

V.5.229.3 Fields

buffer
`public static final int buffer`

The defID for the buffer field = 2.

V.5.230 org.iso.mpeg.mpegj.scene.Field.Cone

V.5.230.1 Syntax

```
public static interface Field.Cone
```

V.5.230.2 Description

An interface defining constants for the field defIDs of the Cone node.

Member Summary	
Fields	
int	bottomRadius
int	height
int	side
int	bottom

V.5.230.3 Fields

bottom
`public static final int bottom`

The defID for the bottom field = 3.

bottomRadius
`public static final int bottomRadius`

The defID for the bottomRadius field = 0.

height
`public static final int height`

The defID for the height field = 1.

side
`public static final int side`

The defID for the side field = 2.

V.5.231 org.iso.mpeg.mpegj.scene.Field.Coordinate

V.5.231.1 Syntax

```
public static interface Field.Coordinate
```

V.5.231.2 Description

An interface defining constants for the field defIDs of the Coordinate node.

Member Summary	
Fields	
int	point

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000610

V.5.231.3 Fields

point
`public static final int point`

The defID for the point field = 0.

V.5.232 org.iso.mpeg.mpegj.scene.Field.Coordinate2D

V.5.232.1 Syntax

```
public static interface Field.Coordinate2D
```

V.5.232.2 Description

An interface defining constants for the field defIDs of the Coordinate2D node.

Member Summary	
Fields	
int	point

V.5.232.3 Fields

point
`public static final int point`

The defID for the point field = 0.

V.5.233 org.iso.mpeg.mpegj.scene.Field.CoordinateInterpolator

V.5.233.1 Syntax

```
public static interface Field.CoordinateInterpolator
```

V.5.233.2 Description

An interface defining constants for the field defIDs of the CoordinateInterpolator node.

Member Summary	
Fields	
int	key
int	keyValue

V.5.233.3 Fields

key
`public static final int key`

The defID for the key field = 1.

keyValue
`public static final int keyValue`

The defID for the keyValue field = 2.

V.5.234 org.iso.mpeg.mpegj.scene.Field.CoordinateInterpolator2D

V.5.234.1 Syntax

```
public static interface Field.CoordinateInterpolator2D
```

V.5.234.2 Description

An interface defining constants for the field defIDs of the CoordinateInterpolator2D node.

Member Summary	
Fields	

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000611

int	key
int	keyValue

V.5.234.3 Fields

key
`public static final int key`

The defID for the key field = 1.

keyValue
`public static final int keyValue`

The defID for the keyValue field = 2.

V.5.235 org.iso.mpeg.mpegj.scene.Field.Curve2D

V.5.235.1 Syntax

```
public static interface Field.Curve2D
```

V.5.235.2 Description

An interface defining constants for the field defIDs of the Curve2D node.

Member Summary	
Fields	
int	point
int	fineness
int	type

V.5.235.3 Fields

fineness
`public static final int fineness`

The defID for the fineness field = 1.

point
`public static final int point`

The defID for the point field = 0.

type
`public static final int type`

The defID for the type field = 2.

V.5.236 org.iso.mpeg.mpegj.scene.Field.Cylinder

V.5.236.1 Syntax

```
public static interface Field.Cylinder
```

V.5.236.2 Description

An interface defining constants for the field defIDs of the Cylinder node.

Member Summary	
Fields	
int	bottom
int	height
int	radius
int	side
int	top

V.5.236.3 Fields

bottom

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000612

```
public static final int bottom
```

The defID for the bottom field = 0.

height

```
public static final int height
```

The defID for the height field = 1.

radius

```
public static final int radius
```

The defID for the radius field = 2.

side

```
public static final int side
```

The defID for the side field = 3.

top

```
public static final int top
```

The defID for the top field = 4.

V.5.237 org.iso.mpeg.mpegj.scene.Field.CylinderSensor

V.5.237.1 Syntax

```
public static interface Field.CylinderSensor
```

V.5.237.2 Description

An interface defining constants for the field defIDs of the CylinderSensor node.

Member Summary	
Fields	
int	autoOffset
int	diskAngle
int	enabled
int	maxAngle
int	minAngle
int	offset

V.5.237.3 Fields

autoOffset

```
public static final int autoOffset
```

The defID for the autoOffset field = 0.

diskAngle

```
public static final int diskAngle
```

The defID for the diskAngle field = 1.

enabled

```
public static final int enabled
```

The defID for the enabled field = 2.

maxAngle

```
public static final int maxAngle
```

The defID for the maxAngle field = 3.

minAngle

```
public static final int minAngle
```

The defID for the minAngle field = 4.

offset

```
public static final int offset
```

The defID for the offset field = 5.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000613

V.5.238 org.iso.mpeg.mpegj.scene.Field.DirectionalLight

V.5.238.1 Syntax

```
public static interface Field.DirectionalLight
```

V.5.238.2 Description

An interface defining constants for the field defIDs of the DirectionalLight node.

Member Summary	
Fields	
int	ambientIntensity
int	color
int	direction
int	intensity
int	on

V.5.238.3 Fields

ambientIntensity
`public static final int ambientIntensity`

The defID for the ambientIntensity field = 0.

color
`public static final int color`

The defID for the color field = 1.

direction
`public static final int direction`

The defID for the direction field = 2.

intensity
`public static final int intensity`

The defID for the intensity field = 3.

on
`public static final int on`

The defID for the on field = 4.

V.5.239 org.iso.mpeg.mpegj.scene.Field.DiscSensor

V.5.239.1 Syntax

```
public static interface Field.DiscSensor
```

V.5.239.2 Description

An interface defining constants for the field defIDs of the DiscSensor node.

Member Summary	
Fields	
int	autoOffset
int	enabled
int	maxAngle
int	minAngle
int	offset

V.5.239.3 Fields

autoOffset
`public static final int autoOffset`

The defID for the autoOffset field = 0.

enabled
`public static final int enabled`

The defID for the enabled field = 1.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000614

maxAngle
`public static final int maxAngle`

The defID for the maxAngle field = 2.

minAngle
`public static final int minAngle`

The defID for the minAngle field = 3.

offset
`public static final int offset`

The defID for the offset field = 4.

V.5.240 org.iso.mpeg.mpegj.scene.Field.ElevationGrid

V.5.240.1 Syntax

```
public static interface Field.ElevationGrid
```

V.5.240.2 Description

An interface defining constants for the field defIDs of the ElevationGrid node.

Member Summary	
Fields	
int	color
int	normal
int	texCoord
int	height
int	ccw
int	colorPerVertex
int	creaseAngle
int	normalPerVertex
int	solid
int	xDimension
int	xSpacing
int	zDimension
int	zSpacing

V.5.240.3 Fields

ccw
`public static final int ccw`

The defID for the ccw field = 5.

color
`public static final int color`

The defID for the color field = 1.

colorPerVertex
`public static final int colorPerVertex`

The defID for the colorPerVertex field = 6.

creaseAngle
`public static final int creaseAngle`

The defID for the creaseAngle field = 7.

height
`public static final int height`

The defID for the height field = 4.

normal
`public static final int normal`

The defID for the normal field = 2.

normalPerVertex
`public static final int normalPerVertex`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000615

The defID for the normalPerVertex field = 8.

solid
`public static final int solid`

The defID for the solid field = 9.

texCoord
`public static final int texCoord`

The defID for the texCoord field = 3.

xDimension
`public static final int xDimension`

The defID for the xDimension field = 10.

xSpacing
`public static final int xSpacing`

The defID for the xSpacing field = 11.

zDimension
`public static final int zDimension`

The defID for the zDimension field = 12.

zSpacing
`public static final int zSpacing`

The defID for the zSpacing field = 13.

V.5.241 org.iso.mpeg.mpegj.scene.Field.Expression

V.5.241.1 Syntax

```
public static interface Field.Expression
```

V.5.241.2 Description

An interface defining constants for the field defIDs of the Expression node.

Member Summary	
Fields	
int	expression_select1
int	expression_intensity1
int	expression_select2
int	expression_intensity2
int	init_face
int	expression_def

V.5.241.3 Fields

expression_def
`public static final int expression_def`

The defID for the expression_def field = 5.

expression_intensity1
`public static final int expression_intensity1`

The defID for the expression_intensity1 field = 1.

expression_intensity2
`public static final int expression_intensity2`

The defID for the expression_intensity2 field = 3.

expression_select1
`public static final int expression_select1`

The defID for the expression_select1 field = 0.

expression_select2
`public static final int expression_select2`

The defID for the expression_select2 field = 2.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000616

init_face
`public static final int init_face`

The defID for the init_face field = 4.

V.5.242 org.iso.mpeg.mpegj.scene.Field.Extrusion

V.5.242.1 Syntax

```
public static interface Field.Extrusion
```

V.5.242.2 Description

An interface defining constants for the field defIDs of the Extrusion node.

Member Summary	
Fields	
int	beginCap
int	ccw
int	convex
int	creaseAngle
int	crossSection
int	endCap
int	orientation
int	scale
int	solid
int	spine

V.5.242.3 Fields

beginCap
`public static final int beginCap`

The defID for the beginCap field = 4.

ccw
`public static final int ccw`

The defID for the ccw field = 5.

convex
`public static final int convex`

The defID for the convex field = 6.

creaseAngle
`public static final int creaseAngle`

The defID for the creaseAngle field = 7.

crossSection
`public static final int crossSection`

The defID for the crossSection field = 8.

endCap
`public static final int endCap`

The defID for the endCap field = 9.

orientation
`public static final int orientation`

The defID for the orientation field = 10.

scale
`public static final int scale`

The defID for the scale field = 11.

solid
`public static final int solid`

The defID for the solid field = 12.

spine

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000617

```
public static final int spine
```

The defID for the spine field = 13.

V.5.243 org.iso.mpeg.mpegj.scene.Field.Face

V.5.243.1 Syntax

```
public static interface Field.Face
```

V.5.243.2 Description

An interface defining constants for the field defIDs of the Face node.

Member Summary	
Fields	
int	fap
int	fdp
int	fit
int	ttsSource
int	renderedFace

V.5.243.3 Fields

fap
```
public static final int fap
```

The defID for the fap field = 0.

fdp
```
public static final int fdp
```

The defID for the fdp field = 1.

fit
```
public static final int fit
```

The defID for the fit field = 2.

renderedFace
```
public static final int renderedFace
```

The defID for the renderedFace field = 4.

ttsSource
```
public static final int ttsSource
```

The defID for the ttsSource field = 3.

V.5.244 org.iso.mpeg.mpegj.scene.Field.FaceDefMesh

V.5.244.1 Syntax

```
public static interface Field.FaceDefMesh
```

V.5.244.2 Description

An interface defining constants for the field defIDs of the FaceDefMesh node.

Member Summary	
Fields	
int	faceSceneGraphNode
int	intervalBorders
int	coordIndex
int	displacements

V.5.244.3 Fields

coordIndex
```
public static final int coordIndex
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000618

The defID for the coordIndex field = 2.

displacements
`public static final int displacements`

The defID for the displacements field = 3.

faceSceneGraphNode
`public static final int faceSceneGraphNode`

The defID for the faceSceneGraphNode field = 0.

intervalBorders
`public static final int intervalBorders`

The defID for the intervalBorders field = 1.

V.5.245 org.iso.mpeg.mpegj.scene.Field.FaceDefTables

V.5.245.1 Syntax

```
public static interface Field.FaceDefTables
```

V.5.245.2 Description

An interface defining constants for the field defIDs of the FaceDefTables node.

Member Summary	
Fields	
int	fapID
int	highLevelSelect
int	faceDefMesh
int	faceDefTransform

V.5.245.3 Fields

faceDefMesh
`public static final int faceDefMesh`

The defID for the faceDefMesh field = 2.

faceDefTransform
`public static final int faceDefTransform`

The defID for the faceDefTransform field = 3.

fapID
`public static final int fapID`

The defID for the fapID field = 0.

highLevelSelect
`public static final int highLevelSelect`

The defID for the highLevelSelect field = 1.

V.5.246 org.iso.mpeg.mpegj.scene.Field.FaceDefTransform

V.5.246.1 Syntax

```
public static interface Field.FaceDefTransform
```

V.5.246.2 Description

An interface defining constants for the field defIDs of the FaceDefTransform node.

Member Summary	
Fields	
int	faceSceneGraphNode
int	fieldId
int	rotationDef
int	scaleDef
int	translationDef

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000619

V.5.246.3 Fields

faceSceneGraphNode
`public static final int faceSceneGraphNode`

The defID for the faceSceneGraphNode field = 0.

fieldId
`public static final int fieldId`

The defID for the fieldId field = 1.

rotationDef
`public static final int rotationDef`

The defID for the rotationDef field = 2.

scaleDef
`public static final int scaleDef`

The defID for the scaleDef field = 3.

translationDef
`public static final int translationDef`

The defID for the translationDef field = 4.

V.5.247 org.iso.mpeg.mpegj.scene.Field.FAP

V.5.247.1 Syntax

```
public static interface Field.FAP
```

V.5.247.2 Description

An interface defining constants for the field defIDs of the FAP node.

Member Summary	
Fields	
int	viseme
int	expression
int	open_jaw
int	lower_t_midlip
int	raise_b_midlip
int	stretch_l_corner
int	stretch_r_corner
int	lower_t_lip_lm
int	lower_t_lip_rm
int	lower_b_lip_lm
int	lower_b_lip_rm
int	raise_l_cornerlip
int	raise_r_cornerlip
int	thrust_jaw
int	shift_jaw
int	push_b_lip
int	push_t_lip
int	depress_chin
int	close_t_l_eyelid
int	close_t_r_eyelid
int	close_b_l_eyelid
int	close_b_r_eyelid
int	yaw_l_eyeball
int	yaw_r_eyeball
int	pitch_l_eyeball
int	pitch_r_eyeball
int	thrust_l_eyeball
int	thrust_r_eyeball
int	dilate_l_pupil
int	dilate_r_pupil
int	raise_l_i_eyebrow
int	raise_r_i_eyebrow
int	raise_l_m_eyebrow

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000620

int	raise_r_m_eyebrow
int	raise_l_o_eyebrow
int	raise_r_o_eyebrow
int	squeeze_l_eyebrow
int	squeeze_r_eyebrow
int	puff_l_cheek
int	puff_r_cheek
int	lift_l_cheek
int	lift_r_cheek
int	shift_tongue_tip
int	raise_tongue_tip
int	thrust_tongue_tip
int	raise_tongue
int	tongue_roll
int	head_pitch
int	head_yaw
int	head_roll
int	lower_t_midlip_o
int	raise_b_midlip_o
int	stretch_l_cornerlip
int	stretch_r_cornerlip
int	lower_t_lip_lm_o
int	lower_t_lip_rm_o
int	raise_b_lip_lm_o
int	raise_b_lip_rm_o
int	raise_l_cornerlip_o
int	raise_r_cornerlip_o
int	stretch_l_nose
int	stretch_r_nose
int	raise_nose
int	bend_nose
int	raise_l_ear
int	raise_r_ear
int	pull_l_ear
int	pull_r_ear

V.5.247.3 Fields

bend_nose
`public static final int bend_nose`

The defID for the bend_nose field = 63.

close_b_l_eyelid
`public static final int close_b_l_eyelid`

The defID for the close_b_l_eyelid field = 20.

close_b_r_eyelid
`public static final int close_b_r_eyelid`

The defID for the close_b_r_eyelid field = 21.

close_t_l_eyelid
`public static final int close_t_l_eyelid`

The defID for the close_t_l_eyelid field = 18.

close_t_r_eyelid
`public static final int close_t_r_eyelid`

The defID for the close_t_r_eyelid field = 19.

depress_chin
`public static final int depress_chin`

The defID for the depress_chin field = 17.

dilate_l_pupil
`public static final int dilate_l_pupil`

The defID for the dilate_l_pupil field = 28.

dilate_r_pupil
`public static final int dilate_r_pupil`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000621

The defID for the dilate_r_pupil field = 29.

expression
`public static final int expression`

The defID for the expression field = 1.

head_pitch
`public static final int head_pitch`

The defID for the head_pitch field = 47.

head_roll
`public static final int head_roll`

The defID for the head_roll field = 49.

head_yaw
`public static final int head_yaw`

The defID for the head_yaw field = 48.

lift_l_cheek
`public static final int lift_l_cheek`

The defID for the lift_l_cheek field = 40.

lift_r_cheek
`public static final int lift_r_cheek`

The defID for the lift_r_cheek field = 41.

lower_b_lip_lm
`public static final int lower_b_lip_lm`

The defID for the lower_b_lip_lm field = 9.

lower_b_lip_rm
`public static final int lower_b_lip_rm`

The defID for the lower_b_lip_rm field = 10.

lower_t_lip_lm
`public static final int lower_t_lip_lm`

The defID for the lower_t_lip_lm field = 7.

lower_t_lip_lm_o
`public static final int lower_t_lip_lm_o`

The defID for the lower_t_lip_lm_o field = 54.

lower_t_lip_rm
`public static final int lower_t_lip_rm`

The defID for the lower_t_lip_rm field = 8.

lower_t_lip_rm_o
`public static final int lower_t_lip_rm_o`

The defID for the lower_t_lip_rm_o field = 55.

lower_t_midlip
`public static final int lower_t_midlip`

The defID for the lower_t_midlip field = 3.

lower_t_midlip_o
`public static final int lower_t_midlip_o`

The defID for the lower_t_midlip_o field = 50.

open_jaw
`public static final int open_jaw`

The defID for the open_jaw field = 2.

pitch_l_eyeball
`public static final int pitch_l_eyeball`

The defID for the pitch_l_eyeball field = 24.

pitch_r_eyeball
`public static final int pitch_r_eyeball`

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000622

The defID for the pitch_r_eyeball field = 25.

puff_l_cheek
`public static final int puff_l_cheek`

The defID for the puff_l_cheek field = 38.

puff_r_cheek
`public static final int puff_r_cheek`

The defID for the puff_r_cheek field = 39.

pull_l_ear
`public static final int pull_l_ear`

The defID for the pull_l_ear field = 66.

pull_r_ear
`public static final int pull_r_ear`

The defID for the pull_r_ear field = 67.

push_b_lip
`public static final int push_b_lip`

The defID for the push_b_lip field = 15.

push_t_lip
`public static final int push_t_lip`

The defID for the push_t_lip field = 16.

raise_b_lip_lm_o
`public static final int raise_b_lip_lm_o`

The defID for the raise_b_lip_lm_o field = 56.

raise_b_lip_rm_o
`public static final int raise_b_lip_rm_o`

The defID for the raise_b_lip_rm_o field = 57.

raise_b_midlip
`public static final int raise_b_midlip`

The defID for the raise_b_midlip field = 4.

raise_b_midlip_o
`public static final int raise_b_midlip_o`

The defID for the raise_b_midlip_o field = 51.

raise_l_cornerlip
`public static final int raise_l_cornerlip`

The defID for the raise_l_cornerlip field = 11.

raise_l_cornerlip_o
`public static final int raise_l_cornerlip_o`

The defID for the raise_l_cornerlip_o field = 58.

raise_l_ear
`public static final int raise_l_ear`

The defID for the raise_l_ear field = 64.

raise_l_i_eyebrow
`public static final int raise_l_i_eyebrow`

The defID for the raise_l_i_eyebrow field = 30.

raise_l_m_eyebrow
`public static final int raise_l_m_eyebrow`

The defID for the raise_l_m_eyebrow field = 32.

raise_l_o_eyebrow
`public static final int raise_l_o_eyebrow`

The defID for the raise_l_o_eyebrow field = 34.

raise_nose
`public static final int raise_nose`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000623

The defID for the raise_nose field = 62.

raise_r_cornerlip
`public static final int raise_r_cornerlip`

The defID for the raise_r_cornerlip field = 12.

raise_r_cornerlip_o
`public static final int raise_r_cornerlip_o`

The defID for the raise_r_cornerlip_o field = 59.

raise_r_ear
`public static final int raise_r_ear`

The defID for the raise_r_ear field = 65.

raise_r_i_eyebrow
`public static final int raise_r_i_eyebrow`

The defID for the raise_r_i_eyebrow field = 31.

raise_r_m_eyebrow
`public static final int raise_r_m_eyebrow`

The defID for the raise_r_m_eyebrow field = 33.

raise_r_o_eyebrow
`public static final int raise_r_o_eyebrow`

The defID for the raise_r_o_eyebrow field = 35.

raise_tongue
`public static final int raise_tongue`

The defID for the raise_tongue field = 45.

raise_tongue_tip
`public static final int raise_tongue_tip`

The defID for the raise_tongue_tip field = 43.

shift_jaw
`public static final int shift_jaw`

The defID for the shift_jaw field = 14.

shift_tongue_tip
`public static final int shift_tongue_tip`

The defID for the shift_tongue_tip field = 42.

squeeze_l_eyebrow
`public static final int squeeze_l_eyebrow`

The defID for the squeeze_l_eyebrow field = 36.

squeeze_r_eyebrow
`public static final int squeeze_r_eyebrow`

The defID for the squeeze_r_eyebrow field = 37.

stretch_l_corner
`public static final int stretch_l_corner`

The defID for the stretch_l_corner field = 5.

stretch_l_cornerlip
`public static final int stretch_l_cornerlip`

The defID for the stretch_l_cornerlip field = 52.

stretch_l_nose
`public static final int stretch_l_nose`

The defID for the stretch_l_nose field = 60.

stretch_r_corner
`public static final int stretch_r_corner`

The defID for the stretch_r_corner field = 6.

stretch_r_cornerlip
`public static final int stretch_r_cornerlip`

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000624

The defID for the stretch_r_cornerlip field = 53.

stretch_r_nose
`public static final int stretch_r_nose`

The defID for the stretch_r_nose field = 61.

thrust_jaw
`public static final int thrust_jaw`

The defID for the thrust_jaw field = 13.

thrust_l_eyeball
`public static final int thrust_l_eyeball`

The defID for the thrust_l_eyeball field = 26.

thrust_r_eyeball
`public static final int thrust_r_eyeball`

The defID for the thrust_r_eyeball field = 27.

thrust_tongue_tip
`public static final int thrust_tongue_tip`

The defID for the thrust_tongue_tip field = 44.

tongue_roll
`public static final int tongue_roll`

The defID for the tongue_roll field = 46.

viseme
`public static final int viseme`

The defID for the viseme field = 0.

yaw_l_eyeball
`public static final int yaw_l_eyeball`

The defID for the yaw_l_eyeball field = 22.

yaw_r_eyeball
`public static final int yaw_r_eyeball`

The defID for the yaw_r_eyeball field = 23.

V.5.248 org.iso.mpeg.mpegj.scene.Field.FDP

V.5.248.1 Syntax

```
public static interface Field.FDP
```

V.5.248.2 Description

An interface defining constants for the field defIDs of the FDP node.

Member Summary	
Fields	
int	featurePointsCoord
int	textureCoord
int	faceDefTables
int	faceSceneGraph
int	useOrthoTexture

V.5.248.3 Fields

faceDefTables
`public static final int faceDefTables`

The defID for the faceDefTables field = 2.

faceSceneGraph
`public static final int faceSceneGraph`

The defID for the faceSceneGraph field = 3.

featurePointsCoord

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000625

```
public static final int featurePointsCoord
```

The defID for the featurePointsCoord field = 0.

textureCoord
```
public static final int textureCoord
```

The defID for the textureCoord field = 1.

useOrthoTexture
```
public static final int useOrthoTexture
```

The defID for the useOrthoTexture field = 4.

V.5.249 org.iso.mpeg.mpegj.scene.Field.FIT

V.5.249.1 Syntax

```
public static interface Field.FIT
```

V.5.249.2 Description

An interface defining constants for the field defIDs of the FIT node.

Member Summary	
Fields	
int	FAPs
int	Graph
int	numeratorExp
int	denominatorExp
int	numeratorImpulse
int	numeratorTerms
int	denominatorTerms
int	numeratorCoefs
int	denominatorCoefs

V.5.249.3 Fields

denominatorCoefs
```
public static final int denominatorCoefs
```

The defID for the denominatorCoefs field = 8.

denominatorExp
```
public static final int denominatorExp
```

The defID for the denominatorExp field = 3.

denominatorTerms
```
public static final int denominatorTerms
```

The defID for the denominatorTerms field = 6.

FAPs
```
public static final int FAPs
```

The defID for the FAPs field = 0.

Graph
```
public static final int Graph
```

The defID for the Graph field = 1.

numeratorCoefs
```
public static final int numeratorCoefs
```

The defID for the numeratorCoefs field = 7.

numeratorExp
```
public static final int numeratorExp
```

The defID for the numeratorExp field = 2.

numeratorImpulse
```
public static final int numeratorImpulse
```

The defID for the numeratorImpulse field = 4.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000626

numeratorTerms
`public static final int numeratorTerms`

The defID for the numeratorTerms field = 5.

V.5.250 org.iso.mpeg.mpegj.scene.Field.Fog

V.5.250.1 Syntax

```
public static interface Field.Fog
```

V.5.250.2 Description

An interface defining constants for the field defIDs of the Fog node.

Member Summary	
Fields	
int	color
int	fogType
int	visibilityRange

V.5.250.3 Fields

color
`public static final int color`

The defID for the color field = 0.

fogType
`public static final int fogType`

The defID for the fogType field = 1.

visibilityRange
`public static final int visibilityRange`

The defID for the visibilityRange field = 2.

V.5.251 org.iso.mpeg.mpegj.scene.Field.FontStyle

V.5.251.1 Syntax

```
public static interface Field.FontStyle
```

V.5.251.2 Description

An interface defining constants for the field defIDs of the FontStyle node.

Member Summary	
Fields	
int	family
int	horizontal
int	justify
int	language
int	leftToRight
int	size
int	spacing
int	style
int	topToBottom

V.5.251.3 Fields

family
`public static final int family`

The defID for the family field = 0.

horizontal
`public static final int horizontal`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000627

The defID for the horizontal field = 1.

justify
`public static final int justify`

The defID for the justify field = 2.

language
`public static final int language`

The defID for the language field = 3.

leftToRight
`public static final int leftToRight`

The defID for the leftToRight field = 4.

size
`public static final int size`

The defID for the size field = 5.

spacing
`public static final int spacing`

The defID for the spacing field = 6.

style
`public static final int style`

The defID for the style field = 7.

topToBottom
`public static final int topToBottom`

The defID for the topToBottom field = 8.

V.5.252 org.iso.mpeg.mpegj.scene.Field.Form

V.5.252.1 Syntax

```
public static interface Field.Form
```

V.5.252.2 Description

An interface defining constants for the field defIDs of the Form node.

Member Summary	
Fields	
int	children
int	size
int	groups
int	constraints
int	groupsIndex

V.5.252.3 Fields

children
`public static final int children`

The defID for the children field = 2.

constraints
`public static final int constraints`

The defID for the constraints field = 5.

groups
`public static final int groups`

The defID for the groups field = 4.

groupsIndex
`public static final int groupsIndex`

The defID for the groupsIndex field = 6.

size

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000628

```
public static final int size
```

The defID for the size field = 3.

V.5.253 org.iso.mpeg.mpegj.scene.Field.Group

V.5.253.1 Syntax

```
public static interface Field.Group
```

V.5.253.2 Description

An interface defining constants for the field defIDs of the Group node.

Member Summary	
Fields	
int	children

V.5.253.3 Fields

children
```
public static final int children
```

The defID for the children field = 2.

V.5.254 org.iso.mpeg.mpegj.scene.Field.ImageTexture

V.5.254.1 Syntax

```
public static interface Field.ImageTexture
```

V.5.254.2 Description

An interface defining constants for the field defIDs of the ImageTexture node.

Member Summary	
Fields	
int	url
int	repeatS
int	repeatT

V.5.254.3 Fields

repeatS
```
public static final int repeatS
```

The defID for the repeatS field = 1.

repeatT
```
public static final int repeatT
```

The defID for the repeatT field = 2.

url
```
public static final int url
```

The defID for the url field = 0.

V.5.255 org.iso.mpeg.mpegj.scene.Field.IndexedFaceSet

V.5.255.1 Syntax

```
public static interface Field.IndexedFaceSet
```

V.5.255.2 Description

An interface defining constants for the field defIDs of the IndexedFaceSet node.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000629

Member Summary	
Fields	
int	color
int	coord
int	normal
int	texCoord
int	ccw
int	colorIndex
int	colorPerVertex
int	convex
int	coordIndex
int	creaseAngle
int	normalIndex
int	normalPerVertex
int	solid
int	texCoordIndex

V.5.255.3 Fields

ccw
`public static final int ccw`

The defID for the ccw field = 8.

color
`public static final int color`

The defID for the color field = 4.

colorIndex
`public static final int colorIndex`

The defID for the colorIndex field = 9.

colorPerVertex
`public static final int colorPerVertex`

The defID for the colorPerVertex field = 10.

convex
`public static final int convex`

The defID for the convex field = 11.

coord
`public static final int coord`

The defID for the coord field = 5.

coordIndex
`public static final int coordIndex`

The defID for the coordIndex field = 12.

creaseAngle
`public static final int creaseAngle`

The defID for the creaseAngle field = 13.

normal
`public static final int normal`

The defID for the normal field = 6.

normalIndex
`public static final int normalIndex`

The defID for the normalIndex field = 14.

normalPerVertex
`public static final int normalPerVertex`

The defID for the normalPerVertex field = 15.

solid
`public static final int solid`

The defID for the solid field = 16.

texCoord

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000630

```
public static final int texCoord
```

The defID for the texCoord field = 7.

texCoordIndex
```
public static final int texCoordIndex
```

The defID for the texCoordIndex field = 17.

V.5.256 org.iso.mpeg.mpegj.scene.Field.IndexedFaceSet2D

V.5.256.1 Syntax

```
public static interface Field.IndexedFaceSet2D
```

V.5.256.2 Description

An interface defining constants for the field defIDs of the IndexedFaceSet2D node.

Member Summary	
Fields	
int	color
int	coord
int	texCoord
int	colorIndex
int	colorPerVertex
int	convex
int	coordIndex
int	texCoordIndex

V.5.256.3 Fields

color
```
public static final int color
```

The defID for the color field = 3.

colorIndex
```
public static final int colorIndex
```

The defID for the colorIndex field = 6.

colorPerVertex
```
public static final int colorPerVertex
```

The defID for the colorPerVertex field = 7.

convex
```
public static final int convex
```

The defID for the convex field = 8.

coord
```
public static final int coord
```

The defID for the coord field = 4.

coordIndex
```
public static final int coordIndex
```

The defID for the coordIndex field = 9.

texCoord
```
public static final int texCoord
```

The defID for the texCoord field = 5.

texCoordIndex
```
public static final int texCoordIndex
```

The defID for the texCoordIndex field = 10.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000631

V.5.257 org.iso.mpeg.mpegj.scene.Field.IndexedLineSet

V.5.257.1 Syntax

```
public static interface Field.IndexedLineSet
```

V.5.257.2 Description

An interface defining constants for the field defIDs of the IndexedLineSet node.

Member Summary	
Fields	
int	color
int	coord
int	colorIndex
int	colorPerVertex
int	coordIndex

V.5.257.3 Fields

color
```
public static final int color
```
The defID for the color field = 2.

colorIndex
```
public static final int colorIndex
```
The defID for the colorIndex field = 4.

colorPerVertex
```
public static final int colorPerVertex
```
The defID for the colorPerVertex field = 5.

coord
```
public static final int coord
```
The defID for the coord field = 3.

coordIndex
```
public static final int coordIndex
```
The defID for the coordIndex field = 6.

V.5.258 org.iso.mpeg.mpegj.scene.Field.IndexedLineSet2D

V.5.258.1 Syntax

```
public static interface Field.IndexedLineSet2D
```

V.5.258.2 Description

An interface defining constants for the field defIDs of the IndexedLineSet2D node.

Member Summary	
Fields	
int	color
int	coord
int	colorIndex
int	colorPerVertex
int	coordIndex

V.5.258.3 Fields

color
```
public static final int color
```
The defID for the color field = 2.

colorIndex

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000632

```
public static final int colorIndex
```

The defID for the colorIndex field = 4.

colorPerVertex

```
public static final int colorPerVertex
```

The defID for the colorPerVertex field = 5.

coord

```
public static final int coord
```

The defID for the coord field = 3.

coordIndex

```
public static final int coordIndex
```

The defID for the coordIndex field = 6.

V.5.259 org.iso.mpeg.mpegj.scene.Field.Inline

V.5.259.1 Syntax

```
public static interface Field.Inline
```

V.5.259.2 Description

An interface defining constants for the field defIDs of the Inline node.

Member Summary	
Fields	
int	url

V.5.259.3 Fields

url

```
public static final int url
```

The defID for the url field = 0.

V.5.260 org.iso.mpeg.mpegj.scene.Field.Layer2D

V.5.260.1 Syntax

```
public static interface Field.Layer2D
```

V.5.260.2 Description

An interface defining constants for the field defIDs of the Layer2D node.

Member Summary	
Fields	
int	children
int	size
int	background
int	viewport

V.5.260.3 Fields

background

```
public static final int background
```

The defID for the background field = 4.

children

```
public static final int children
```

The defID for the children field = 2.

size

```
public static final int size
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000633

The defID for the size field = 3.

viewport
`public static final int viewport`

The defID for the viewport field = 5.

V.5.261 org.iso.mpeg.mpegj.scene.Field.Layer3D

V.5.261.1 Syntax

```
public static interface Field.Layer3D
```

V.5.261.2 Description

An interface defining constants for the field defIDs of the Layer3D node.

Member Summary	
Fields	
int	children
int	size
int	background
int	fog
int	navigationInfo
int	viewpoint

V.5.261.3 Fields

background
`public static final int background`

The defID for the background field = 4.

children
`public static final int children`

The defID for the children field = 2.

fog
`public static final int fog`

The defID for the fog field = 5.

navigationInfo
`public static final int navigationInfo`

The defID for the navigationInfo field = 6.

size
`public static final int size`

The defID for the size field = 3.

viewpoint
`public static final int viewpoint`

The defID for the viewpoint field = 7.

V.5.262 org.iso.mpeg.mpegj.scene.Field.Layout

V.5.262.1 Syntax

```
public static interface Field.Layout
```

V.5.262.2 Description

An interface defining constants for the field defIDs of the Layout node.

Member Summary	
Fields	
int	children
int	wrap
int	size

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000634

int	horizontal
int	justify
int	leftToRight
int	topToBottom
int	spacing
int	smoothScroll
int	loop
int	scrollVertical
int	scrollRate

V.5.262.3 Fields

children
`public static final int children`

The defID for the children field = 2.

horizontal
`public static final int horizontal`

The defID for the horizontal field = 5.

justify
`public static final int justify`

The defID for the justify field = 6.

leftToRight
`public static final int leftToRight`

The defID for the leftToRight field = 7.

loop
`public static final int loop`

The defID for the loop field = 11.

scrollRate
`public static final int scrollRate`

The defID for the scrollRate field = 13.

scrollVertical
`public static final int scrollVertical`

The defID for the scrollVertical field = 12.

size
`public static final int size`

The defID for the size field = 4.

smoothScroll
`public static final int smoothScroll`

The defID for the smoothScroll field = 10.

spacing
`public static final int spacing`

The defID for the spacing field = 9.

topToBottom
`public static final int topToBottom`

The defID for the topToBottom field = 8.

wrap
`public static final int wrap`

The defID for the wrap field = 3.

V.5.263 org.iso.mpeg.mpegj.scene.Field.LineProperties

V.5.263.1 Syntax

```
public static interface Field.LineProperties
```

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000635

V.5.263.2 Description

An interface defining constants for the field defIDs of the LineProperties node.

Member Summary	
Fields	
int	lineColor
int	lineStyle
int	width

V.5.263.3 Fields

lineColor
`public static final int lineColor`

The defID for the lineColor field = 0.

lineStyle
`public static final int lineStyle`

The defID for the lineStyle field = 1.

width
`public static final int width`

The defID for the width field = 2.

V.5.264 org.iso.mpeg.mpegj.scene.Field.ListeningPoint

V.5.264.1 Syntax

```
public static interface Field.ListeningPoint
```

V.5.264.2 Description

An interface defining constants for the field defIDs of the ListeningPoint node.

Member Summary	
Fields	
int	jump
int	orientation
int	position
int	description

V.5.264.3 Fields

description
`public static final int description`

The defID for the description field = 4.

jump
`public static final int jump`

The defID for the jump field = 1.

orientation
`public static final int orientation`

The defID for the orientation field = 2.

position
`public static final int position`

The defID for the position field = 3.

V.5.265 org.iso.mpeg.mpegj.scene.Field.LOD

V.5.265.1 Syntax

```
public static interface Field.LOD
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000636

V.5.265.2 Description

An interface defining constants for the field defIDs of the LOD node.

Member Summary	
Fields	
int	level
int	center
int	range

V.5.265.3 Fields

center
`public static final int center`

The defID for the center field = 1.

level
`public static final int level`

The defID for the level field = 0.

range
`public static final int range`

The defID for the range field = 2.

V.5.266 org.iso.mpeg.mpegj.scene.Field.Material

V.5.266.1 Syntax

```
public static interface Field.Material
```

V.5.266.2 Description

An interface defining constants for the field defIDs of the Material node.

Member Summary	
Fields	
int	ambientIntensity
int	diffuseColor
int	emissiveColor
int	shininess
int	specularColor
int	transparency

V.5.266.3 Fields

ambientIntensity
`public static final int ambientIntensity`

The defID for the ambientIntensity field = 0.

diffuseColor
`public static final int diffuseColor`

The defID for the diffuseColor field = 1.

emissiveColor
`public static final int emissiveColor`

The defID for the emissiveColor field = 2.

shininess
`public static final int shininess`

The defID for the shininess field = 3.

specularColor
`public static final int specularColor`

The defID for the specularColor field = 4.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000637

transparency
`public static final int transparency`

The defID for the transparency field = 5.

V.5.267 org.iso.mpeg.mpegj.scene.Field.Material2D

V.5.267.1 Syntax

```
public static interface Field.Material2D
```

V.5.267.2 Description

An interface defining constants for the field defIDs of the Material2D node.

Member Summary	
Fields	
int	emissiveColor
int	filled
int	lineProps
int	transparency

V.5.267.3 Fields

emissiveColor
`public static final int emissiveColor`

The defID for the emissiveColor field = 0.

filled
`public static final int filled`

The defID for the filled field = 1.

lineProps
`public static final int lineProps`

The defID for the lineProps field = 2.

transparency
`public static final int transparency`

The defID for the transparency field = 3.

V.5.268 org.iso.mpeg.mpegj.scene.Field.MovieTexture

V.5.268.1 Syntax

```
public static interface Field.MovieTexture
```

V.5.268.2 Description

An interface defining constants for the field defIDs of the MovieTexture node.

Member Summary	
Fields	
int	loop
int	speed
int	startTime
int	stopTime
int	url
int	repeatS
int	repeatT

V.5.268.3 Fields

loop
`public static final int loop`

The defID for the loop field = 0.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000638

repeatS
`public static final int repeatS`

The defID for the repeatS field = 5.

repeatT
`public static final int repeatT`

The defID for the repeatT field = 6.

speed
`public static final int speed`

The defID for the speed field = 1.

startTime
`public static final int startTime`

The defID for the startTime field = 2.

stopTime
`public static final int stopTime`

The defID for the stopTime field = 3.

url
`public static final int url`

The defID for the url field = 4.

V.5.269 org.iso.mpeg.mpegj.scene.Field.NavigationInfo

V.5.269.1 Syntax

```
public static interface Field.NavigationInfo
```

V.5.269.2 Description

An interface defining constants for the field defIDs of the NavigationInfo node.

Member Summary	
Fields	
int	avatarSize
int	headlight
int	speed
int	type
int	visibilityLimit

V.5.269.3 Fields

avatarSize
`public static final int avatarSize`

The defID for the avatarSize field = 1.

headlight
`public static final int headlight`

The defID for the headlight field = 2.

speed
`public static final int speed`

The defID for the speed field = 3.

type
`public static final int type`

The defID for the type field = 4.

visibilityLimit
`public static final int visibilityLimit`

The defID for the visibilityLimit field = 5.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000639

V.5.270 org.iso.mpeg.mpegj.scene.Field.Normal

V.5.270.1 Syntax

```
public static interface Field.Normal
```

V.5.270.2 Description

An interface defining constants for the field defIDs of the Normal node.

Member Summary	
Fields	
int	vector

V.5.270.3 Fields

vector
```
public static final int vector
```
The defID for the vector field = 0.

V.5.271 org.iso.mpeg.mpegj.scene.Field.NormalInterpolator

V.5.271.1 Syntax

```
public static interface Field.NormalInterpolator
```

V.5.271.2 Description

An interface defining constants for the field defIDs of the NormalInterpolator node.

Member Summary	
Fields	
int	key
int	keyValue

V.5.271.3 Fields

key
```
public static final int key
```
The defID for the key field = 1.

keyValue
```
public static final int keyValue
```
The defID for the keyValue field = 2.

V.5.272 org.iso.mpeg.mpegj.scene.Field.OrderedGroup

V.5.272.1 Syntax

```
public static interface Field.OrderedGroup
```

V.5.272.2 Description

An interface defining constants for the field defIDs of the OrderedGroup node.

Member Summary	
Fields	
int	children
int	order

V.5.272.3 Fields

children
```
public static final int children
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000640

The defID for the children field = 2.

order
`public static final int order`

The defID for the order field = 3.

V.5.273 org.iso.mpeg.mpegj.scene.Field.OrientationInterpolator

V.5.273.1 Syntax

```
public static interface Field.OrientationInterpolator
```

V.5.273.2 Description

An interface defining constants for the field defIDs of the OrientationInterpolator node.

Member Summary	
Fields	
int	key
int	keyValue

V.5.273.3 Fields

key
`public static final int key`

The defID for the key field = 1.

keyValue
`public static final int keyValue`

The defID for the keyValue field = 2.

V.5.274 org.iso.mpeg.mpegj.scene.Field.PixelTexture

V.5.274.1 Syntax

```
public static interface Field.PixelTexture
```

V.5.274.2 Description

An interface defining constants for the field defIDs of the PixelTexture node.

Member Summary	
Fields	
int	image
int	repeatS
int	repeatT

V.5.274.3 Fields

image
`public static final int image`

The defID for the image field = 0.

repeatS
`public static final int repeatS`

The defID for the repeatS field = 1.

repeatT
`public static final int repeatT`

The defID for the repeatT field = 2.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000641

V.5.275 org.iso.mpeg.mpegj.scene.Field.PlaneSensor

V.5.275.1 Syntax

```
public static interface Field.PlaneSensor
```

V.5.275.2 Description

An interface defining constants for the field defIDs of the PlaneSensor node.

Member Summary	
Fields	
int	autoOffset
int	enabled
int	maxPosition
int	minPosition
int	offset

V.5.275.3 Fields

autoOffset
`public static final int autoOffset`

The defID for the autoOffset field = 0.

enabled
`public static final int enabled`

The defID for the enabled field = 1.

maxPosition
`public static final int maxPosition`

The defID for the maxPosition field = 2.

minPosition
`public static final int minPosition`

The defID for the minPosition field = 3.

offset
`public static final int offset`

The defID for the offset field = 4.

V.5.276 org.iso.mpeg.mpegj.scene.Field.PlaneSensor2D

V.5.276.1 Syntax

```
public static interface Field.PlaneSensor2D
```

V.5.276.2 Description

An interface defining constants for the field defIDs of the PlaneSensor2D node.

Member Summary	
Fields	
int	autoOffset
int	enabled
int	maxPosition
int	minPosition
int	offset

V.5.276.3 Fields

autoOffset
`public static final int autoOffset`

The defID for the autoOffset field = 0.

enabled

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000642

```
public static final int enabled
```

The defID for the enabled field = 1.

maxPosition
```
public static final int maxPosition
```

The defID for the maxPosition field = 2.

minPosition
```
public static final int minPosition
```

The defID for the minPosition field = 3.

offset
```
public static final int offset
```

The defID for the offset field = 4.

V.5.277 org.iso.mpeg.mpegj.scene.Field.PointLight

V.5.277.1 Syntax

```
public static interface Field.PointLight
```

V.5.277.2 Description

An interface defining constants for the field defIDs of the PointLight node.

Member Summary	
Fields	
int	ambientIntensity
int	attenuation
int	color
int	intensity
int	location
int	on
int	radius

V.5.277.3 Fields

ambientIntensity
```
public static final int ambientIntensity
```

The defID for the ambientIntensity field = 0.

attenuation
```
public static final int attenuation
```

The defID for the attenuation field = 1.

color
```
public static final int color
```

The defID for the color field = 2.

intensity
```
public static final int intensity
```

The defID for the intensity field = 3.

location
```
public static final int location
```

The defID for the location field = 4.

on
```
public static final int on
```

The defID for the on field = 5.

radius
```
public static final int radius
```

The defID for the radius field = 6.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000643

V.5.278 org.iso.mpeg.mpegj.scene.Field.PointSet

V.5.278.1 Syntax

```
public static interface Field.PointSet
```

V.5.278.2 Description

An interface defining constants for the field defIDs of the PointSet node.

Member Summary	
Fields	
int	color
int	coord

V.5.278.3 Fields

color
```
public static final int color
```
The defID for the color field = 0.

coord
```
public static final int coord
```
The defID for the coord field = 1.

V.5.279 org.iso.mpeg.mpegj.scene.Field.PointSet2D

V.5.279.1 Syntax

```
public static interface Field.PointSet2D
```

V.5.279.2 Description

An interface defining constants for the field defIDs of the PointSet2D node.

Member Summary	
Fields	
int	color
int	coord

V.5.279.3 Fields

color
```
public static final int color
```
The defID for the color field = 0.

coord
```
public static final int coord
```
The defID for the coord field = 1.

V.5.280 org.iso.mpeg.mpegj.scene.Field.PositionInterpolator

V.5.280.1 Syntax

```
public static interface Field.PositionInterpolator
```

V.5.280.2 Description

An interface defining constants for the field defIDs of the PositionInterpolator node.

Member Summary	
Fields	
int	key
int	keyValue

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000644

V.5.280.3 Fields

key
`public static final int key`

The defID for the key field = 1.

keyValue
`public static final int keyValue`

The defID for the keyValue field = 2.

V.5.281 org.iso.mpeg.mpegj.scene.Field.PositionInterpolator2D

V.5.281.1 Syntax

```
public static interface Field.PositionInterpolator2D
```

V.5.281.2 Description

An interface defining constants for the field defIDs of the PositionInterpolator2D node.

Member Summary	
Fields	
int	key
int	keyValue

V.5.281.3 Fields

key
`public static final int key`

The defID for the key field = 1.

keyValue
`public static final int keyValue`

The defID for the keyValue field = 2.

V.5.282 org.iso.mpeg.mpegj.scene.Field.ProximitySensor

V.5.282.1 Syntax

```
public static interface Field.ProximitySensor
```

V.5.282.2 Description

An interface defining constants for the field defIDs of the ProximitySensor node.

Member Summary	
Fields	
int	center
int	size
int	enabled

V.5.282.3 Fields

center
`public static final int center`

The defID for the center field = 0.

enabled
`public static final int enabled`

The defID for the enabled field = 2.

size
`public static final int size`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000645

The defID for the size field = 1.

V.5.283 org.iso.mpeg.mpegj.scene.Field.ProximitySensor2D

V.5.283.1 Syntax

```
public static interface Field.ProximitySensor2D
```

V.5.283.2 Description

An interface defining constants for the field defIDs of the ProximitySensor2D node.

Member Summary	
Fields	
int	center
int	size
int	enabled

V.5.283.3 Fields

center
```
public static final int center
```
The defID for the center field = 0.

enabled
```
public static final int enabled
```
The defID for the enabled field = 2.

size
```
public static final int size
```
The defID for the size field = 1.

V.5.284 org.iso.mpeg.mpegj.scene.Field.QuantizationParameter

V.5.284.1 Syntax

```
public static interface Field.QuantizationParameter
```

V.5.284.2 Description

An interface defining constants for the field defIDs of the QuantizationParameter node.

Member Summary	
Fields	
int	isLocal
int	position3DQuant
int	position3DMin
int	position3DMax
int	position3DNbBits
int	position2DQuant
int	position2DMin
int	position2DMax
int	position2DNbBits
int	drawOrderQuant
int	drawOrderMin
int	drawOrderMax
int	drawOrderNbBits
int	colorQuant
int	colorMin
int	colorMax
int	colorNbBits
int	textureCoordinateQuant
int	textureCoordinateMin
int	textureCoordinateMax
int	textureCoordinateNbBits
int	angleQuant
int	angleMin
int	angleMax

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000646

int	angleNbBits
int	scaleQuant
int	scaleMin
int	scaleMax
int	scaleNbBits
int	keyQuant
int	keyMin
int	keyMax
int	keyNbBits
int	normalQuant
int	normalNbBits
int	sizeQuant
int	sizeMin
int	sizeMax
int	sizeNbBits
int	useEfficientCoding

V.5.284.3 Fields

angleMax
`public static final int angleMax`

The defID for the angleMax field = 23.

angleMin
`public static final int angleMin`

The defID for the angleMin field = 22.

angleNbBits
`public static final int angleNbBits`

The defID for the angleNbBits field = 24.

angleQuant
`public static final int angleQuant`

The defID for the angleQuant field = 21.

colorMax
`public static final int colorMax`

The defID for the colorMax field = 15.

colorMin
`public static final int colorMin`

The defID for the colorMin field = 14.

colorNbBits
`public static final int colorNbBits`

The defID for the colorNbBits field = 16.

colorQuant
`public static final int colorQuant`

The defID for the colorQuant field = 13.

drawOrderMax
`public static final int drawOrderMax`

The defID for the drawOrderMax field = 11.

drawOrderMin
`public static final int drawOrderMin`

The defID for the drawOrderMin field = 10.

drawOrderNbBits
`public static final int drawOrderNbBits`

The defID for the drawOrderNbBits field = 12.

drawOrderQuant
`public static final int drawOrderQuant`

The defID for the drawOrderQuant field = 9.

isLocal

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000647

```
public static final int isLocal
```

The defID for the isLocal field = 0.

keyMax
```
public static final int keyMax
```

The defID for the keyMax field = 31.

keyMin
```
public static final int keyMin
```

The defID for the keyMin field = 30.

keyNbBits
```
public static final int keyNbBits
```

The defID for the keyNbBits field = 32.

keyQuant
```
public static final int keyQuant
```

The defID for the keyQuant field = 29.

normalNbBits
```
public static final int normalNbBits
```

The defID for the normalNbBits field = 34.

normalQuant
```
public static final int normalQuant
```

The defID for the normalQuant field = 33.

position2DMax
```
public static final int position2DMax
```

The defID for the position2DMax field = 7.

position2DMin
```
public static final int position2DMin
```

The defID for the position2DMin field = 6.

position2DNbBits
```
public static final int position2DNbBits
```

The defID for the position2DNbBits field = 8.

position2DQuant
```
public static final int position2DQuant
```

The defID for the position2DQuant field = 5.

position3DMax
```
public static final int position3DMax
```

The defID for the position3DMax field = 3.

position3DMin
```
public static final int position3DMin
```

The defID for the position3DMin field = 2.

position3DNbBits
```
public static final int position3DNbBits
```

The defID for the position3DNbBits field = 4.

position3DQuant
```
public static final int position3DQuant
```

The defID for the position3DQuant field = 1.

scaleMax
```
public static final int scaleMax
```

The defID for the scaleMax field = 27.

scaleMin
```
public static final int scaleMin
```

The defID for the scaleMin field = 26.

scaleNbBits

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000648

```
public static final int scaleNbBits
```

The defID for the scaleNbBits field = 28.

scaleQuant
```
public static final int scaleQuant
```

The defID for the scaleQuant field = 25.

sizeMax
```
public static final int sizeMax
```

The defID for the sizeMax field = 37.

sizeMin
```
public static final int sizeMin
```

The defID for the sizeMin field = 36.

sizeNbBits
```
public static final int sizeNbBits
```

The defID for the sizeNbBits field = 38.

sizeQuant
```
public static final int sizeQuant
```

The defID for the sizeQuant field = 35.

textureCoordinateMax
```
public static final int textureCoordinateMax
```

The defID for the textureCoordinateMax field = 19.

textureCoordinateMin
```
public static final int textureCoordinateMin
```

The defID for the textureCoordinateMin field = 18.

textureCoordinateNbBits
```
public static final int textureCoordinateNbBits
```

The defID for the textureCoordinateNbBits field = 20.

textureCoordinateQuant
```
public static final int textureCoordinateQuant
```

The defID for the textureCoordinateQuant field = 17.

useEfficientCoding
```
public static final int useEfficientCoding
```

The defID for the useEfficientCoding field = 39.

V.5.285 org.iso.mpeg.mpegj.scene.Field.Rectangle

V.5.285.1 Syntax

```
public static interface Field.Rectangle
```

V.5.285.2 Description

An interface defining constants for the field defIDs of the Rectangle node.

Member Summary	
Fields	
int	size

V.5.285.3 Fields

size
```
public static final int size
```

The defID for the size field = 0.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000649

V.5.286 org.iso.mpeg.mpegj.scene.Field.ScalarInterpolator

V.5.286.1 Syntax

```
public static interface Field.ScalarInterpolator
```

V.5.286.2 Description

An interface defining constants for the field defIDs of the ScalarInterpolator node.

Member Summary	
Fields	
int	key
int	keyValue

V.5.286.3 Fields

key
```
public static final int key
```
The defID for the key field = 1.

keyValue
```
public static final int keyValue
```
The defID for the keyValue field = 2.

V.5.287 org.iso.mpeg.mpegj.scene.Field.Script

V.5.287.1 Syntax

```
public static interface Field.Script
```

V.5.287.2 Description

An interface defining constants for the field defIDs of the Script node.

Member Summary	
Fields	
int	url
int	directOutput
int	mustEvaluate

V.5.287.3 Fields

directOutput
```
public static final int directOutput
```
The defID for the directOutput field = 1.

mustEvaluate
```
public static final int mustEvaluate
```
The defID for the mustEvaluate field = 2.

url
```
public static final int url
```
The defID for the url field = 0.

V.5.288 org.iso.mpeg.mpegj.scene.Field.Shape

V.5.288.1 Syntax

```
public static interface Field.Shape
```

V.5.288.2 Description

An interface defining constants for the field defIDs of the Shape node.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000650

Member Summary	
Fields	
int	appearance
int	geometry

V.5.288.3 Fields

appearance
`public static final int appearance`

The defID for the appearance field = 0.

geometry
`public static final int geometry`

The defID for the geometry field = 1.

V.5.289 org.iso.mpeg.mpegj.scene.Field.Sound

V.5.289.1 Syntax

```
public static interface Field.Sound
```

V.5.289.2 Description

An interface defining constants for the field defIDs of the Sound node.

Member Summary	
Fields	
int	direction
int	intensity
int	location
int	maxBack
int	maxFront
int	minBack
int	minFront
int	priority
int	source
int	spatialize

V.5.289.3 Fields

direction
`public static final int direction`

The defID for the direction field = 0.

intensity
`public static final int intensity`

The defID for the intensity field = 1.

location
`public static final int location`

The defID for the location field = 2.

maxBack
`public static final int maxBack`

The defID for the maxBack field = 3.

maxFront
`public static final int maxFront`

The defID for the maxFront field = 4.

minBack
`public static final int minBack`

The defID for the minBack field = 5.

minFront
`public static final int minFront`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000651

The defID for the minFront field = 6.

priority
`public static final int priority`

The defID for the priority field = 7.

source
`public static final int source`

The defID for the source field = 8.

spatialize
`public static final int spatialize`

The defID for the spatialize field = 9.

V.5.290 org.iso.mpeg.mpegj.scene.Field.Sound2D

V.5.290.1 Syntax

```
public static interface Field.Sound2D
```

V.5.290.2 Description

An interface defining constants for the field defIDs of the Sound2D node.

Member Summary	
Fields	
int	intensity
int	location
int	source
int	spatialize

V.5.290.3 Fields

intensity
`public static final int intensity`

The defID for the intensity field = 0.

location
`public static final int location`

The defID for the location field = 1.

source
`public static final int source`

The defID for the source field = 2.

spatialize
`public static final int spatialize`

The defID for the spatialize field = 3.

V.5.291 org.iso.mpeg.mpegj.scene.Field.Sphere

V.5.291.1 Syntax

```
public static interface Field.Sphere
```

V.5.291.2 Description

An interface defining constants for the field defIDs of the Sphere node.

Member Summary	
Fields	
int	radius

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000652

V.5.291.3 Fields

radius
`public static final int radius`

The defID for the radius field = 0.

V.5.292 org.iso.mpeg.mpegj.scene.Field.SphereSensor

V.5.292.1 Syntax

```
public static interface Field.SphereSensor
```

V.5.292.2 Description

An interface defining constants for the field defIDs of the SphereSensor node.

Member Summary	
Fields	
int	autoOffset
int	enabled
int	offset

V.5.292.3 Fields

autoOffset
`public static final int autoOffset`

The defID for the autoOffset field = 0.

enabled
`public static final int enabled`

The defID for the enabled field = 1.

offset
`public static final int offset`

The defID for the offset field = 2.

V.5.293 org.iso.mpeg.mpegj.scene.Field.SpotLight

V.5.293.1 Syntax

```
public static interface Field.SpotLight
```

V.5.293.2 Description

An interface defining constants for the field defIDs of the SpotLight node.

Member Summary	
Fields	
int	ambientIntensity
int	attenuation
int	beamWidth
int	color
int	cutOffAngle
int	direction
int	intensity
int	location
int	on
int	radius

V.5.293.3 Fields

ambientIntensity
`public static final int ambientIntensity`

The defID for the ambientIntensity field = 0.

attenuation

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000653

```
public static final int attenuation
```

The defID for the attenuation field = 1.

beamWidth
```
public static final int beamWidth
```

The defID for the beamWidth field = 2.

color
```
public static final int color
```

The defID for the color field = 3.

cutOffAngle
```
public static final int cutOffAngle
```

The defID for the cutOffAngle field = 4.

direction
```
public static final int direction
```

The defID for the direction field = 5.

intensity
```
public static final int intensity
```

The defID for the intensity field = 6.

location
```
public static final int location
```

The defID for the location field = 7.

on
```
public static final int on
```

The defID for the on field = 8.

radius
```
public static final int radius
```

The defID for the radius field = 9.

V.5.294 org.iso.mpeg.mpegj.scene.Field.Switch

V.5.294.1 Syntax

```
public static interface Field.Switch
```

V.5.294.2 Description

An interface defining constants for the field defIDs of the Switch node.

Member Summary	
Fields	
int	choice
int	whichChoice

V.5.294.3 Fields

choice
```
public static final int choice
```

The defID for the choice field = 0.

whichChoice
```
public static final int whichChoice
```

The defID for the whichChoice field = 1.

V.5.295 org.iso.mpeg.mpegj.scene.Field.TermCap

V.5.295.1 Syntax

```
public static interface Field.TermCap
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000654

V.5.295.2 Description

An interface defining constants for the field defIDs of the TermCap node.

Member Summary	
Fields	
int	capability

V.5.295.3 Fields

capability
`public static final int capability`

The defID for the capability field = 1.

V.5.296 org.iso.mpeg.mpegj.scene.Field.Text

V.5.296.1 Syntax

```
public static interface Field.Text
```

V.5.296.2 Description

An interface defining constants for the field defIDs of the Text node.

Member Summary	
Fields	
int	string
int	length
int	fontStyle
int	maxExtent

V.5.296.3 Fields

fontStyle
`public static final int fontStyle`

The defID for the fontStyle field = 2.

length
`public static final int length`

The defID for the length field = 1.

maxExtent
`public static final int maxExtent`

The defID for the maxExtent field = 3.

string
`public static final int string`

The defID for the string field = 0.

V.5.297 org.iso.mpeg.mpegj.scene.Field.TextureCoordinate

V.5.297.1 Syntax

```
public static interface Field.TextureCoordinate
```

V.5.297.2 Description

An interface defining constants for the field defIDs of the TextureCoordinate node.

Member Summary	
Fields	
int	point

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000655

V.5.297.3 Fields

point
`public static final int point`

The defID for the point field = 0.

V.5.298 org.iso.mpeg.mpegj.scene.Field.TextureTransform

V.5.298.1 Syntax

```
public static interface Field.TextureTransform
```

V.5.298.2 Description

An interface defining constants for the field defIDs of the TextureTransform node.

Member Summary	
Fields	
int	center
int	rotation
int	scale
int	translation

V.5.298.3 Fields

center
`public static final int center`

The defID for the center field = 0.

rotation
`public static final int rotation`

The defID for the rotation field = 1.

scale
`public static final int scale`

The defID for the scale field = 2.

translation
`public static final int translation`

The defID for the translation field = 3.

V.5.299 org.iso.mpeg.mpegj.scene.Field.TimeSensor

V.5.299.1 Syntax

```
public static interface Field.TimeSensor
```

V.5.299.2 Description

An interface defining constants for the field defIDs of the TimeSensor node.

Member Summary	
Fields	
int	cycleInterval
int	enabled
int	loop
int	startTime
int	stopTime

V.5.299.3 Fields

cycleInterval
`public static final int cycleInterval`

The defID for the cycleInterval field = 0.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000656

enabled
`public static final int enabled`

The defID for the enabled field = 1.

loop
`public static final int loop`

The defID for the loop field = 2.

startTime
`public static final int startTime`

The defID for the startTime field = 3.

stopTime
`public static final int stopTime`

The defID for the stopTime field = 4.

V.5.300 org.iso.mpeg.mpegj.scene.Field.TouchSensor

V.5.300.1 Syntax

```
public static interface Field.TouchSensor
```

V.5.300.2 Description

An interface defining constants for the field defIDs of the TouchSensor node.

Member Summary	
Fields	
int	enabled

V.5.300.3 Fields

enabled
`public static final int enabled`

The defID for the enabled field = 0.

V.5.301 org.iso.mpeg.mpegj.scene.Field.Transform

V.5.301.1 Syntax

```
public static interface Field.Transform
```

V.5.301.2 Description

An interface defining constants for the field defIDs of the Transform node.

Member Summary	
Fields	
int	center
int	children
int	rotation
int	scale
int	scaleOrientation
int	translation

V.5.301.3 Fields

center
`public static final int center`

The defID for the center field = 2.

children
`public static final int children`

The defID for the children field = 3.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000657

rotation
`public static final int rotation`

The defID for the rotation field = 4.

scale
`public static final int scale`

The defID for the scale field = 5.

scaleOrientation
`public static final int scaleOrientation`

The defID for the scaleOrientation field = 6.

translation
`public static final int translation`

The defID for the translation field = 7.

V.5.302 org.iso.mpeg.mpegj.scene.Field.Transform2D

V.5.302.1 Syntax

```
public static interface Field.Transform2D
```

V.5.302.2 Description

An interface defining constants for the field defIDs of the Transform2D node.

Member Summary	
Fields	
int	children
int	center
int	rotationAngle
int	scale
int	scaleOrientation
int	translation

V.5.302.3 Fields

center
`public static final int center`

The defID for the center field = 3.

children
`public static final int children`

The defID for the children field = 2.

rotationAngle
`public static final int rotationAngle`

The defID for the rotationAngle field = 4.

scale
`public static final int scale`

The defID for the scale field = 5.

scaleOrientation
`public static final int scaleOrientation`

The defID for the scaleOrientation field = 6.

translation
`public static final int translation`

The defID for the translation field = 7.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000658

V.5.303 org.iso.mpeg.mpegj.scene.Field.Valuator

V.5.303.1 Syntax

```
public static interface Field.Valuator
```

V.5.303.2 Description

An interface defining constants for the field defIDs of the Valuator node.

Member Summary	
Fields	
int	Factor1
int	Factor2
int	Factor3
int	Factor4
int	Offset1
int	Offset2
int	Offset3
int	viseme_def

V.5.303.3 Fields

Factor1
```
public static final int Factor1
```
The defID for the Factor1 field = 32.

Factor2
```
public static final int Factor2
```
The defID for the Factor2 field = 33.

Factor3
```
public static final int Factor3
```
The defID for the Factor3 field = 34.

Factor4
```
public static final int Factor4
```
The defID for the Factor4 field = 35.

Offset1
```
public static final int Offset1
```
The defID for the Offset1 field = 36.

Offset2
```
public static final int Offset2
```
The defID for the Offset2 field = 37.

Offset3
```
public static final int Offset3
```
The defID for the Offset3 field = 38.

viseme_def
```
public static final int viseme_def
```
The defID for the viseme_def field = 3.

V.5.304 org.iso.mpeg.mpegj.scene.Field.WorldInfo

V.5.304.1 Syntax

```
public static interface Field.WorldInfo
```

V.5.304.2 Description

An interface defining constants for the field defIDs of the WorldInfo node.

Member Summary	

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000659

Fields	
int	info
int	title

V.5.304.3 Fields

info
`public static final int info`

The defID for the info field = 0.

title
`public static final int title`

The defID for the title field = 1.

V.5.305 org.iso.mpeg.mpegj.scene.FieldValue

V.5.305.1 Syntax

```
public interface FieldValue
```

All Known Subinterfaces:
org.iso.mpeg.mpegj.scene.MFColorFieldValue, org.iso.mpeg.mpegj.scene.MFFieldValue, org.iso.mpeg.mpegj.scene.MFFloatFieldValue, org.iso.mpeg.mpegj.scene.MFInt32FieldValue, org.iso.mpeg.mpegj.scene.MFNodeFieldValue, org.iso.mpeg.mpegj.scene.MFRotationFieldValue, org.iso.mpeg.mpegj.scene.MFStringFieldValue, org.iso.mpeg.mpegj.scene.MFTimeFieldValue, org.iso.mpeg.mpegj.scene.MFVec2fFieldValue, org.iso.mpeg.mpegj.scene.MFVec3fFieldValue, org.iso.mpeg.mpegj.scene.SFBoolFieldValue, org.iso.mpeg.mpegj.scene.SFColorFieldValue, org.iso.mpeg.mpegj.scene.SFFloatFieldValue, org.iso.mpeg.mpegj.scene.SFInt32FieldValue, org.iso.mpeg.mpegj.scene.SFNodeFieldValue, org.iso.mpeg.mpegj.scene.SFRotationFieldValue, org.iso.mpeg.mpegj.scene.SFStringFieldValue, org.iso.mpeg.mpegj.scene.SFTimeFieldValue, org.iso.mpeg.mpegj.scene.SFVec2fFieldValue, org.iso.mpeg.mpegj.scene.SFVec3fFieldValue

V.5.305.2 Description

A tagging interface used to identify objects that can return field values.

V.5.306 org.iso.mpeg.mpegj.scene.InvalidNodeException

V.5.306.1 Syntax

```
public class InvalidNodeException extends org.iso.mpeg.mpegj.MPEGJException
```

java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--**org.iso.mpeg.mpegj.scene.InvalidNodeException**

All Implemented Interfaces:
java.io.Serializable

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000660

V.5.306.2 Description

An exception used to indicate that an operation was attempted on a node that is no longer valid.

Member Summary	
Constructors	`InvalidNodeException()` `InvalidNodeException(String)`

V.5.306.3 Constructors

InvalidNodeException()
`public InvalidNodeException()`

Constructs an InvalidNodeException with no specified detail message.

InvalidNodeException(String)
`public InvalidNodeException(java.lang.String)`

Constructs an InvalidNodeException with a detailed message.

Parameters:
`message` - the detail message.

V.5.307 org.iso.mpeg.mpegj.scene.InvalidSceneException

V.5.307.1 Syntax

`public class InvalidSceneException extends org.iso.mpeg.mpegj.MPEGJException`

java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--org.iso.mpeg.mpegj.MPEGJException
             |
             +--**org.iso.mpeg.mpegj.scene.InvalidSceneException**

All Implemented Interfaces:
java.io.Serializable

V.5.307.2 Description

An exception used by this package to indicate that an operation was attempted on a Scene that is no longer valid.

V.5.308 org.iso.mpeg.mpegj.scene.MFColorFieldValue

V.5.308.1 Syntax

`public interface MFColorFieldValue extends org.iso.mpeg.mpegj.scene.MFFieldValue`

All Superinterfaces:
`org.iso.mpeg.mpegj.scene.FieldValue`, `org.iso.mpeg.mpegj.scene.MFFieldValue`

V.5.308.2 Description

An interface used for obtaining MFColor values.

Member Summary	
Methods	
`SFColorFieldValue[]`	`getMFColorValue()`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000661

V.5.308.3 Methods

getMFColorValue()
`public org.iso.mpeg.mpegj.scene.SFColorFieldValue getMFColorValue()`

Obtain the MFColor value.

Returns:
an array of SFColorFieldValue objects.

V.5.309 org.iso.mpeg.mpegj.scene.MFFieldValue

V.5.309.1 Syntax

```
public interface MFFieldValue extends org.iso.mpeg.mpegj.scene.FieldValue
```

All Known Subinterfaces:
org.iso.mpeg.mpegj.scene.MFColorFieldValue, org.iso.mpeg.mpegj.scene.MFFloatFieldValue, org.iso.mpeg.mpegj.scene.MFInt32FieldValue, org.iso.mpeg.mpegj.scene.MFNodeFieldValue, org.iso.mpeg.mpegj.scene.MFRotationFieldValue, org.iso.mpeg.mpegj.scene.MFStringFieldValue, org.iso.mpeg.mpegj.scene.MFTimeFieldValue, org.iso.mpeg.mpegj.scene.MFVec2fFieldValue, org.iso.mpeg.mpegj.scene.MFVec3fFieldValue

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue

V.5.309.2 Description

A tagging interface used to classify FieldValue objects that return compound values.

V.5.310 org.iso.mpeg.mpegj.scene.MFFloatFieldValue

V.5.310.1 Syntax

```
public interface MFFloatFieldValue extends org.iso.mpeg.mpegj.scene.MFFieldValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue, org.iso.mpeg.mpegj.scene.MFFieldValue

V.5.310.2 Description

An interface used for obtaining MFFloat values.

Member Summary	
Methods	
SFFloatFieldValue[]	getMFFloatValue()

V.5.310.3 Methods

getMFFloatValue()
`public org.iso.mpeg.mpegj.scene.SFFloatFieldValue getMFFloatValue()`

Obtain the MFFloat value.

Returns:
an array of MFFloatFieldValue objects.

V.5.311 org.iso.mpeg.mpegj.scene.MFInt32FieldValue

V.5.311.1 Syntax

```
public interface MFInt32FieldValue extends org.iso.mpeg.mpegj.scene.MFFieldValue
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000662

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue, org.iso.mpeg.mpegj.scene.MFFieldValue

V.5.311.2 Description

An interface used for obtaining MFInt32 values.

Member Summary	
Methods	
SFInt32FieldValue[]	getMFInt32Value()

V.5.311.3 Methods

getMFInt32Value()
`public org.iso.mpeg.mpegj.scene.SFInt32FieldValue getMFInt32Value()`

Obtain the SFInt32 value.

Returns:
an array of SFInt32Value objects.

V.5.312 org.iso.mpeg.mpegj.scene.MFNodeFieldValue

V.5.312.1 Syntax

```
public interface MFNodeFieldValue extends org.iso.mpeg.mpegj.scene.MFFieldValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue, org.iso.mpeg.mpegj.scene.MFFieldValue

V.5.312.2 Description

An interface used for obtaining MFNode values.

Member Summary	
Methods	
SFNodeFieldValue[]	getMFNodeValue()

V.5.312.3 Methods

getMFNodeValue()
`public org.iso.mpeg.mpegj.scene.SFNodeFieldValue getMFNodeValue()`

Obtain the MFNode value.

Returns:
an array of SFNodeFieldValue objects.

V.5.313 org.iso.mpeg.mpegj.scene.MFRotationFieldValue

V.5.313.1 Syntax

```
public interface MFRotationFieldValue extends org.iso.mpeg.mpegj.scene.MFFieldValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue, org.iso.mpeg.mpegj.scene.MFFieldValue

V.5.313.2 Description

An interface used for obtaining MFRotation values.

Member Summary	
Methods	
SFRotationFieldValue[]	getMFRotationValue()

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000663

V.5.313.3 Methods

getMFRotationValue()
`public org.iso.mpeg.mpegj.scene.SFRotationFieldValue getMFRotationValue()`

Obtain the MFRotation value.

Returns:
an array of MFRotationFieldValue objects.

V.5.314 org.iso.mpeg.mpegj.scene.MFStringFieldValue

V.5.314.1 Syntax

```
public interface MFStringFieldValue extends org.iso.mpeg.mpegj.scene.MFFieldValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue, org.iso.mpeg.mpegj.scene.MFFieldValue

V.5.314.2 Description

An interface used for obtaining MFString values.

Member Summary	
Methods	
SFStringFieldValue[]	getMFStringValue()

V.5.314.3 Methods

getMFStringValue()
`public org.iso.mpeg.mpegj.scene.SFStringFieldValue getMFStringValue()`

Obtain the MFString value.

Returns:
an array of SFStringFieldValue objects.

V.5.315 org.iso.mpeg.mpegj.scene.MFTimeFieldValue

V.5.315.1 Syntax

```
public interface MFTimeFieldValue extends org.iso.mpeg.mpegj.scene.MFFieldValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue, org.iso.mpeg.mpegj.scene.MFFieldValue

V.5.315.2 Description

An interface used for obtaining MFTime values.

Member Summary	
Methods	
SFTimeFieldValue[]	getMFTimeValue()

V.5.315.3 Methods

getMFTimeValue()
`public org.iso.mpeg.mpegj.scene.SFTimeFieldValue getMFTimeValue()`

Obtain the MFTime value.

Returns:
an array of SFTimeFieldValue objects.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000664

V.5.316 org.iso.mpeg.mpegj.scene.MFVec2fFieldValue

V.5.316.1 Syntax

```
public interface MFVec2fFieldValue extends org.iso.mpeg.mpegj.scene.MFFieldValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue, org.iso.mpeg.mpegj.scene.MFFieldValue

V.5.316.2 Description

An interface used for obtaining MFVec2f values.

Member Summary	
Methods	
SFVec2fFieldValue[]	getMFVec2fValue()

V.5.316.3 Methods

getMFVec2fValue()
public org.iso.mpeg.mpegj.scene.SFVec2fFieldValue getMFVec2fValue()

Obtain the MFVec2f value.

Returns:
an array of SFVec2fFieldValue objects.

V.5.317 org.iso.mpeg.mpegj.scene.MFVec3fFieldValue

V.5.317.1 Syntax

```
public interface MFVec3fFieldValue extends org.iso.mpeg.mpegj.scene.MFFieldValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue, org.iso.mpeg.mpegj.scene.MFFieldValue

V.5.317.2 Description

An interface used for obtaining MFVec3f values.

Member Summary	
Methods	
SFVec3fFieldValue[]	getMFVec3fValue()

V.5.317.3 Methods

getMFVec3fValue()
public org.iso.mpeg.mpegj.scene.SFVec3fFieldValue getMFVec3fValue()

Obtain the MFVec3f value.

Returns:
an array of SFVec3fFieldValue objects.

V.5.318 org.iso.mpeg.mpegj.scene.NewNode

V.5.318.1 Syntax

```
public interface NewNode extends org.iso.mpeg.mpegj.scene.NodeValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.NodeValue

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000665

V.5.318.2 Description

An interface used to specify the creation of a new node.

Member Summary	
Methods	
int	getNodeType()
int	getNodeID()
FieldValue	getField(int)

V.5.318.3 Methods

getField(int)
`public org.iso.mpeg.mpegj.scene.FieldValue getField(int)`

Obtain the value of a field in the new node identified by the given defID, or null if the field has the default value. For example, the defID argument may be Field.Transform.translation, in which case the object returned shall implement the SFVec3fFieldValue interface.

Parameters:
`defID` - the field defID.

Returns:
the field value, or null for the default value.

Throws:
`org.iso.mpeg.mpegj.scene.BadParameterException` - if the defID is not a valid field defID.

See Also:
`org.iso.mpeg.mpegj.scene.Field`

getNodeID()
`public int getNodeID()`

Obtain the DEF identifier of the new node, or zero if the node does not have a DEF identifier.

Returns:
the DEF identifier, or zero if none.

getNodeType()
`public int getNodeType()`

Obtain the node type of the new node. For example, this method may return NodeType.Transform.

Returns:
the integer node type.

See Also:
`org.iso.mpeg.mpegj.scene.NodeType`

V.5.319 org.iso.mpeg.mpegj.scene.Node

V.5.319.1 Syntax

```
public interface Node
```

V.5.319.2 Description

An interface that acts as a proxy for a BIFS node in the scene graph. Only nodes in the scene that have been instanced with DEF may have a Node proxy.

Member Summary	
Methods	
void	sendEventIn(int, FieldValue)
FieldValue	getEventOut(int)
void	addEventOutListener(int, EventOutListener)
void	removeEventOutListener(int, EventOutListener)
int	getNodeType()

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000666

V.5.319.3 Methods

addEventOutListener(int, EventOutListener)
`public void addEventOutListener(int, org.iso.mpeg.mpegj.scene.EventOutListener)`

Install a listener for an eventOut or exposedField in this node of the scene graph.

Parameters:
`outID` - The outID of the field to be monitored.

`listener` - The listener to notify of changes.

Throws:
`org.iso.mpeg.mpegj.scene.BadParameterException` - Thrown if the selected field ID is not valid for this node.

`org.iso.mpeg.mpegj.scene.InvalidNodeException` - Thrown if the node is no longer valid (due to removal or replacement).

See Also:
`org.iso.mpeg.mpegj.scene.EventOut`, `org.iso.mpeg.mpegj.scene.EventOutListener`

getEventOut(int)
`public org.iso.mpeg.mpegj.scene.FieldValue getEventOut(int)`

Reads the current value of an eventOut or exposedField in this node of the scene graph.

Parameters:
`outID` - The outID of the field to be read.

Returns:
a FieldValue containing the value read from the desired field.

Throws:
`org.iso.mpeg.mpegj.scene.BadParameterException` - Thrown if the selected field ID is not valid for this node.

`org.iso.mpeg.mpegj.scene.InvalidNodeException` - Thrown if the node is no longer valid (due to removal or replacement).

See Also:
`org.iso.mpeg.mpegj.scene.EventOut`

getNodeType()
`public int getNodeType()`

Obtain the node type of the node.

Returns:
an int indicating the node type.

Throws:
`org.iso.mpeg.mpegj.scene.InvalidNodeException` - Thrown if the node is no longer valid.

See Also:
`org.iso.mpeg.mpegj.scene.NodeType`

removeEventOutListener(int, EventOutListener)
`public void removeEventOutListener(int, org.iso.mpeg.mpegj.scene.EventOutListener)`

Remove a listener previously added for an eventOut or exposedField in this node of the scene graph.

Parameters:
`outID` - The outID of the field no longer to be monitored.

`listener` - The listener that is to be removed.

Throws:
`org.iso.mpeg.mpegj.scene.BadParameterException` - Thrown if the selected field ID is not valid for this node.

`org.iso.mpeg.mpegj.scene.InvalidNodeException` - Thrown if the node is no longer valid (due to removal or replacement).

See Also:
`org.iso.mpeg.mpegj.scene.EventOut`, `org.iso.mpeg.mpegj.scene.EventOutListener`

sendEventIn(int, FieldValue)
`public void sendEventIn(int, org.iso.mpeg.mpegj.scene.FieldValue)`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000667

Updates the value of an eventIn or exposedField in this node of the scene graph. This is a synchronous call that will not return until the field is updated in the scene.

Parameters:
`inID` - The inID of the field to be updated.

`newValue` - The desired new value for the field.

Throws:
`org.iso.mpeg.mpegj.scene.BadParameterException` - Thrown if the selected field ID is not valid for this node, or if the value is not an appropriate type for the field.

`org.iso.mpeg.mpegj.scene.InvalidNodeException` - Thrown if the node is no longer valid (due to removal or replacement).

See Also:
`org.iso.mpeg.mpegj.scene.EventIn`

V.5.320 org.iso.mpeg.mpegj.scene.NodeType

V.5.320.1 Syntax

```
public interface NodeType
```

V.5.320.2 Description

An interface defining constants for node types.

Member Summary	
Fields	
int	Anchor
int	AnimationStream
int	Appearance
int	AudioBuffer
int	AudioClip
int	AudioDelay
int	AudioFX
int	AudioMix
int	AudioSource
int	AudioSwitch
int	Background
int	Background2D
int	Billboard
int	Bitmap
int	Box
int	Circle
int	Collision
int	Color
int	ColorInterpolator
int	CompositeTexture2D
int	CompositeTexture3D
int	Conditional
int	Cone
int	Coordinate
int	Coordinate2D
int	CoordinateInterpolator
int	CoordinateInterpolator2D
int	Curve2D
int	Cylinder
int	CylinderSensor
int	DirectionalLight
int	DiscSensor
int	ElevationGrid
int	Expression
int	Extrusion
int	Face
int	FaceDefMesh
int	FaceDefTables
int	FaceDefTransform
int	FAP
int	FDP

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000668

int	FIT
int	Fog
int	FontStyle
int	Form
int	Group
int	ImageTexture
int	IndexedFaceSet
int	IndexedFaceSet2D
int	IndexedLineSet
int	IndexedLineSet2D
int	Inline
int	LOD
int	Layer2D
int	Layer3D
int	Layout
int	LineProperties
int	ListeningPoint
int	Material
int	Material2D
int	MovieTexture
int	NavigationInfo
int	Normal
int	NormalInterpolator
int	OrderedGroup
int	OrientationInterpolator
int	PixelTexture
int	PlaneSensor
int	PlaneSensor2D
int	PointLight
int	PointSet
int	PointSet2D
int	PositionInterpolator
int	PositionInterpolator2D
int	ProximitySensor2D
int	ProximitySensor
int	QuantizationParameter
int	Rectangle
int	ScalarInterpolator
int	Script
int	Shape
int	Sound
int	Sound2D
int	Sphere
int	SphereSensor
int	SpotLight
int	Switch
int	TermCap
int	Text
int	TextureCoordinate
int	TextureTransform
int	TimeSensor
int	TouchSensor
int	Transform
int	Transform2D
int	Valuator
int	Viewpoint
int	VisibilitySensor
int	Viseme
int	WorldInfo

V.5.320.3 Fields

Anchor
`public static final int Anchor`

Anchor node = 1.

AnimationStream
`public static final int AnimationStream`

AnimationStream node = 2.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000669

Appearance
`public static final int Appearance`

Appearance node = 3.

AudioBuffer
`public static final int AudioBuffer`

AudioBuffer node = 4.

AudioClip
`public static final int AudioClip`

AudioClip node = 5.

AudioDelay
`public static final int AudioDelay`

AudioDelay node = 6.

AudioFX
`public static final int AudioFX`

AudioFX node = 7.

AudioMix
`public static final int AudioMix`

AudioMix node = 8 .

AudioSource
`public static final int AudioSource`

AudioSource node = 9 .

AudioSwitch
`public static final int AudioSwitch`

AudioSwitch node = 10.

Background
`public static final int Background`

Background node = 11.

Background2D
`public static final int Background2D`

Background2D node = 12.

Billboard
`public static final int Billboard`

Billboard node = 13.

Bitmap
`public static final int Bitmap`

Bitmap node = 14.

Box
`public static final int Box`

Box node = 15.

Circle
`public static final int Circle`

Circle node = 16.

Collision
`public static final int Collision`

Collision node = 17.

Color
`public static final int Color`

Color node = 18.

ColorInterpolator
`public static final int ColorInterpolator`

ColorInterpolator node = 19.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000670

CompositeTexture2D
`public static final int CompositeTexture2D`

CompositeTexture2D node = 20.

CompositeTexture3D
`public static final int CompositeTexture3D`

CompositeTexture3D node = 21.

Conditional
`public static final int Conditional`

Conditional node = 22.

Cone
`public static final int Cone`

Cone node = 23.

Coordinate
`public static final int Coordinate`

Coordinate node = 24.

Coordinate2D
`public static final int Coordinate2D`

Coordinate2D node = 25.

CoordinateInterpolator
`public static final int CoordinateInterpolator`

CoordinateInterpolator node = 26.

CoordinateInterpolator2D
`public static final int CoordinateInterpolator2D`

CoordinateInterpolator2D node = 27.

Curve2D
`public static final int Curve2D`

Curve2D node = 28.

Cylinder
`public static final int Cylinder`

Cylinder node = 29.

CylinderSensor
`public static final int CylinderSensor`

CylinderSensor node = 30.

DirectionalLight
`public static final int DirectionalLight`

DirectionalLight node = 31.

DiscSensor
`public static final int DiscSensor`

DiscSensor node = 32.

ElevationGrid
`public static final int ElevationGrid`

ElevationGrid node = 33.

Expression
`public static final int Expression`

Expression node = 34.

Extrusion
`public static final int Extrusion`

Extrusion node = 35.

Face
`public static final int Face`

Face node = 36.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000671

FaceDefMesh
`public static final int FaceDefMesh`

FaceDefMesh node = 37.

FaceDefTables
`public static final int FaceDefTables`

FaceDefTables node = 38.

FaceDefTransform
`public static final int FaceDefTransform`

FaceDefTransform node = 39.

FAP
`public static final int FAP`

FAP node = 40.

FDP
`public static final int FDP`

FDP node = 41.

FIT
`public static final int FIT`

FIT node = 42.

Fog
`public static final int Fog`

Fog node = 43.

FontStyle
`public static final int FontStyle`

FontStyle node = 44.

Form
`public static final int Form`

Form node = 45.

Group
`public static final int Group`

Group node = 46.

ImageTexture
`public static final int ImageTexture`

ImageTexture node = 47.

IndexedFaceSet
`public static final int IndexedFaceSet`

IndexedFaceSet node = 48.

IndexedFaceSet2D
`public static final int IndexedFaceSet2D`

IndexedFaceSet2D node = 49.

IndexedLineSet
`public static final int IndexedLineSet`

IndexedLineSet node = 50.

IndexedLineSet2D
`public static final int IndexedLineSet2D`

IndexedLineSet2D node = 51.

Inline
`public static final int Inline`

Inline node = 52.

Layer2D
`public static final int Layer2D`

Layer2D node = 54.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000672

Layer3D
`public static final int Layer3D`

Layer3D node = 55.

Layout
`public static final int Layout`

Layout node = 56.

LineProperties
`public static final int LineProperties`

LineProperties node = 57.

ListeningPoint
`public static final int ListeningPoint`

ListeningPoint node = 58.

LOD
`public static final int LOD`

LOD node = 53.

Material
`public static final int Material`

Material node = 59.

Material2D
`public static final int Material2D`

Material2D node = 60.

MovieTexture
`public static final int MovieTexture`

MovieTexture node = 61.

NavigationInfo
`public static final int NavigationInfo`

NavigationInfo node = 62.

Normal
`public static final int Normal`

Normal node = 63.

NormalInterpolator
`public static final int NormalInterpolator`

NormalInterpolator node = 64.

OrderedGroup
`public static final int OrderedGroup`

OrderedGroup node = 65.

OrientationInterpolator
`public static final int OrientationInterpolator`

OrientationInterpolator node = 66.

PixelTexture
`public static final int PixelTexture`

PixelTexture node = 67.

PlaneSensor
`public static final int PlaneSensor`

PlaneSensor node = 68.

PlaneSensor2D
`public static final int PlaneSensor2D`

PlaneSensor2D node = 69.

PointLight
`public static final int PointLight`

PointLight node = 70.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000673

PointSet
`public static final int PointSet`

PointSet node = 71.

PointSet2D
`public static final int PointSet2D`

PointSet2D node = 72.

PositionInterpolator
`public static final int PositionInterpolator`

PositionInterpolator node = 73.

PositionInterpolator2D
`public static final int PositionInterpolator2D`

PositionInterpolator2D node = 74.

ProximitySensor
`public static final int ProximitySensor`

ProximitySensor node = 76.

ProximitySensor2D
`public static final int ProximitySensor2D`

ProximitySensor2D node = 75.

QuantizationParameter
`public static final int QuantizationParameter`

QuantizationParameter node = 77.

Rectangle
`public static final int Rectangle`

Rectangle node = 78.

ScalarInterpolator
`public static final int ScalarInterpolator`

ScalarInterpolator node = 79.

Script
`public static final int Script`

Script node = 80.

Shape
`public static final int Shape`

Shape node = 81.

Sound
`public static final int Sound`

Sound node = 82.

Sound2D
`public static final int Sound2D`

Sound2D node = 83.

Sphere
`public static final int Sphere`

Sphere node = 84.

SphereSensor
`public static final int SphereSensor`

SphereSensor node = 85.

SpotLight
`public static final int SpotLight`

SpotLight node = 86.

Switch
`public static final int Switch`

Switch node = 87.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000674

TermCap
`public static final int TermCap`

TermCap node = 88.

Text
`public static final int Text`

Text node = 89.

TextureCoordinate
`public static final int TextureCoordinate`

TextureCoordinate node = 90.

TextureTransform
`public static final int TextureTransform`

TextureTransform node = 91.

TimeSensor
`public static final int TimeSensor`

TimeSensor node = 92.

TouchSensor
`public static final int TouchSensor`

TouchSensor node = 93.

Transform
`public static final int Transform`

Transform node = 94.

Transform2D
`public static final int Transform2D`

Transform2D node = 95.

Valuator
`public static final int Valuator`

Valuator node = 96.

Viewpoint
`public static final int Viewpoint`

Viewpoint node = 97.

Viseme
`public static final int Viseme`

Viseme node = 99.

VisibilitySensor
`public static final int VisibilitySensor`

VisibilitySensor node = 98.

WorldInfo
`public static final int WorldInfo`

WorldInfo node = 100.

V.5.321 org.iso.mpeg.mpegj.scene.NodeValue

V.5.321.1 Syntax

```
public interface NodeValue
```

All Known Subinterfaces:
`org.iso.mpeg.mpegj.scene.NewNode`

V.5.321.2 Description

An interface for node values that either act as a proxy for a node in the BIFS scene, or specify the creation of a new node.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000675

V.5.322 org.iso.mpeg.mpegj.scene.Scene

V.5.322.1 Syntax

```
public interface Scene
```

V.5.322.2 Description

An interface that acts as a proxy for a BIFS scene.

Member Summary	
Methods	
Node	getNode(int)

V.5.322.3 Methods

getNode(int)
`public org.iso.mpeg.mpegj.scene.Node getNode(int)`

Returns an instance of a Node proxy for an instanced node in the scene.

Parameters:
`defID` - The id of the desired node. This node must have been instanced using DEF.

Returns:
a Node proxy for the desired node in the scene.

Throws:
`org.iso.mpeg.mpegj.scene.BadParameterException` - Thrown if no node exists with the specified id.

`org.iso.mpeg.mpegj.scene.InvalidSceneException` - Thrown if the Scene is no longer valid (due to removal or replacement).

V.5.323 org.iso.mpeg.mpegj.scene.SceneListener

V.5.323.1 Syntax

```
public interface SceneListener
```

V.5.323.2 Description

An interface that allows monitoring of changes to a BIFS scene graph.

Member Summary	
Inner Classes	
static interface	SceneListener.Message
Methods	
void	notify(int, Scene)

V.5.323.3 Methods

notify(int, Scene)
`public void notify(int, org.iso.mpeg.mpegj.scene.Scene)`

Called by the scene manager when the BIFS scene has been changed.

Parameters:
`whatHappened` - contains the message number that indicates the change.

`newScene` - contains the new Scene object when appropriate.

V.5.324 org.iso.mpeg.mpegj.scene.SceneListener.Message

V.5.324.1 Syntax

```
public static interface SceneListener.Message
```


© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000676

V.5.324.2 Description

The message numbers used in the notify method.

Member Summary	
Fields	
int	SCENE_READY
int	SCENE_REPLACED
int	SCENE_REMOVED

V.5.324.3 Fields

SCENE_READY
`public static final int SCENE_READY`

When the listener is first added this message is sent with the current valid Scene instance. SCENE_READY = 1;

SCENE_REMOVED
`public static final int SCENE_REMOVED`

Indicates that the scene has been removed. SCENE_REMOVED = 3;

SCENE_REPLACED
`public static final int SCENE_REPLACED`

Indicates that the scene has been replaced. SCENE_REPLACED = 2;

V.5.325 org.iso.mpeg.mpegj.scene.SceneManager

V.5.325.1 Syntax

```
public interface SceneManager
```

V.5.325.2 Description

An interface that allows access to the MPEG-4 terminal's native scene. An instance is obtained using getSceneManager() in the MPEGJTerminal class.

Note that scene graph access requires first obtaining a Scene instance. The only normative way to get a Scene instance is through a notification on a SceneListener instance.

See Also:
`org.iso.mpeg.mpegj.MpegjTerminal`, `org.iso.mpeg.mpegj.scene.SceneListener`, `org.iso.mpeg.mpegj.scene.Scene`

Member Summary	
Methods	
void	addSceneListener(SceneListener)
void	removeSceneListener(SceneListener)

V.5.325.3 Methods

addSceneListener(SceneListener)
`public void addSceneListener(org.iso.mpeg.mpegj.scene.SceneListener)`

Add a SceneListener to this scene manager. The listener will be informed of important changes to the scene, and will be given the appropriate Scene instance that will allow modification to nodes and fields of the scene. The base specification does not limit the number of SceneListeners that may be registered.

Parameters:
`listener` - the SceneListener instance to add.

removeSceneListener(SceneListener)
`public void removeSceneListener(org.iso.mpeg.mpegj.scene.SceneListener)`

Remove a SceneListener from the scene manager.

Parameters:
`listener` - the SceneListener instance to remove.

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000677

Throws:
org.iso.mpeg.mpegj.scene.BadParameterException - Thrown if the specified listener was not found on the scene manager's list of listeners.

V.5.326 org.iso.mpeg.mpegj.scene.SFBoolFieldValue

V.5.326.1 Syntax

```
public interface SFBoolFieldValue extends org.iso.mpeg.mpegj.scene.FieldValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue

V.5.326.2 Description

An interface used for obtaining SFBool values.

Member Summary	
Methods boolean	getSFBoolValue()

V.5.326.3 Methods

getSFBoolValue()
```
public boolean getSFBoolValue()
```
Obtain the SFBool value.

Returns:
a boolean that contains the SFBool value.

V.5.327 org.iso.mpeg.mpegj.scene.SFColorFieldValue

V.5.327.1 Syntax

```
public interface SFColorFieldValue extends org.iso.mpeg.mpegj.scene.FieldValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue

V.5.327.2 Description

An interface used for obtaining SFColor values.

Member Summary	
Methods float[]	getSFColorValue()

V.5.327.3 Methods

getSFColorValue()
```
public float[] getSFColorValue()
```
Obtain the SFColor value.

Returns:
an array of three floats ordered as red, green, blue.

V.5.328 org.iso.mpeg.mpegj.scene.SFFloatFieldValue

V.5.328.1 Syntax

```
public interface SFFloatFieldValue extends org.iso.mpeg.mpegj.scene.FieldValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000678

V.5.328.2 Description

An interface used for obtaining SFFloat values.

Member Summary	
Methods	
float	getSFFloatValue()

V.5.328.3 Methods

getSFFloatValue()
`public float getSFFloatValue()`

Obtain the SFFloat value.

Returns:
a float value of the SFFloat field.

V.5.329 org.iso.mpeg.mpegj.scene.SFInt32FieldValue

V.5.329.1 Syntax

```
public interface SFInt32FieldValue extends org.iso.mpeg.mpegj.scene.FieldValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue

V.5.329.2 Description

An interface used for obtaining SFInt32 values.

Member Summary	
Methods	
int	getSFInt32Value()

V.5.329.3 Methods

getSFInt32Value()
`public int getSFInt32Value()`

Obtain the SFInt32 value.

Returns:
a int value of the SFInt32 field.

V.5.330 org.iso.mpeg.mpegj.scene.SFNodeFieldValue

V.5.330.1 Syntax

```
public interface SFNodeFieldValue extends org.iso.mpeg.mpegj.scene.FieldValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue

V.5.330.2 Description

An interface used for obtaining SFNode values.

Member Summary	
Methods	
NodeValue	getSFNodeValue()

V.5.330.3 Methods

getSFNodeValue()
`public org.iso.mpeg.mpegj.scene.NodeValue getSFNodeValue()`

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000679

Obtain the SFNode value. If this object was obtained by the getEventOut method of the Node interface, then the return value shall contain a proxy for a node in the BIFS scene. If this object has been passed to the sendEventIn method of the Node interface, then the return value shall contain either a proxy or an object implementing the NewNode interface to specify how to create a new node.

Returns:
a NodeValue object either acting as a proxy for a node in the BIFS scene or specifying how to create a new node.

V.5.331 org.iso.mpeg.mpegj.scene.SFRotationFieldValue

V.5.331.1 Syntax

```
public interface SFRotationFieldValue extends org.iso.mpeg.mpegj.scene.FieldValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue

V.5.331.2 Description

An interface used for obtaining SFRotation values.

Member Summary	
Methods	
float[]	getSFRotationValue()

V.5.331.3 Methods

getSFRotationValue()
```
public float[] getSFRotationValue()
```
Obtain the SFRotation value.

Returns:
a float array ordered as (x,y,z,angle) with VRML-97 semantics.

V.5.332 org.iso.mpeg.mpegj.scene.SFStringFieldValue

V.5.332.1 Syntax

```
public interface SFStringFieldValue extends org.iso.mpeg.mpegj.scene.FieldValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue

V.5.332.2 Description

An interface used for obtaining SFString values.

Member Summary	
Methods	
String	getSFStringValue()

V.5.332.3 Methods

getSFStringValue()
```
public java.lang.String getSFStringValue()
```
Obtain the SFString value.

Returns:
a String value of the SFString field.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000680

V.5.333 org.iso.mpeg.mpegj.scene.SFTimeFieldValue

V.5.333.1 Syntax

```
public interface SFTimeFieldValue extends org.iso.mpeg.mpegj.scene.FieldValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue

V.5.333.2 Description

An interface used for obtaining SFTime values.

Member Summary	
Methods	
double	getSFTimeValue()

V.5.333.3 Methods

getSFTimeValue()
```
public double getSFTimeValue()
```
Obtain the SFTime value.

Returns:
a double value of the SFTime field.

V.5.334 org.iso.mpeg.mpegj.scene.SFVec2fFieldValue

V.5.334.1 Syntax

```
public interface SFVec2fFieldValue extends org.iso.mpeg.mpegj.scene.FieldValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue

V.5.334.2 Description

An interface used for obtaining SFVec2f values.

Member Summary	
Methods	
float[]	getSFVec2fValue()

V.5.334.3 Methods

getSFVec2fValue()
```
public float[] getSFVec2fValue()
```
Obtain the SFVec2f value.

Returns:
a float array containing the value of the SFVec2f field.

V.5.335 org.iso.mpeg.mpegj.scene.SFVec3fFieldValue

V.5.335.1 Syntax

```
public interface SFVec3fFieldValue extends org.iso.mpeg.mpegj.scene.FieldValue
```

All Superinterfaces:
org.iso.mpeg.mpegj.scene.FieldValue

© ISO/IEC 2001 – All rights reserved

Hulu, LLC - Ex. 1024, Page 000681

V.5.335.2 Description

An interface used for obtaining SFVec3f values.

Member Summary	
Methods	
`float[]`	`getSFVec3fValue()`

V.5.335.3 Methods

getSFVec3fValue()

V.5.336 public float[] getSFVec3fValue()

Obtain the SFVec3f value.

Returns:
a float array containing the value of the SFVec3f field.

© ISO/IEC 2001 – All rights reserved
Hulu, LLC - Ex. 1024, Page 000682

ICS 35.040

Price based on 691 pages

© ISO/IEC 2001 – All rights reserved