INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC1/SC29/WG11
CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC1/SC29/WG11 N1901
21 November 1997

Source: **MPEG-4 Systems**
Status: **Approved at the 41th Meeting**
Title: **Text for CD 14496-1 Systems**
Authors: **Alexandros Eleftheriadis, Carsten Herpel, Ganesh Rajan, and Liam Ward (Editors)**

Version of: 21 September 2007 11:29:11

Please address any comments or suggestions to spec-sys@fzi.de

Table of Contents

0. Introduction....1

0.1 Architecture....1

0.2 Systems Decoder Model....2

0.2.1 Timing Model....3

0.2.2 Buffer Model....3

0.3 FlexMux and TransMux Layer....3

0.4 AccessUnit Layer....3

0.5 Compression Layer....3

0.5.1 Object Descriptor Elementary Streams....3

0.5.2 Scene Description Streams....4

0.5.3 Upchannel Streams....4

0.5.4 Object Content Information Streams....4

1. Scope....4

2. Normative References....5

3. Additional References....5

4. Definitions....5

5. Abbreviations and Symbols....7

6. Conventions....7

6.1 Syntax Description....7

7. Specification....8

7.1 Systems Decoder Model....8

7.1.1 Introduction....8

7.1.2 Concepts of the Systems Decoder Model....9

7.1.2.1 DMIF Application Interface (DAI)....9

7.1.2.2 AL-packetized Stream (APS)....9

7.1.2.3 Access Units (AU)....9

7.1.2.4 Decoding Buffer (DB)....9

7.1.2.5 Elementary Streams (ES)....9

7.1.2.6 Elementary Stream Interface (ESI)....9

7.1.2.7 Media Object Decoder....10

7.1.2.8 Composition Units (CU)....10

7.1.2.9 Composition Memory (CM)....10

7.1.2.10 Compositor....10

7.1.3 Timing Model Specification....10

7.1.3.1 System Time Base (STB)....10

7.1.3.2 Object Time Base (OTB)....10

7.1.3.3 Object Clock Reference (OCR)....11

7.1.3.4 Decoding Time Stamp (DTS)....11

7.1.3.5 Composition Time Stamp (CTS)....11

7.1.3.6 Occurrence of timing information in Elementary Streams....11

7.1.3.7 Example....11

7.1.4 Buffer Model Specification....12

7.1.4.1 Elementary decoder model....12

7.1.4.2 Assumptions....13

7.1.4.2.1 Constant end-to-end delay 13
7.1.4.2.2 Demultiplexer 13
7.1.4.2.3 Decoding Buffer 13
7.1.4.2.4 Decoder 14
7.1.4.2.5 Composition Memory 14
7.1.4.2.6 Compositor 14
7.1.4.3 Managing Buffers: A Walkthrough 14

7.2 Scene Description 15
7.2.1 Introduction 15
7.2.1.1 Scope 15
7.2.1.2 Composition 16
7.2.1.3 Scene Description 16
7.2.1.3.1 Grouping of objects 16
7.2.1.3.2 Spatio-Temporal positioning of objects 16
7.2.1.3.3 Attribute value selection 17
7.2.2 Concepts 17
7.2.2.1 Global structure of a BIFS Scene Description 17
7.2.2.2 BIFS Scene graph 17
7.2.2.3 2D Coordinate System 18
7.2.2.4 3D Coordinate System 19
7.2.2.5 Standard Units 19
7.2.2.6 Mapping of scenes to screens 20
7.2.2.7 Nodes and fields 20
7.2.2.7.1 Nodes 20
7.2.2.7.2 Fields and Events 20
7.2.2.8 Basic data types 20
7.2.2.8.1 Numerical data and string data types 20
7.2.2.8.1.1 SFBool 21
7.2.2.8.1.2 SFColor/MFColor 21
7.2.2.8.1.3 SFFloat/MFFloat 21
7.2.2.8.1.4 SFInt32/MFInt32 21
7.2.2.8.1.5 SFRotation/MFRotation 21
7.2.2.8.1.6 SFString/MFString 21
7.2.2.8.1.7 SFTime 21
7.2.2.8.1.8 SFVec2f/MFVec2f 21
7.2.2.8.1.9 SFVec3f/MFVec3f 21
7.2.2.8.2 Node data types 21
7.2.2.9 Attaching nodeIDs to nodes 21
7.2.2.10 Using pre-defined nodes 21
7.2.2.11 Scene Structure and Semantics 22
7.2.2.11.1 2D Grouping Nodes 22
7.2.2.11.2 2D Geometry Nodes 22
7.2.2.11.3 2D Material Nodes 22
7.2.2.11.4 Face and Body nodes 22
7.2.2.11.5 Mixed 2D/3D Nodes 22
7.2.2.12 Internal, ASCII and Binary Representation of Scenes 23
7.2.2.12.1 Binary Syntax Overview 23
7.2.2.12.1.1 Scene Description 23
7.2.2.12.1.2 Node Description 23
7.2.2.12.1.3 Fields description 23
7.2.2.12.1.4 ROUTE description 23
7.2.2.13 BIFS Elementary Streams 23
7.2.2.13.1 BIFS-Update commands 23
7.2.2.13.2 BIFS Access Units 23
7.2.2.13.3 Requirements on BIFS elementary stream transport 24
7.2.2.13.4 Time base for the scene description 24
7.2.2.13.5 Composition Time Stamp semantics for BIFS Access Units 24
7.2.2.13.6 Multiple BIFS streams 24

7.2.2.13.7 Time Fields in BIFS nodes....24
7.2.2.13.7.1 Example....25
7.2.2.13.8 Time events based on media time....25
7.2.2.14 Sound....25
7.2.2.14.1 Overview of sound node semantics....26
7.2.2.14.1.1 Sample-rate conversion....26
7.2.2.14.1.1.1 Example....27
7.2.2.14.1.2 Number of output channels....27
7.2.2.14.2 Audio-specific BIFS....27
7.2.2.14.2.1 Audio-related BIFS nodes....27
7.2.2.15 Drawing Order....28
7.2.2.15.1 Scope of Drawing Order....28
7.2.2.16 Bounding Boxes....28
7.2.2.17 Sources of modification to the scene....28
7.2.2.17.1 Interactivity and behaviors....28
7.2.2.17.2 External modification of the scene: BIFS Update....28
7.2.2.17.2.1 Overview....28
7.2.2.17.2.2 Update examples....29
7.2.2.17.3 External animation of the scene: BIFS-Anim....30
7.2.2.17.3.1 Overview....30
7.2.2.17.3.2 Animation Mask....30
7.2.2.17.3.3 Animation Frames....30
7.2.2.17.3.4 Animation Examples....30
7.2.3 BIFS Binary Syntax....31
7.2.3.1 BIFS Scene and Nodes Syntax....31
7.2.3.1.1 BIFSScene....31
7.2.3.1.2 BIFSNodes....31
7.2.3.1.3 SFNode....31
7.2.3.1.4 MaskNodeDescription....32
7.2.3.1.5 ListNodeDescription....32
7.2.3.1.6 NodeType....32
7.2.3.1.7 Field....32
7.2.3.1.8 MFField....33
7.2.3.1.9 SFField....33
7.2.3.1.9.1 SFBool....34
7.2.3.1.9.2 SFColor....34
7.2.3.1.9.3 SFFloat....34
7.2.3.1.9.4 SFImage....34
7.2.3.1.9.5 SFInt32....34
7.2.3.1.9.6 SFRotation....35
7.2.3.1.9.7 SFString....35
7.2.3.1.9.8 SFTime....35
7.2.3.1.9.9 SFUrl....35
7.2.3.1.9.10 SFVec2f....35
7.2.3.1.9.11 SFVec3f....36
7.2.3.1.10 QuantizedField....36
7.2.3.1.11 Field IDs syntax....37
7.2.3.1.11.1 defID....37
7.2.3.1.11.2 inID....37
7.2.3.1.11.3 outID....37
7.2.3.1.11.4 dynID....37
7.2.3.1.12 ROUTE syntax....37
7.2.3.1.12.1 ROUTEs....37
7.2.3.1.12.2 ListROUTEs....38
7.2.3.1.12.3 VectorROUTEs....38
7.2.3.1.12.3.1 ROUTE....38
7.2.3.2 BIFS-Update Syntax....38
7.2.3.2.1 Update Frame....38
7.2.3.2.2 Update Command....38

7.2.3.2.3 Insertion Command....39
7.2.3.2.3.1 Node Insertion....39
7.2.3.2.3.2 IndexedValue Insertion....39
7.2.3.2.3.3 ROUTE Insertion....40
7.2.3.2.4 Deletion Command....40
7.2.3.2.4.1 Node Deletion....40
7.2.3.2.4.2 IndexedValue Deletion....40
7.2.3.2.4.3 ROUTE Deletion....41
7.2.3.2.5 Replacement Command....41
7.2.3.2.5.1 Node Replacement....41
7.2.3.2.5.2 Field Replacement....41
7.2.3.2.5.3 IndexedValue Replacement....41
7.2.3.2.5.4 ROUTE Replacement....42
7.2.3.2.5.5 Scene Replacement....42
7.2.3.3 BIFS-Anim Syntax....42
7.2.3.3.1 BIFS AnimationMask....42
7.2.3.3.1.1 AnimationMask....42
7.2.3.3.1.2 Elementary mask....42
7.2.3.3.1.3 InitialFieldsMask....42
7.2.3.3.1.4 }InitialAnimQP....43
7.2.3.3.2 Animation Frame Syntax....44
7.2.3.3.2.1 AnimationFrame....44
7.2.3.3.2.2 AnimationFrameHeader....44
7.2.3.3.2.3 AnimationFrameData....45
7.2.3.3.2.4 AnimationField....45
7.2.3.3.2.5 AnimQP....46
7.2.3.3.2.6 AnimationIValue....48
7.2.3.3.2.7 AnimationPValue....49
7.2.4 BIFS Decoding Process and Semantic....50
7.2.4.1 BIFS Scene and Nodes Decoding Process....50
7.2.4.1.1 BIFS Scene....50
7.2.4.1.2 BIFS Nodes....50
7.2.4.1.3 SFNode....50
7.2.4.1.4 MaskNodeDescription....50
7.2.4.1.5 ListNodeDescription....51
7.2.4.1.6 NodeType....51
7.2.4.1.7 Field....51
7.2.4.1.8 MFField....51
7.2.4.1.9 SFField....51
7.2.4.1.10 QuantizedField....52
7.2.4.1.11 Field and Events IDs Decoding Process....54
7.2.4.1.11.1 DefID....54
7.2.4.1.11.2 inID....54
7.2.4.1.11.3 outID....54
7.2.4.1.11.4 dynID....54
7.2.4.1.12 ROUTE Decoding Process....54
7.2.4.2 BIFS-Update Decoding Process....54
7.2.4.2.1 Update Frame....54
7.2.4.2.2 Update Command....54
7.2.4.2.3 Insertion Command....54
7.2.4.2.3.1 Node Insertion....54
7.2.4.2.3.2 IndexedValue Insertion....54
7.2.4.2.3.3 ROUTE Insertion....54
7.2.4.2.4 Deletion Command....55
7.2.4.2.4.1 Node Deletion....55
7.2.4.2.4.2 IndexedValue Deletion....55
7.2.4.2.4.3 ROUTE Deletion....55
7.2.4.2.5 Replacement Command....55
7.2.4.2.5.1 Node Replacement....55

7.2.4.2.5.2 Field Replacement....55
7.2.4.2.5.3 IndexedValue Replacement....55
7.2.4.2.5.4 ROUTE Replacement....55
7.2.4.2.5.5 Scene Replacement....55
7.2.4.2.5.6 Scene Repeat....55
7.2.4.3 BIFS-Anim Decoding Process....56
7.2.4.3.1 BIFS AnimationMask....56
7.2.4.3.1.1 AnimationMask....56
7.2.4.3.1.2 Elementary mask....56
7.2.4.3.1.3 InitialFieldsMask....57
7.2.4.3.1.4 InitialAnimQP....57
7.2.4.3.2 Animation Frame Decoding Process....57
7.2.4.3.2.1 AnimationFrame....57
7.2.4.3.2.2 AnimationFrameHeader....57
7.2.4.3.2.3 AnimationFrameData....57
7.2.4.3.2.4 AnimationField....57
7.2.4.3.2.5 AnimQP....58
7.2.4.3.2.6 AnimationIValue....58
7.2.4.3.2.7 AnimationPValue....58
7.2.5 Nodes Semantic....58
7.2.5.1 Shared Nodes....58
7.2.5.1.1 Shared Nodes Overview....58
7.2.5.1.2 Shared MPEG-4 Nodes....58
7.2.5.1.2.1 AnimationStream....58
7.2.5.1.2.1.1 Semantic Table....58
7.2.5.1.2.1.2 Main Functionality....59
7.2.5.1.2.1.3 Detailed Semantic....59
7.2.5.1.2.2 AudioDelay....59
7.2.5.1.2.2.1 Semantic Table....59
7.2.5.1.2.2.2 Main Functionality....60
7.2.5.1.2.2.3 Detailed Semantics....60
7.2.5.1.2.3 AudioMix....60
7.2.5.1.2.3.1 Semantic Table....60
7.2.5.1.2.3.2 Main Functionality....60
7.2.5.1.2.3.3 Detailed Semantics....60
7.2.5.1.2.3.4 Calculation....60
7.2.5.1.2.4 AudioSource....61
7.2.5.1.2.4.1 Semantic Table....61
7.2.5.1.2.4.2 Main Functionality....61
7.2.5.1.2.4.3 Detailed Semantics....61
7.2.5.1.2.4.4 Calculation....61
7.2.5.1.2.5 AudioFX....62
7.2.5.1.2.5.1 Semantic Table....62
7.2.5.1.2.5.2 Main Functionality....62
7.2.5.1.2.5.3 Detailed Semantics....62
7.2.5.1.2.5.4 Calculation....62
7.2.5.1.2.6 AudioSwitch....63
7.2.5.1.2.6.1 Semantic Table....63
7.2.5.1.2.6.2 Main Functionality....63
7.2.5.1.2.6.3 Detailed Semantics....63
7.2.5.1.2.6.4 Calculation....63
7.2.5.1.2.7 Conditional....63
7.2.5.1.2.7.1 Semantic Table....63
7.2.5.1.2.7.2 Main Functionality....63
7.2.5.1.2.7.3 Detailed Semantics....64
7.2.5.1.2.8 MediaTimeSensor....64
7.2.5.1.2.8.1 Semantic Table....64
7.2.5.1.2.8.2 Main Functionality....64
7.2.5.1.2.8.3 Detailed Semantics....64

7.2.5.1.2.9 QuantizationParameter....64
7.2.5.1.2.9.1 Semantic Table....64
7.2.5.1.2.9.2 Main Functionality....65
7.2.5.1.2.9.3 Detailed Semantics....65
7.2.5.1.2.9.4 Example....65
7.2.5.1.2.10 StreamingText....66
7.2.5.1.2.10.1 Semantic Table....66
7.2.5.1.2.10.2 Main Functionality....66
7.2.5.1.2.10.3 Detailed Semantics....66
7.2.5.1.2.11 Valuator....66
7.2.5.1.2.11.1 Semantic Table....66
7.2.5.1.2.11.2 Main Functionality....67
7.2.5.1.2.11.3 Detailed Semantics....67
7.2.5.1.3 Shared VRML Nodes....67
7.2.5.1.3.1 Appearance....68
7.2.5.1.3.1.1 Semantic Table....68
7.2.5.1.3.2 AudioClip....68
7.2.5.1.3.2.1 Semantic Table....68
7.2.5.1.3.2.2 Main Functionality....68
7.2.5.1.3.2.3 Detailed Semantics....68
7.2.5.1.3.2.4 Calculation....68
7.2.5.1.3.3 Color 69
7.2.5.1.3.3.1 Semantic Table....69
7.2.5.1.3.4 ColorInterpolator....69
7.2.5.1.3.4.1 Semantic Table....69
7.2.5.1.3.5 FontStyle....69
7.2.5.1.3.5.1 Semantic Table....69
7.2.5.1.3.6 ImageTexture....69
7.2.5.1.3.6.1 Semantic Table....69
7.2.5.1.3.6.2 Detailed Semantics....69
7.2.5.1.3.7 MovieTexture....70
7.2.5.1.3.7.1 Semantic Table....70
7.2.5.1.3.7.2 Detailed Semantics....70
7.2.5.1.3.8 ScalarInterpolator....71
7.2.5.1.3.8.1 Semantic Table....71
7.2.5.1.3.9 Shape 71
7.2.5.1.3.9.1 Semantic Table....71
7.2.5.1.3.10 Sound....71
7.2.5.1.3.10.1 Semantic Table....71
7.2.5.1.3.10.2 Main Functionality....71
7.2.5.1.3.10.3 Detailed Semantics....71
7.2.5.1.3.10.4 Nodes above the Sound node....72
7.2.5.1.3.10.5 Example....72
7.2.5.1.3.11 Switch 72
7.2.5.1.3.11.1 Semantic Table....72
7.2.5.1.3.12 Text....73
7.2.5.1.3.12.1 Semantic Table....73
7.2.5.1.3.13 TextureCoordinate....73
7.2.5.1.3.13.1 Semantic Table....73
7.2.5.1.3.14 TextureTransform....73
7.2.5.1.3.14.1 Semantic Table....73
7.2.5.1.3.15 TimeSensor....73
7.2.5.1.3.15.1 Semantic Table....73
7.2.5.1.3.16 TouchSensor....73
7.2.5.1.3.16.1 Semantic Table....73
7.2.5.1.3.17 WorldInfo....74
7.2.5.1.3.17.1 Semantic Table....74
7.2.5.2 2D Nodes....74
7.2.5.2.1 2D Nodes Overview....74

7.2.5.2.2 2D MPEG-4 Nodes....74
7.2.5.2.2.1 Background2D....74
7.2.5.2.2.1.1 Semantic Table....74
7.2.5.2.2.1.2 Main Functionality....74
7.2.5.2.2.1.3 Detailed Semantics....74
7.2.5.2.2.1.4 Example....74
7.2.5.2.2.2 Circle....75
7.2.5.2.2.2.1 Semantic Table....75
7.2.5.2.2.2.2 Main Functionality....75
7.2.5.2.2.2.3 Detailed Semantics....75
7.2.5.2.2.3 Coordinate2D....75
7.2.5.2.2.3.1 Semantic Table....75
7.2.5.2.2.3.2 Main Functionality....75
7.2.5.2.2.3.3 Detailed Semantics....75
7.2.5.2.2.4 Curve2D....75
7.2.5.2.2.4.1 Semantic Table....75
7.2.5.2.2.4.2 Main Functionality....75
7.2.5.2.2.4.3 Detailed Semantics....76
7.2.5.2.2.4.4 Example....76
7.2.5.2.2.5 DiscSensor....76
7.2.5.2.2.5.1 Semantic Table....76
7.2.5.2.2.5.2 Main Functionality....76
7.2.5.2.2.5.3 Detailed Semantics....76
7.2.5.2.2.6 Form....77
7.2.5.2.2.6.1 Semantic Table....77
7.2.5.2.2.6.2 Main Functionality....77
7.2.5.2.2.6.3 Detailed Semantics....77
7.2.5.2.2.6.4 Example....78
7.2.5.2.2.7 Group2D....79
7.2.5.2.2.7.1 Semantic Table....79
7.2.5.2.2.7.2 Main Functionality....79
7.2.5.2.2.7.3 Detailed Semantics....79
7.2.5.2.2.7.4 Example....79
7.2.5.2.2.8 Image2D....80
7.2.5.2.2.8.1 Semantic Table....80
7.2.5.2.2.8.2 Main Functionality....80
7.2.5.2.2.8.3 Detailed Semantics....80
7.2.5.2.2.9 IndexedFaceSet2D....80
7.2.5.2.2.9.1 Semantic Table....80
7.2.5.2.2.9.2 Main Functionality....80
7.2.5.2.2.9.3 Detailed Semantics....80
7.2.5.2.2.10 IndexedLineSet2D....81
7.2.5.2.2.10.1 Semantic Table....81
7.2.5.2.2.10.2 Main Functionality....81
7.2.5.2.2.10.3 Detailed Semantics....81
7.2.5.2.2.11 Inline2D....81
7.2.5.2.2.11.1 Semantic Table....81
7.2.5.2.2.11.2 Main Functionality....81
7.2.5.2.2.11.3 Detailed Semantics....81
7.2.5.2.2.11.4 Example....81
7.2.5.2.2.12 Layout....82
7.2.5.2.2.12.1 Semantic Table....82
7.2.5.2.2.12.2 Main functionality....82
7.2.5.2.2.12.3 Detailed Semantics....82
7.2.5.2.2.12.4 Example....83
7.2.5.2.2.13 LineProperties....84
7.2.5.2.2.13.1 Semantic Table....84
7.2.5.2.2.13.2 Main Functionality....84
7.2.5.2.2.13.3 Detailed Semantics....84

7.2.5.2.2.14 Material2D....84
7.2.5.2.2.14.1 Semantic Table....84
7.2.5.2.2.14.2 Main Functionality....85
7.2.5.2.2.14.3 Detailed Semantics....85
7.2.5.2.2.15 VideoObject2D....85
7.2.5.2.2.15.1 Semantic Table....85
7.2.5.2.2.15.2 Main Functionality....85
7.2.5.2.2.15.3 Detailed Semantics....85
7.2.5.2.2.16 PlaneSensor2D....86
7.2.5.2.2.16.1 Semantic Table....86
7.2.5.2.2.16.2 Main Functionality....86
7.2.5.2.2.16.3 Detailed Semantics....86
7.2.5.2.2.17 PointSet2D....86
7.2.5.2.2.17.1 Semantic Table....86
7.2.5.2.2.17.2 Main Functionality....87
7.2.5.2.2.17.3 Detailed Semantics....87
7.2.5.2.2.18 Position2DInterpolator....87
7.2.5.2.2.18.1 Semantic Table....87
7.2.5.2.2.18.2 Main Functionality....87
7.2.5.2.2.18.3 Detailed Semantics....87
7.2.5.2.2.19 Proximity2DSensor....87
7.2.5.2.2.19.1 Semantic Table....87
7.2.5.2.2.19.2 Main Functionality....87
7.2.5.2.2.19.3 Detailed Semantics....87
7.2.5.2.2.20 Rectangle....87
7.2.5.2.2.20.1 Semantic Table....87
7.2.5.2.2.20.2 Main Functionality....87
7.2.5.2.2.20.3 Detailed Semantics....87
7.2.5.2.2.21 ShadowProperties....88
7.2.5.2.2.21.1 Main Functionality....88
7.2.5.2.2.21.2 Detailed Semantics....88
7.2.5.2.2.22 Switch2D....88
7.2.5.2.2.22.1 Semantic Table....88
7.2.5.2.2.22.2 Main functionality....88
7.2.5.2.2.22.3 Detailed Semantics....88
7.2.5.2.2.23 Transform2D....88
7.2.5.2.2.23.1 Semantic Table....88
7.2.5.2.2.23.2 Main Functionality....89
7.2.5.2.2.23.3 Detailed Semantics....89
7.2.5.2.2.24 VideoObject2D....89
7.2.5.3 3D Nodes....90
7.2.5.3.1 3D Nodes Overview....90
7.2.5.3.2 3D MPEG-4 Nodes....90
7.2.5.3.2.1 ListeningPoint....90
7.2.5.3.2.1.1 Semantic Table....90
7.2.5.3.2.1.2 Main Functionality....90
7.2.5.3.2.1.3 Detailed Semantics....91
7.2.5.3.2.2 FBA....91
7.2.5.3.2.2.1 Semantic Table....91
7.2.5.3.2.2.2 Main Functionality....91
7.2.5.3.2.2.3 Detailed Semantics....91
7.2.5.3.2.3 Face....91
7.2.5.3.2.3.1 Semantic Table....91
7.2.5.3.2.3.2 Main Functionality....91
7.2.5.3.2.3.3 Detailed Semantics....91
7.2.5.3.2.3.3.1 Interpolating FAPs....91
7.2.5.3.2.4 FIT....93
7.2.5.3.2.4.1 Semantic Table....93
7.2.5.3.2.4.2 Main Functionality....93

7.2.5.3.2.4.3 Detailed Semantics....93
7.2.5.3.2.4.4 Example....94
7.2.5.3.2.5 FAP....94
7.2.5.3.2.5.1 Semantic Table....94
7.2.5.3.2.5.2 Main Functionality....96
7.2.5.3.2.5.3 Detailed Semantics....96
7.2.5.3.2.6 FDP....96
7.2.5.3.2.6.1 Semantic Table....96
7.2.5.3.2.6.2 Main Functionality....96
7.2.5.3.2.6.3 Detailed Semantics....97
7.2.5.3.2.7 FBADefTable....97
7.2.5.3.2.7.1 Semantic Table....97
7.2.5.3.2.7.2 Main Functionality....97
7.2.5.3.2.7.3 Detailed Semantics....97
7.2.5.3.2.8 FBADefTransform....98
7.2.5.3.2.8.1 Semantic Table....98
7.2.5.3.2.8.2 Main Functionality....98
7.2.5.3.2.8.3 Detailed Semantics....98
7.2.5.3.2.9 FBADefMesh....98
7.2.5.3.2.9.1 Semantic Table....98
7.2.5.3.2.9.2 Main Functionality....98
7.2.5.3.2.9.3 Detailed Semantics....98
7.2.5.3.3 3D VRML Nodes....99
7.2.5.3.3.1 Background....99
7.2.5.3.3.1.1 Semantic Table....99
7.2.5.3.3.1.2 Detailed Semantics....99
7.2.5.3.3.2 Billboard....99
7.2.5.3.3.2.1 Semantic Table....99
7.2.5.3.3.3 Box....100
7.2.5.3.3.3.1 Semantic Table....100
7.2.5.3.3.4 Collision....100
7.2.5.3.3.4.1 Semantic Table....100
7.2.5.3.3.5 Cone....100
7.2.5.3.3.5.1 Semantic Table....100
7.2.5.3.3.6 Coordinate....100
7.2.5.3.3.6.1 Semantic Table....100
7.2.5.3.3.7 CoordinateInterpolator....100
7.2.5.3.3.7.1 Semantic Table....100
7.2.5.3.3.8 Cylinder....101
7.2.5.3.3.8.1 Semantic Table....101
7.2.5.3.3.9 DirectionalLight....101
7.2.5.3.3.9.1 Semantic Table....101
7.2.5.3.3.10 ElevationGrid....101
7.2.5.3.3.10.1 Semantic Table....101
7.2.5.3.3.11 Extrusion....101
7.2.5.3.3.11.1 Semantic Table....101
7.2.5.3.3.12 Group....102
7.2.5.3.3.12.1 Semantic Table....102
7.2.5.3.3.12.2 Detailed Semantics....102
7.2.5.3.3.13 IndexedFaceSet....102
7.2.5.3.3.13.1 Semantic Table....102
7.2.5.3.3.13.2 Main Functionality....103
7.2.5.3.3.13.3 Detailed Semantics....103
7.2.5.3.3.14 IndexedLineSet....103
7.2.5.3.3.14.1 Semantic Table....103
7.2.5.3.3.15 Inline....103
7.2.5.3.3.15.1 Semantic Table....103
7.2.5.3.3.15.2 Detailed Semantics....103
7.2.5.3.3.16 LOD....103

7.2.5.3.3.16.1 Semantic Table....103
7.2.5.3.3.17 Material....104
7.2.5.3.3.17.1 Semantic Table....104
7.2.5.3.3.18 Normal....104
7.2.5.3.3.18.1 Semantic Table....104
7.2.5.3.3.19 NormalInterpolator....104
7.2.5.3.3.19.1 Semantic Table....104
7.2.5.3.3.20 OrientationInterpolator....104
7.2.5.3.3.20.1 Semantic Table....104
7.2.5.3.3.21 PointLight....104
7.2.5.3.3.21.1 Semantic Table....104
7.2.5.3.3.22 PointSet....105
7.2.5.3.3.22.1 Semantic Table....105
7.2.5.3.3.23 PositionInterpolator....105
7.2.5.3.3.23.1 Semantic Table....105
7.2.5.3.3.24 ProximitySensor....105
7.2.5.3.3.24.1 Semantic Table....105
7.2.5.3.3.25 Sphere....105
7.2.5.3.3.25.1 Semantic Table....105
7.2.5.3.3.26 SpotLight....105
7.2.5.3.3.27 Semantic Table....105
7.2.5.3.3.28 Transform....106
7.2.5.3.3.28.1 Semantic Table....106
7.2.5.3.3.28.2 Detailed Semantics....106
7.2.5.3.3.29 Viewpoint....106
7.2.5.3.3.29.1 Semantic Table....106
7.2.5.3.3.29.2 Detailed Semantics....107
7.2.5.4 Mixed 2D/3D Nodes....107
7.2.5.4.1 Mixed 2D/3D Nodes Overview....107
7.2.5.4.2 2D/3D MPEG-4 Nodes....107
7.2.5.4.2.1 Layer2D....107
7.2.5.4.2.1.1 Semantic Table....107
7.2.5.4.2.1.2 Main Functionality....107
7.2.5.4.2.1.3 Detailed Semantics....107
7.2.5.4.2.2 Layer3D....108
7.2.5.4.2.2.1 Semantic Table....108
7.2.5.4.2.2.2 Main Functionality....108
7.2.5.4.2.2.3 Detailed Semantics....108
7.2.5.4.2.3 Composite2DTexture....109
7.2.5.4.2.3.1 Semantic Table....109
7.2.5.4.2.3.2 Main Functionality....109
7.2.5.4.2.3.3 Detailed Semantics....110
7.2.5.4.2.4 Composite3DTexture....110
7.2.5.4.2.4.1 Semantic Table....110
7.2.5.4.2.4.2 Main Functionality....110
7.2.5.4.2.4.3 Detailed Semantics....111
7.2.5.4.2.5 CompositeMap....111
7.2.5.4.2.5.1 Semantic Table....111
7.2.5.4.2.5.2 Main Functionality....112
7.2.5.4.2.5.3 The CompositeMap node is used for a 2D scene appearing on a plane in a 3D scene. A similar functionnality can be achieved with the CompositeTexture2D, but when using the CompsiteMap, you do not have to use texture projection to draw the 2D scene in the local XY planeDetailed Semantics112
7.2.6 Node Coding Parameters....112
7.2.6.1 Table Semantic....112
7.2.6.2 Node Data Type tables....113
7.2.6.2.1 SF2DNode....113
7.2.6.2.2 SF3DNode....113
7.2.6.2.3 SFAppearanceNode....114

7.2.6.2.4 SFAudioNode....114
7.2.6.2.5 SFColorNode....114
7.2.6.2.6 SFCoordinate2DNode....114
7.2.6.2.7 SFCoordinateNode....115
7.2.6.2.8 SFFAPNode....115
7.2.6.2.9 SFFBADefNode....115
7.2.6.2.10 SFFBADefTableNode....115
7.2.6.2.11 SFFDPNode....115
7.2.6.2.12 SFFaceNode....115
7.2.6.2.13 SFFitNode....115
7.2.6.2.14 SFFontStyleNode....115
7.2.6.2.15 SFGeometryNode....115
7.2.6.2.16 SFLayerNode....116
7.2.6.2.17 SFLinePropertiesNode....116
7.2.6.2.18 SFMaterialNode....116
7.2.6.2.19 SFNormalNode....116
7.2.6.2.20 SFShadowPropertiesNode....116
7.2.6.2.21 SFStreamingNode....116
7.2.6.2.22 SFTextureCoordinateNode....117
7.2.6.2.23 SFTextureNode....117
7.2.6.2.24 SFTextureTransformNode....117
7.2.6.2.25 SFTimerNode....117
7.2.6.2.26 SFTopNode....117
7.2.6.2.27 SFWorldInfoNode....117
7.2.6.2.28 SFWorldNode....117
7.2.6.3 Node Coding Tables....119
7.2.6.3.1 Key for Node Coding Tables....119
7.2.6.3.2 AnimationStream....119
7.2.6.3.3 AudioDelay....119
7.2.6.3.4 AudioMix....120
7.2.6.3.5 AudioSource....120
7.2.6.3.6 AudioFX....120
7.2.6.3.7 AudioSwitch....120
7.2.6.3.8 Conditional....121
7.2.6.3.9 MediaTimeSensor....121
7.2.6.3.10 QuantizationParameter....121
7.2.6.3.11 StreamingText....122
7.2.6.3.12 Valuator....122
7.2.6.3.13 Appearance....123
7.2.6.3.14 AudioClip....123
7.2.6.3.15 Color....124
7.2.6.3.16 ColorInterpolator....124
7.2.6.3.17 FontStyle....124
7.2.6.3.18 ImageTexture....124
7.2.6.3.19 MovieTexture....124
7.2.6.3.20 ScalarInterpolator....125
7.2.6.3.21 Shape....125
7.2.6.3.22 Sound....125
7.2.6.3.23 Switch....126
7.2.6.3.24 Text....126
7.2.6.3.25 TextureCoordinate....126
7.2.6.3.26 TextureTransform....126
7.2.6.3.27 TimeSensor....126
7.2.6.3.28 TouchSensor....127
7.2.6.3.29 WorldInfo....127
7.2.6.3.30 Background2D....127
7.2.6.3.31 Circle....127
7.2.6.3.32 Coordinate2D....127
7.2.6.3.33 Curve2D....128

7.2.6.3.34 DiscSensor....128
7.2.6.3.35 Form....128
7.2.6.3.36 Group2D....128
7.2.6.3.37 Image2D....128
7.2.6.3.38 IndexedFaceSet2D....129
7.2.6.3.39 IndexedLineSet2D....129
7.2.6.3.40 Inline2D....129
7.2.6.3.41 Layout....129
7.2.6.3.42 LineProperties....130
7.2.6.3.43 Material2D....130
7.2.6.3.44 PlaneSensor2D....130
7.2.6.3.45 PointSet2D....130
7.2.6.3.46 Position2DInterpolator....131
7.2.6.3.47 Proximity2DSensor....131
7.2.6.3.48 Rectangle....131
7.2.6.3.49 ShadowProperties....131
7.2.6.3.50 Switch2D....131
7.2.6.3.51 Transform2D....131
7.2.6.3.52 VideoObject2D....132
7.2.6.3.53 ListeningPoint....132
7.2.6.3.54 FBA....132
7.2.6.3.55 Face....133
7.2.6.3.56 FIT....133
7.2.6.3.57 FAP....133
7.2.6.3.58 FDP....136
7.2.6.3.59 FBADefMesh....136
7.2.6.3.60 FBADefTable....136
7.2.6.3.61 FBADefTransform....136
7.2.6.3.62 Background....136
7.2.6.3.63 Billboard....137
7.2.6.3.64 Box....137
7.2.6.3.65 Collision....137
7.2.6.3.66 Cone....137
7.2.6.3.67 Coordinate....138
7.2.6.3.68 CoordinateInterpolator....138
7.2.6.3.69 Cylinder....138
7.2.6.3.70 DirectionalLight....138
7.2.6.3.71 ElevationGrid....138
7.2.6.3.72 Extrusion....139
7.2.6.3.73 Group....139
7.2.6.3.74 IndexedFaceSet....139
7.2.6.3.75 IndexedLineSet....140
7.2.6.3.76 Inline....140
7.2.6.3.77 LOD....140
7.2.6.3.78 Material....140
7.2.6.3.79 Normal....141
7.2.6.3.80 NormalInterpolator....141
7.2.6.3.81 OrientationInterpolator....141
7.2.6.3.82 PointLight....141
7.2.6.3.83 PointSet....141
7.2.6.3.84 PositionInterpolator....142
7.2.6.3.85 ProximitySensor....142
7.2.6.3.86 Sphere....142
7.2.6.3.87 SpotLight....142
7.2.6.3.88 Transform....143
7.2.6.3.89 Viewpoint....143
7.2.6.3.90 Layer2D....143
7.2.6.3.91 Layer3D....143
7.2.6.3.92 Composite2DTexture....144

7.2.6.3.93 Composite3DTexture....144
7.2.6.3.94 CompositeMap....144
7.3 Identification and Association of Elementary Streams....145
7.3.1 Introduction....145
7.3.2 Object Descriptor Elementary Stream....145
7.3.2.1 Structure of the Object Descriptor Elementary Stream....146
7.3.2.2 OD-Update Syntax and Semantics....146
7.3.2.2.1 ObjectDescriptorUpdate....146
7.3.2.2.1.1 Syntax....146
7.3.2.2.1.2 Semantics....146
7.3.2.2.2 ObjectDescriptorRemove....146
7.3.2.2.2.1 Syntax....146
7.3.2.2.2.2 Semantics....147
7.3.2.2.3 ES_DescriptorUpdate....147
7.3.2.2.3.1 Syntax....147
7.3.2.2.3.2 Semantics....147
7.3.2.2.4 ES_DescriptorRemove....147
7.3.2.2.4.1 Syntax....147
7.3.2.2.4.2 Semantics....147
7.3.2.3 Descriptor tags 148
7.3.3 Object Descriptor Syntax and Semantics....148
7.3.3.1 ObjectDescriptor....148
7.3.3.1.1 Syntax....148
7.3.3.1.2 Semantics....149
7.3.3.2 ES_descriptor....149
7.3.3.2.1 Syntax....149
7.3.3.2.2 Semantics....150
7.3.3.3 DecoderConfigDescriptor....151
7.3.3.3.1 Syntax....151
7.3.3.3.2 Semantics....151
7.3.3.4 ALConfigDescriptor....153
7.3.3.5 IPI_Descriptor....153
7.3.3.5.1 Syntax....153
7.3.3.5.2 Semantics....153
7.3.3.5.3 IP Identification Data Set....153
7.3.3.5.3.1 Syntax....153
7.3.3.5.3.2 Semantics....154
7.3.3.6 QoS_Descriptor....155
7.3.3.6.1 Syntax....155
7.3.3.6.2 Semantics....155
7.3.3.7 extensionDescriptor....156
7.3.3.7.1 Syntax....156
7.3.3.7.2 Semantics....156
7.3.4 Usage of Object Descriptors....156
7.3.4.1 Association of Object Descriptors to Media Objects....156
7.3.4.2 Rules for Grouping Elementary Streams within one ObjectDescriptor....156
7.3.4.3 Usage of URLs in Object Descriptors....157
7.3.4.4 Object Descriptors and the MPEG-4 Session....158
7.3.4.4.1 MPEG-4 session....158
7.3.4.4.2 The initial Object Descriptor....158
7.3.4.4.3 Scope of objectDescriptorID and ES_ID labels....159
7.3.4.5 Session set up....159
7.3.4.5.1 Pre-conditions....159
7.3.4.5.2 Session set up procedure....159
7.3.4.5.2.1 Example....159
7.3.4.5.3 Set up for retrieval of a single Elementary Stream from a remote location. .160
7.4 Synchronization of Elementary Streams....161
7.4.1 Introduction....161

7.4.2 Access Unit Layer....162
7.4.2.1 AL-PDU Specification....162
7.4.2.1.1 Syntax....162
7.4.2.1.2 Semantics....163
7.4.2.2 AL-PDU Header Configuration....163
7.4.2.2.1 Syntax....163
7.4.2.2.2 Semantics....163
7.4.2.3 AL-PDU Header Specification....165
7.4.2.3.1 Syntax....165
7.4.2.3.2 Semantics....167
7.4.2.4 Clock Reference Stream....168
7.4.3 Elementary Stream Interface....168
7.4.4 Stream Multiplex Interface....169

7.5 Multiplexing of Elementary Streams....171
7.5.1 Introduction....171
7.5.2 FlexMux Tool....171
7.5.2.1 Simple Mode....171
7.5.2.2 MuxCode mode....172
7.5.2.3 FlexMux-PDU specification....173
7.5.2.3.1 Syntax....173
7.5.2.3.2 Semantics....173
7.5.2.3.3 Configuration for MuxCode Mode....174
7.5.2.3.3.1 Syntax....174
7.5.2.3.3.2 Semantics....174
7.5.2.4 Usage of MuxCode Mode....175
7.5.2.4.1 Example....175

7.6 Syntactic Description Language....176
7.6.1 Introduction....176
7.6.2 Elementary Data Types....176
7.6.2.1 Constant-Length Direct Representation Bit Fields....176
7.6.2.2 Variable Length Direct Representation Bit Fields....177
7.6.2.3 Constant-Length Indirect Representation Bit Fields....177
7.6.2.4 Variable Length Indirect Representation Bit Fields....178
7.6.3 Composite Data Types....179
7.6.3.1 Classes....179
7.6.3.2 Parameter types....180
7.6.3.3 Arrays....180
7.6.4 Arithmetic and Logical Expressions....181
7.6.5 Non-Parsable Variables....181
7.6.6 Syntactic Flow Control....182
7.6.7 Bult-In Operators....183
7.6.8 Scoping Rules....183

7.7 Object Content Information....185
7.7.1 Introduction....185
7.7.2 Object Content Information (OCI) Data Stream....185
7.7.3 Object Content Information (OCI) Syntax and Semantics....185
7.7.3.1 OCI Decoder Configuration....185
7.7.3.1.1 Syntax....185
7.7.3.1.2 Semantics....185
7.7.3.2 OCI_Events....186
7.7.3.2.1 Syntax....186
7.7.3.2.2 Semantics....186
7.7.3.3 Descriptors....186
7.7.3.3.1 OCI_Descriptor Class....186
7.7.3.3.1.1 Syntax....186
7.7.3.3.1.2 Semantics....187
7.7.3.3.2 Content classification descriptor....187

7.7.3.3.2.1 Syntax....187
7.7.3.3.2.2 Semantics....187
7.7.3.3.3 Key wording descriptor....187
7.7.3.3.3.1 Syntax....187
7.7.3.3.3.2 Semantics....188
7.7.3.3.4 Rating descriptor....188
7.7.3.3.4.1 Syntax....188
7.7.3.3.4.2 Semantics....188
7.7.3.3.5 Language descriptor....189
7.7.3.3.5.1 Syntax....189
7.7.3.3.5.2 Semantics....189
7.7.3.3.6 Short textual descriptor....189
7.7.3.3.6.1 Syntax....189
7.7.3.3.6.2 Semantics....189
7.7.3.3.7 Expanded textual descriptor....190
7.7.3.3.7.1 Syntax....190
7.7.3.3.7.2 Semantics....190
7.7.3.3.8 Name of content creators descriptor....191
7.7.3.3.8.1 Syntax....191
7.7.3.3.8.2 Semantics....191
7.7.3.3.9 Date of content creation descriptor....192
7.7.3.3.9.1 Syntax....192
7.7.3.3.9.2 Semantics....192
7.7.3.3.10 Name of OCI creators descriptor....192
7.7.3.3.10.1 Syntax....192
7.7.3.3.10.2 Semantics....192
7.7.3.3.11 Date of OCI creation descriptor....193
7.7.3.3.11.1 Syntax....193
7.7.3.3.11.2 Semantics....193
7.7.4 193
Annex: Conversion between time and date conventions....194

7.8 Profiles....196
7.8.1 Scene Description Profiles....196
7.8.1.1 2D profile....196
7.8.1.2 3D profile....196
7.8.1.3 VRML profile....196
7.8.1.4 Complete profile....196
7.8.1.5 Audio profile....196

7.9 Elementary Streams for Upstream Control Information....197

B.1 Time base reconstruction....199
B.1.1 Adjusting the receivers OTB....199
B.1.2 Mapping Time Stamps to the STB....199
B.1.3 Adjusting the STB to an OTB....200
B.1.4 System Operation without Object Time Base....200

B.2 Temporal aliasing and audio resampling....200

B.3 Reconstruction of a synchronised audiovisual scene: a walkthrough....200

C.1 ISO/IEC 14496 content embedded in ISO/IEC 13818-1 Transport Stream 202
C.1.1 Introduction....202
C.1.2 IS 14496 Stream Indication in Program Map Table....202
C.1.3 Object Descriptor and Stream Map Table Encapsulation....204
C.1.4 Scene Description Stream Encapsulation....205
C.1.5 Audio Visual Stream Encapsulation....206
C.1.6 Framing of AL-PDU and FM-PDU into TS packets....206
C.1.6.1 Use of MPEG-2 TS Adaptation Field....206

C.1.6.2 Use of MPEG-4 PaddingFlag and PaddingBits....206

C.2 MPEG-4 content embedded in MPEG-2 DSM-CC Data Carousel....208
C.2.1 Scope....208
C.2.2 Introduction....208
C.2.3 DSM-CC Data Carousel....208
C.2.4 General Concept....208
C.2.5 Design of Broadcast Applications....210
C.2.5.1 Program Map Table....210
C.2.5.2 FlexMux Descriptor....212
C.2.5.3 Application Signaling Channel and Data Channels....212
C.2.5.4 Stream Map Table....213
C.2.5.5 TransMux Channel....215
C.2.5.6 FlexMux Channel....215
C.2.5.7 Payload....217

C.3 MPEG-4 content embedded in a Single FlexMux Stream....218
C.3.1 Initial Object Descriptor....218
C.3.2 Stream Map Table....218
C.3.2.1 Syntax....218
C.3.2.2 Semantics....218
C.3.3 Single FlexMux Stream Payload....219

D.1 Introduction....220

D.2 Bitstream Syntax....220
D.2.1 View Dependent Object....220
D.2.2 View Dependent Object Layer....221

D.3 Bitstream Semantics....221
D.3.1 View Dependent Object....221
D.3.2 View Dependent Object Layer....222

D.4 Decoding Process of a View-Dependent Object....222
D.4.1 Introduction....222
D.4.2 General Decoding Scheme....223
D.4.2.1 View-dependent parameters computation....223
D.4.2.2 VD mask computation....223
D.4.2.3 Differential mask computation....223
D.4.2.4 DCT coefficients decoding....223
D.4.2.5 Texture update....223
D.4.2.6 IDCT....223
D.4.2.7 Rendering....223
D.4.3 Computation of the View-Dependent Scalability parameters....224
D.4.3.1 Distance criterion:....225
D.4.3.2 Rendering criterion:....225
D.4.3.3 Orientation criteria:....225
D.4.3.4 Cropping criterion:....226
D.4.4 VD mask computation....226
D.4.5 Differential mask computation....227
D.4.6 DCT coefficients decoding....228
D.4.7 Texture update....228
D.4.8 IDCT....228

List of Figures

Figure 0-1: Processing stages in an audiovisual terminal 2

Figure 7-2: Systems Decoder Model.....8

Figure 7-3: Flow diagram for the Systems Decoder Model...13

Figure 7-4: An example of an MPEG-4 multimedia scene.....15

Figure 7-5: Logical structure of the scene.....16

Figure 7-6: A complete scene graph example. We see the hierarchy of 3 different scene graphs: the 2D graphics scene graph, 3D graphics scene graph, and the layers 3D scene graphs. As shown in the picture, the 3D layer-2 view the same scene as 3D-layer1, but the viewpoint may be different. The 3D object-3 is a Appearance node that uses the 2D-Scene 1 as a texture node.....18

Figure 7-7: 2D Coordinate System.....19

Figure 7-8: 3D Coordinate System.....19

Figure 7-9: Standard Units.....20

Figure 7-10: Media start times and CTS.....25

Figure 7-11: BIFS-Update Commands.....29

Figure 7-12: Encoding dynamic fields.....56

Figure 7-13: An example FIG.....92

Figure 7-14: Three Layer2D and Layer3D examples. Layer2D are signaled by a plain line, Layer3D with 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 overlaping Layer3D.....109

Figure 7-15: A Composite2DTexture example. The 2D scene is projected on the 3D cube.....110

Figure 7-16: A Composite3Dtexture example: The 3D view of the earth is projected onto the 3D cube.....111

Figure 7-17: A CompositeMap example: The 2D scene as defined in Fig. yyy composed of an image, a logo, and a text, is drawn in the local X,Y plane of the back wall.....112

Figure 7-18: Session setup example.....160
Figure 7-19 Systems Layers.....162
Figure 7-20 : Structure of FlexMux-PDU in simple mode...172
Figure 7-21: Structure of FlexMux-PDU in MuxCode mode173
Figure 7-22 Example for a FlexMux-PDU in MuxCode mode 175
Figure 7-23: Conversion routes between Modified Julian Date (MJD) and Coordinated Universal Time (UTC).....194
Figure C-24 : An example of stuffing for the MPEG-2 TS packet.....207
Figure D-25: General Decoding Scheme of a View-Dependent Object.....224
Figure D-26: Definition of a and b angles.....225
Figure D-27: Definition of Out of Field of View cells.....226
Figure D-28: VD mask of an 8x8 block using VD parameters 227
Figure D-29: Differential mask computation scheme.....227
Figure D-30: Texture update scheme.....228

List of Tables

Table 7-1: Alignment Constraints.....77
Table 7-2: Distribution Constraints.....78
Table 7-3: List of Descriptor Tags.....148
Table 7-4: profileAndLevelIndication Values.....151
Table 7-5: streamType Values.....152
Table 7-6: type_of_content Values.....154
Table 7-7: type_of_content_identifier Values.....154
Table 7-8: Predefined QoS_Descriptor Values.....155
Table 7-9: descriptorTag Values.....156
Table 7-10: Overview of predefined ALConfigDescriptor values.....163
Table 7-11: Detailed predefined ALConfigDescriptor values 164
Table C-12 : Transport Stream Program Map Section of ISO/IEC 13818-1.....203
Table C-13 : ISO/IEC 13818-1 Stream Type Assignment.....203
Table C-14 : OD SMT Section.....204
Table C-15 : Stream Map Table.....204
Table C-16 : Private section for the BIFS stream.....205
Table C-17: Transport Stream Program Map Section.....210
Table C-18: Association Tag Descriptor.....211
Table C-19: DSM-CC Section.....212
Table C-20: DSM-CC table_id Assignment.....213
Table C-21: DSM-CC Message Header.....213
Table C-22: Adaptation Header.....214
Table C-23: DSM-CC Adaptation Types.....214
Table C-24: DownloadInfoIndication Message.....214
Table C-25: DSM-CC Download Data Header.....216
Table C-26: DSM-CC Adaptation Types.....216
Table C-27: DSM-CC DownloadDataBlock() Message.....216

λ 0. Introduction

The Systems part of the Committee Draft of International Standard describes a system for communicating audiovisual information. This information consists of the coded representation of natural or synthetic objects (media objects) that can be manifested audibly and/or visually. At the sending side, audiovisual information is compressed, composed, and multiplexed in one or more coded binary streams that are transmitted. At the receiver these streams are demultiplexed, decompressed, composed, and presented to the end user. The end user may have the option to interact with the presentation. Interaction information can be processed locally or transmitted to the sender. This specification provides the semantic and syntactic rules that integrate such natural and synthetic audiovisual information representation.

The Systems part of the Committee Draft of International Standard specifies the following tools: a terminal model for time and buffer management; a coded representation of interactive audiovisual scene information; a coded representation of identification of audiovisual streams and logical dependencies between stream information; a coded representation of synchronization information; multiplexing of individual components in one stream; and a coded representation of audiovisual content related information. These various elements are described functionally in this clause and specified in the normative clauses that follow.

0.1 Architecture

The information representation specified in the Committee Draft of International Standard allows the presentation of an interactive audiovisual scene from coded audiovisual information and associated scene description information. The presentation can be performed by a standalone system, or part of a system that needs to utilize information represented in compliance with this Committee Draft of International Standard. In both cases, the receiver will be generically referred to as an “audiovisual terminal” or just “terminal.”

The basic operations performed by such a system are as follows. Initial information that provides handles to Elementary Streams is known as premises by the terminal. Part 6 of this Committee Draft of International Standard provides for the specification to resolve these premises as well as the interface (TransMux Interface) with the storage or transport medium. Some of these elementary streams may have been grouped together using the FlexMux multiplexing tool (FlexMux Layer) described in this Committee Draft of International Standard.

Elementary streams contain the coded representation of the content data: scene description information (BIFS – Binary Format for Scenes – elementary streams), audio information or visual information (audio or visual elementary streams), content related information (OCI elementary streams) as well as additional data sent to describe the type of the content for each individual stream (elementary stream Object Descriptors). Elementary streams may be downchannel streams (sender to receiver) or upchannel streams (receiver to sender).

Elementary streams are decoded (Compression Layer), composed according to the scene description information and presented to the terminal’s presentation device(s). All these processes are synchronized according to the terminal decoding model (SDM, Systems Decoder Model) and the synchronization information provided at the AcessUnit Layer. In cases where the content is available in random access storage facilities, additional information may be present in the stream in order to allow random access functionality.

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

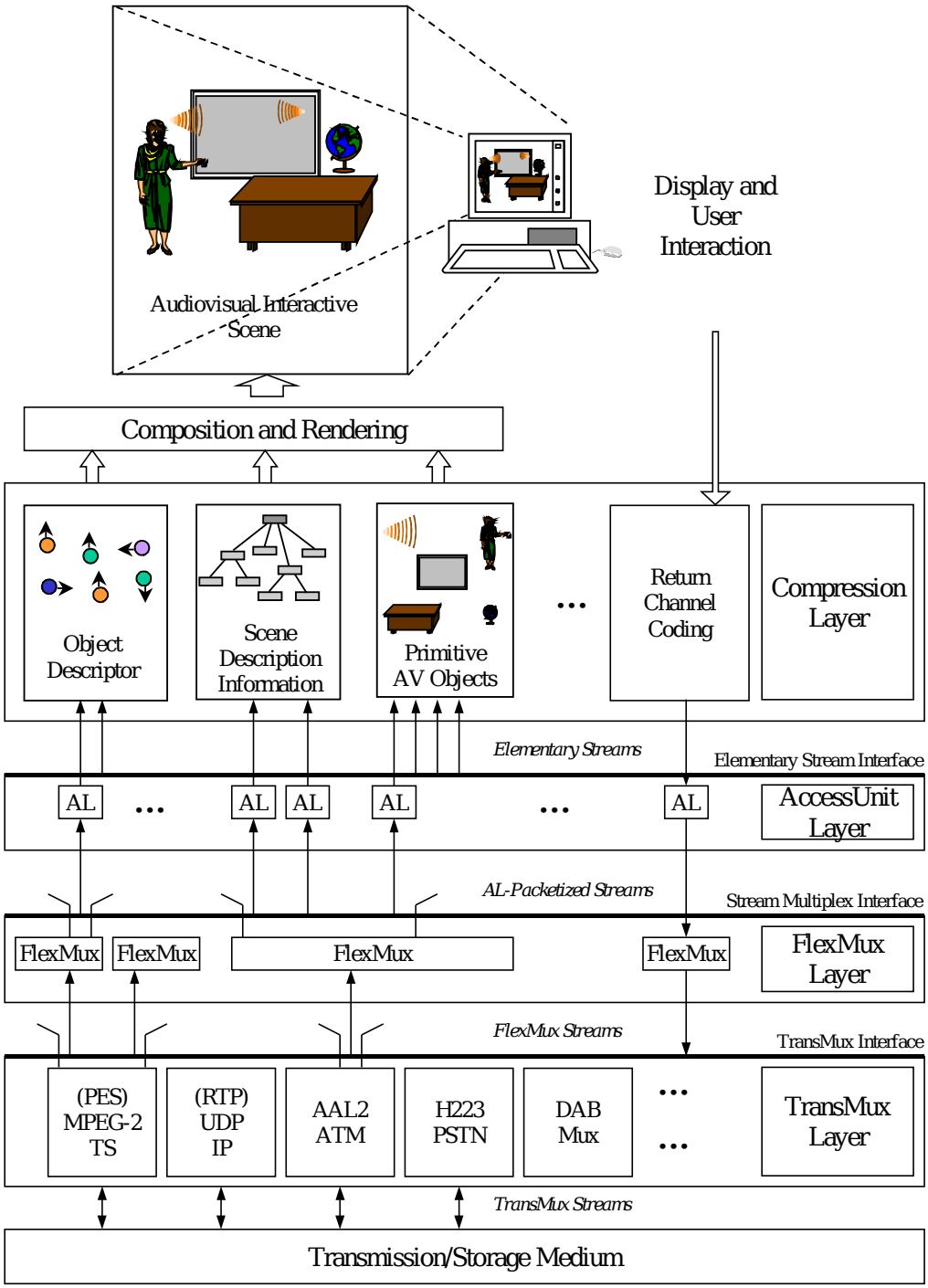


Figure 0-1: Processing stages in an audiovisual terminal

0.2 Systems Decoder Model

The purpose of the Systems Decoder Model (SDM) is to provide an abstract view of the behavior of a terminal complying to this Committee Draft of International Standard. It can be used by the sender to predict how the receiver will behave in terms of buffer management and synchronization when reconstructing the audiovisual information that composes the session. The Systems Decoder Model includes a timing model and a buffer model.

l0.2.1 Timing Model

The System Timing Model enables the receiver to recover the notion of time according to the sender in order to perform certain events at specified instants in time, such as decoding data units or synchronization of audiovisual information. This requires that the transmitted data streams contain implicit or explicit timing information. A first set of timing information, the clock references, is used to convey an encoder time base to the decoder, while a second set, the time stamps, convey the time (in units of an encoder time base) for specific events such as the desired decoding or composition time for portions of the encoded audiovisual information.

l0.2.2 Buffer Model

The Systems Buffering Model enables the sender to monitor the minimum buffer resources that are needed to decode each individual Elementary Stream in the session. These required buffer resources are conveyed to the receiver by means of Elementary Streams Descriptors before the start of the session so that it can decide whether it is capable of handling this session. The model assumptions further allow the sender to manage a known amount of receiver buffers, and schedule data transmission accordingly.

0.3 FlexMux and TransMux Layer

The demultiplexing process is not part of this specification. This Committee Draft of International Standard specifies just the interface to the demultiplexer. It is termed Stream Multiplex Interface and may be embodied by the DMIF Application Interface specified in Part 6 of this Committee Draft of International Standard. It is assumed that a diversity of suitable delivery mechanisms exists below this interface. Some of them are listed in . These mechanisms serve for transmission as well as storage of streaming data. A simple tool for multiplexing, FlexMux, that addresses the specific MPEG-4 needs of low delay and low overhead multiplexing is specified and may optionally be used depending on the properties that a specific delivery protocol stack offers.

0.4 AccessUnit Layer

The Elementary Streams are the basic abstraction of any streaming data source. They are packaged into AL-packetized Streams when they arrive at the Stream Multiplex Interface. This allows it on the Access Unit Layer to extract the timing information that is necessary to enable a synchronized decoding and, subsequently, composition of the Elementary Streams.

0.5 Compression Layer

Decompression recovers the data of a media object from its encoded format (syntax) and performs the necessary operations to reconstruct the original media object (semantics). The reconstructed media object is made available to the composition process for potential use during scene rendering. Composition and rendering are outside the scope of in this Committee Draft of International Standard. The coded representation of audio information and visual information are described in Parts 2 and 3, respectively of this Committee Draft of International Standard. The following subclauses provide for a functional description of the content streams specified in the part of Committee Draft of International Standard.

l0.5.1 Object Descriptor Elementary Streams

In order to access the content of Elementary Streams, the streams must be properly identified. The identification information is carried in a specific stream by entities called Object Descriptors. Identification of Elementary Streams includes information about the source of the conveyed media data, in form of a URL or a numeric identifier, as well as the encoding format, the configuration for the Access Unit Layer packetization of the Elementary Stream and intellectual property information. Optionally more information can be associated

to a media object, most notably Object Content Information. The Object Descriptors' unique identifiers (objectDescriptorIDs) are used to resolve the association between media objects.

l0.5.2 Scene Description Streams

Scene description addresses the organization of audiovisual objects in a scene, in terms of both spatial and temporal positioning. This information allows the composition and rendering of individual audiovisual objects after they are reconstructed by their respective decoders. This specification, however, does not mandate particular composition or rendering algorithms or architectures; these are considered implementation-dependent.

The scene description is represented using a parametric description (BIFS, Binary Format for Scenes). The parametric description is constructed as a coded hierarchy of nodes with attributes and other information (including event sources and targets). The scene description can evolve over time by using coded scene description updates.

In order to allow active user involvement with the presented audiovisual information, this specification provides support for interactive operation. Interactive features are integrated with the scene description information, which defines the relationship between sources and targets of events. It does not, however, specify a particular user interface or a mechanism that maps user actions (e.g., keyboard key pressed or mouse movements) to such events. Local or client-side interactivity is provided via the ROUTES and SENSORS mechanism of BIFS. Such an interactive environment does not need an upstream channel. This Committee Draft of International Standard also provides means for client-server interactive sessions with the ability to set up upchannel elementary streams.

l0.5.3 Upchannel Streams

Media Objects may require upchannel stream control information to allow for interactivity. An Elementary Stream flowing from receiver to transmitter is treated the same way as any downstream Elementary Stream as described in . The content of upstream control streams is specified in the same part of this specification that defines the content of the downstream data for this Media Object. For example, control streams for video compression algorithms are defined in 14496-2.

l0.5.4 Object Content Information Streams

The Object Content Information (OCI) stream carries information about the audiovisual objects. This stream is organized in a sequence of small, synchronized entities called events that contain information descriptors. The main content descriptors are: content classification descriptors, keyword descriptors, rating descriptors, language descriptors, textual descriptors, and descriptors about the creation of the content. These streams can be associated to other media objects with the mechanisms provided by the Object Descriptor.

λ 1. Scope

This part of Committee Draft of International Standard 14496 has been developed to support the combination of audiovisual information in the form of natural or synthetic, aural or visual, 2D and 3D objects coded with methods defined in Parts 1, 2 and 3 of this Committee Draft of International Standard within the context of content-based access for digital storage media, digital audiovisual communication and other applications. The Systems layer supports seven basic functions:

1. the coded representation of an audiovisual scene composed of multiple media objects (i.e., their spatio-temporal positioning), including user interaction;
2. the coded representation of content information related to media objects;
3. the coded representation of identification of audiovisual streams and logical dependencies between streams information, including information for the configuration of the receiving terminal;

4. the coded representation of synchronization information for timing identification and recovery mechanisms;
5. the support and the coded representation of return channel information;
6. the interleaving of multiple audiovisual object streams into one stream (multiplexing);
7. the initialization and continuous management of the receiving terminal's buffers.

λ 2. Normative References

The following ITU-T Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Committee Draft of International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Committee Draft of International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau maintains a list of currently valid ITU-T Recommendations.

λ 3. Additional References

[1][1] ISO/IEC International Standard 13818-1 (MPEG-2 Systems), 1994.

[2][2] ISO/IEC 14472-1 Draft International Standard, Virtual Reality Modeling Language (VRML), 1997.

[3][3] ISO 639, Code for the representation of names of languages, 1988.

[4][4] ISO 3166-1, Codes for the representation of names of countries and their subdivisions – Part 1: Country codes, 1997.

[5][5] The Unicode Standard, Version 2.0, 1996.

λ 4. Definitions

1. **Access Unit (AU):** A logical sub-structure of an *Elementary Stream* to facilitate random access or bitstream manipulation. All consecutive data that refer to the same decoding time form a single Access Unit.
2. **Access Unit Layer (AL):** A layer to adapt *Elementary Stream* data for the communication over the Stream Multiplex Interface. The AL carries the coded representation of time stamp and clock reference information., provides *AL-PDU* numbering and byte alignment of *AL-PDU Payload*. The Adaptation Layer syntax is configurable and can eventually be empty.
3. **Access Unit Layer Protocol Data Unit (AL-PDU)**: The smallest protocol unit exchanged between peer AL Entities. It consists of *AL-PDU Header* and *AL-PDU Payload*.
4. **Access Unit Layer Protocol Data Unit Header (AL-PDU Header)**: Optional information preceding the *AL-PDU Payload*. It is mainly used for Error Detection and Framing of the AL-PDU Payload. The format of the AL-PDU Header is determined through the ALConfigDescriptor conveyed in an *Object Descriptor*.
5. **Access Unit Layer Protocol Data Unit Payload (AL-PDU Payload)**: The data field of an *AL-PDU* containing *Elementary Stream* data.
6. **Media Object**: A Media object is a representation of a natural or synthetic object that can be manifested aurally and/or visually.
7. **Audiovisual Scene (AV Scene)** : An AV Scene is set of *media objects* together with *scene description* information that defines their spatial and temporal positioning, including user interaction.
8. **Buffer Model**: This model enables a terminal complying to this specification to monitor the minimum buffer resources that are needed to decode a session. Information on the required resources may be conveyed to the *decoder* before the start of the session.

9. **Composition**: The process of applying scene description information in order to identify the spatio-temporal positioning of audiovisual objects.
10. **Elementary Stream (ES)**: A sequence of data that originates from a single producer in the transmitting *Terminal* and terminates at a single recipient, e. g., *Media Object*s.
11. **FlexMux Channel:** The sequence of data within a *FlexMux Stream* that carries data from one *Elementary Stream* packetized in a sequence of *AL-PDUs*.
12. **FlexMux Protocol Data Unit (FlexMux-PDU)**: The smallest protocol unit of a *FlexMux Stream* exchanged between peer FlexMux Entities. It consists of *FlexMux-PDU Header* and *FlexMux-PDU Payload*. It carries data from one *FlexMux Channel*.
13. **FlexMux Protocol Data Unit Header (FlexMux-PDU Header):** Information preceding the *FlexMux-PDU* Payload. It identifies the *FlexMux Channel*(s) to which the payload of this FlexMux-PDU belongs.
14. **FlexMux Protocol Data Unit Payload (FlexMux-PDU Payload):** The data field of the *FlexMux-PDU*, consisting of one or more AL-PDUs.
15. **FlexMux Stream**: A sequence of *FlexMux-PDUs* originating from one or more *FlexMux Channels* forming one data stream.
16. **Terminal**: A terminal here is defined as a system that allows *Presentation* of an interactive *Audiovisual Scene* from coded audiovisual information. It can be a standalone application, or part of a system that needs to use content complying to this specification.
17. **Object Descriptor (OD)**: A syntactic structure that provides for the identification of elementary streams (location, encoding format, configuration, etc.) as well as the logical dependencies between elementary streams.
18. **Object Time Base (OTB):** The Object Time Base (OTB) defines the notion of time of a given *Encoder*. All *Timestamps* that the encoder inserts in a coded *AV object* data stream refer to this *Time Base*.
19. **Quality of Service (QoS)** - The performance that an *Elementary Stream* requests from the delivery channel through which it is transported, characterized by a set of parameters (e.g., bit rate, delay jitter, bit error rate).
20. **Random Access:** The capability of reading, decoding, or composing a coded *bitstream* starting from an arbitrary point.
21. **Scene Description**: Information that describes the spatio-termporal positioning of *media objects* as well as user interaction.
22. **Session**: The, possibly interactive, communication of the coded representation of an *audiovisual scene* between two *terminals*. A uni-directional session corresponds to a *program* in a broadcast application.
23. **Syntactic Description Language (SDL)**: A language defined by this specification and which allows the description of a bitstream's syntax.
24. **Systems Decoder Model**: This model is part of the *Systems Receiver Model*, and provides an abstract view of the behavior of the *MPEG-4 Systems*. It consists of the *Buffering Model,* and the *Timing Model*.
25. **System Time Base (STB)**: The Systems Time Base is the terminal's *Time Base*. Its resolution is implementation-dependent. All operations in the terminal are performed according to this time base.
26. **Time Base**: A time base provides a time reference.
27. **Timing Model**: Specifies how timing information is incorporated (explicitly or implicitly) in the coded representation of information, and how it can be recovered at the terminal.
28. **Timestamp**: An information unit related to time information in the *Bitstream* (*see Composition Timestamp* and *Decoding Timestamp*).
29. **User Interaction:** The capability provided to a user to initiate actions during a session.
30. **TransMux:** A generic abstraction for delivery mechanisms able to store or transmit a number of multiplexed Elementary Streams. This specification does not specify a TransMux layer.

λ 5. Abbreviations and Symbols

The following symbols and abbreviations are used in this specification.

1. APS - AL-packetized Stream
2. AL - Access Unit Layer
3. AL-PDU - Access Unit Layer Protocol Data Unit
4. AU - Access Unit
5. BIFS - Binary Format for Scene
6. CU - Composition Unit
7. CM - Composition Memory
8. CTS - Composition Time Stamp
9. DB - Decoding Buffer
10. DTS - Decoding Time Stamp
11. ES - Elementary Stream
12. ES_ID - Elementary Stream Identification
13. IP - Intellectual Property
14. IPI - Intellectual Property Information
15. OCI - Object Content Information
16. OCR - Object Clock Reference
17. OD - Object Descriptor
18. OTB - Object Time Base
19. PDU - Protocol Data Unit
20. PLL - Phase locked loop
21. QoS - Quality of Service
22. SDL - Syntactic Description Language
23. STB - System Time Base
24. URL - Universal Resource Locator

λ 6. Conventions

λ 6.1 Syntax Description

For the purpose of unambiguously defining the syntax of the various bitstream components defined by the normative parts of this Committee Draft of International Standard 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 should be placed in a serialized bitstream. The definition of the language is provided in Subclause .

λ 7. Specification

λ 7.1 Systems Decoder Model

λ 7.1.1 Introduction

The purpose of the Systems Decoder Model (SDM) is to provide an abstract view of the behavior of a terminal complying to this Committee Draft of International Standard. It can be used by the sender to predict how the receiver will behave in terms of buffer management and synchronization when reconstructing the audiovisual information that composes the session. The Systems Decoder Model includes a timing model and a buffer model.

The Systems Decoder Model specifies the access to demultiplexed data streams via the DMIF Application Interface, Decoding Buffers for compressed data for each Elementary Stream, the behavior of media object decoders, composition memory for decompressed data for each media object and the output behavior towards the compositor, as outlined in Figure 7-2. Each Elementary Stream is attached to one single Decoding Buffer. More than one Elementary Stream may be connected to a single media object decoder (e.g.: scaleable media decoders).

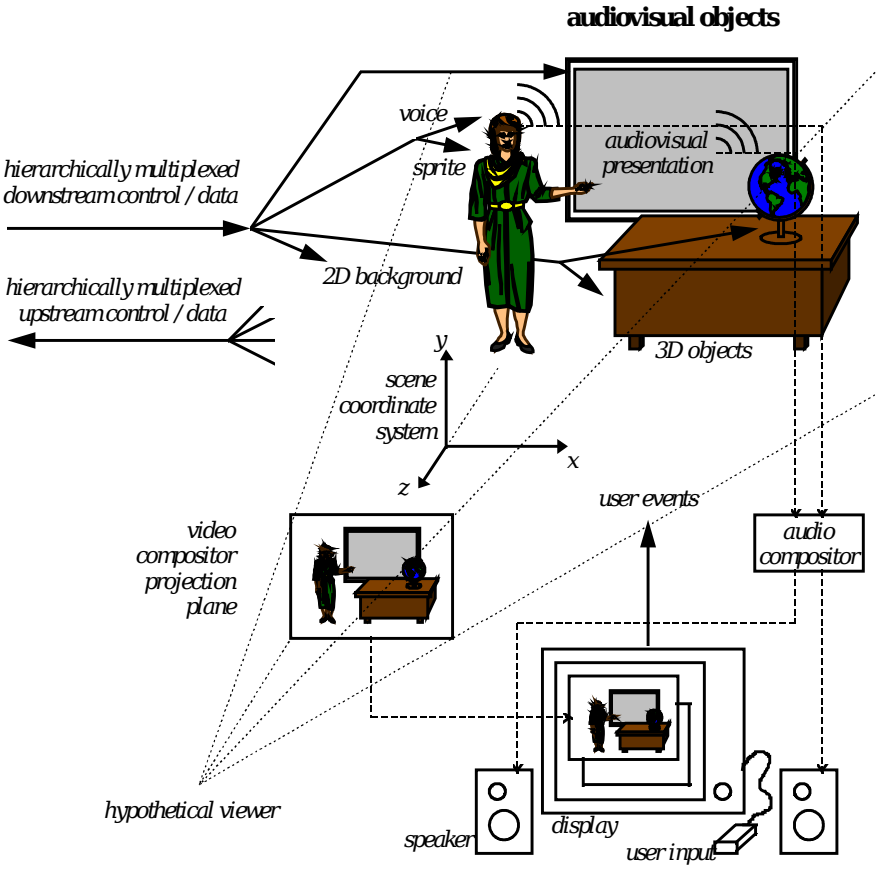


Figure 7-2: Systems Decoder Model

l7.1.2 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 7-2.

λ 7.1.2.1 DMIF Application Interface (DAI)

For the purpose of the Systems Decoder Model, the DMIF Application Interface, which encapsulates the demultiplexer, is a black box that provides multiple handles to streaming data and fills up Decoding Buffers with this data. The streaming data received through the DAI consists of AL-packetized Streams.

λ 7.1.2.2 AL-packetized Stream (APS)

An AL-packetized Stream (AL=Access Unit Layer) consists of a sequence of packets, according to the syntax and semantics specified in Subclause 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. APS data enters the Decoding Buffers.

λ 7.1.2.3 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. Any further structure of the data in an Elementary Stream is not visible for the purpose of the Systems Decoder Model. Access Units are conveyed by AL-packetized streams and are received by the Decoding Buffer. Access Units with the necessary side information (e.g. time stamps) are taken from the Decoding Buffer through the Elementary Stream Interface.

Note: An MPEG-4 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 Subclause . This allows the encoder to dispatch partial AUs immediately as they are generated during the encoding process.

λ 7.1.2.4 Decoding Buffer (DB)

The Decoding Buffer is a receiver buffer that contains Access Units. The Systems Buffering Model enables the sender to monitor the minimum Decoding Buffer resources that are needed during a session.

λ 7.1.2.5 Elementary Streams (ES)

Streaming data received at the output of a Decoding Buffer, independent of its content, is considered as Elementary Stream for the purpose of this specification. The integrity of an Elementary Stream is preserved from end to end between two systems. Elementary Streams are produced and consumed by Compression Layer entities (encoder, decoder).

λ 7.1.2.6 Elementary Stream Interface (ESI)

The Elementary Stream Interface models the exchange of Elementary Stream data and associated control information between the Compression Layer and the Access Unit Layer. At the receiving terminal the ESI is located at the output of the Decoding Buffer. The ESI is specified in Subclause .

λ 7.1.2.7 Media Object Decoder

For the purpose of this model, the media object decoder is a black box that takes Access Units out of the Decoding Buffer at precisely defined points in time and fills up the Composition Memory with Composition Units. A Media Object Decoder may be attached to several Decoding Buffers

λ 7.1.2.8 Composition Units (CU)

Media object decoders produce Composition Units from Access Units. An Access Unit corresponds to an integer number of Composition Units. Composition Units are received by or taken from the Composition Memory.

λ 7.1.2.9 Composition Memory (CM)

The Composition Memory is a random access memory that contains Composition Units. The size of this memory is not normatively specified.

λ 7.1.2.10 Compositor

The compositor is not specified in this Committee Draft of International Standard. The Compositor takes Composition Units out of the Composition Memory and either composits and presents them or skips them. This behavior is not relevant within the context of the model. Subclause details the specifics of which Composition Unit is available to the Compositor at any instant of time.

l7.1.3 Timing Model Specification

The timing model relies on two well-known concepts to synchronize media objects conveyed by one or more Elementary Streams. The concept of a clock and associated clock reference time stamps are used to convey the notion of time of an encoder to the receiving terminal. Time stamps are used to indicate when an event shall happen in relation to a known clock. These time events are attached to Access Units and Composition Units. The semantics of the timing model is defined in the subsequent subclauses. The syntax to convey timing information is specified in Subclause .

Note: This model is designed for rate-controlled (“push”) applications.

λ 7.1.3.1 System Time Base (STB)

The System Time Base (STB) defines the receiving terminal's notion of time. The resolution of this 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 compliant receiver terminals operate on one single STB.

λ 7.1.3.2 Object Time Base (OTB)

The Object Time Base (OTB) defines the notion of time of a given media object encoder. The resolution of this OTB can be selected as required by the application or is governed by a profile . All time stamps that the encoder inserts in a coded media object data stream refer to this time base. The OTB of an object is known at the receiver either by means of information inserted in the media stream, as specified in Subclause , or by indication that its time base is slaved to a time base conveyed with another stream, as specified in Subclause .

Note: Elementary streams may be created for the sole purpose of conveying time base information.

Note: The receiver terminals’ System Time Base need not be locked to any of the Object Time Bases in an MPEG-4 session.

λ 7.1.3.3 Object Clock Reference (OCR)

A special kind of time stamps, Object Clock Reference (OCR), are used to convey the OTB to the media object decoder. The value of the OCR corresponds to the value of the OTB at the time the transmitting terminal generates the Object Clock Reference time stamp. OCR time stamps are placed in the AL-PDU header as described in Subclause . The receiving terminal shall extract and evaluate the OCR when its first byte enters the Decoding Buffer in the receiver system. OCRs shall be conveyed at regular intervals, with the minimum frequency at which OCRs are inserted being application-dependent.

λ 7.1.3.4 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.

This point in time is implicitly known, if the (constant) temporal distance between successive Access Units is indicated in the setup of the Elementary Stream (see Subclause). Otherwise it is conveyed by a decoding time stamp (DTS) placed in the Access Unit Header. It contains the value of the OTB at the nominal decoding time of the Access Unit.

Decoding Time Stamps shall not be present for an Access Unit unless the DTS value is different from the CTS value. Presence of both time stamps in an AU may indicate a reversal between coding order and composition order.

λ 7.1.3.5 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. However, the CU is already available in the Composition Memory for use by the decoder (e.g. prediction) at the time indicated by DTS of the associated AU, since the SDM assumes instantaneous decoding.

This point in time is implicitly known, if the (constant) temporal distance between successive Composition Units is indicated in the setup of the Elementary Stream. Otherwise it is conveyed by a composition time stamp (CTS) placed in the Access Unit Header. It contains the value of the OTB at the nominal composition time of the Composition Unit.

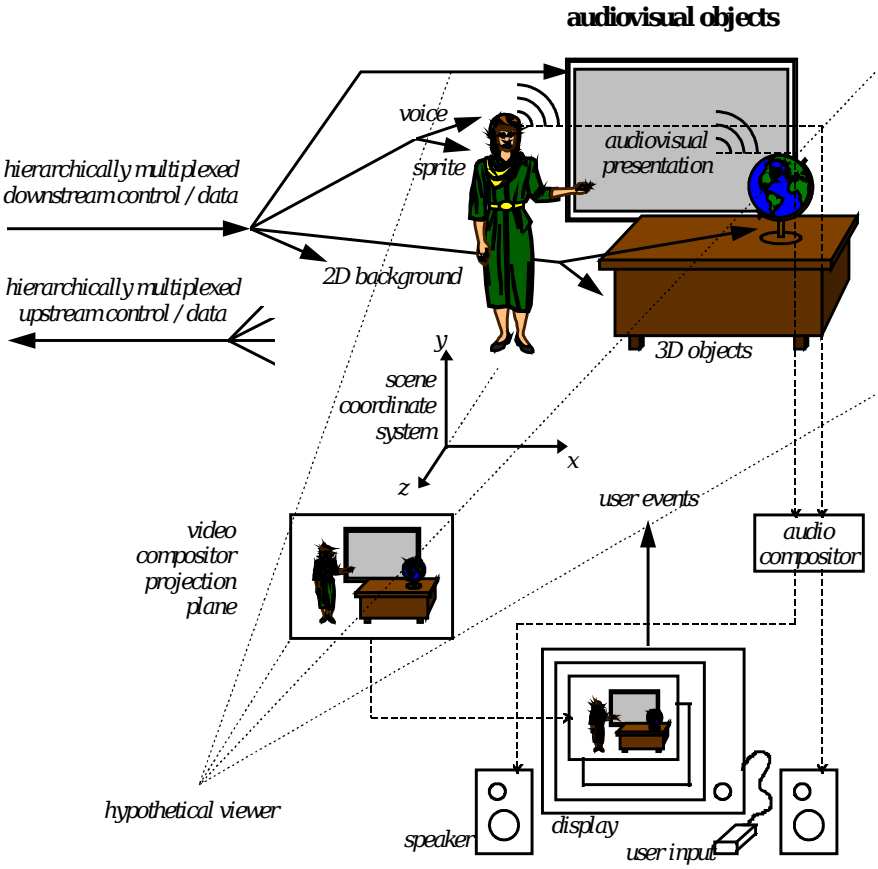
The current CU is available to the compositor between its composition time and the composition time of the subsequent CU. If a subsequent CU does not exist, the current CU becomes unavailable at the end of the life time of its Media Object.

λ 7.1.3.6 Occurrence of timing information in Elementary Streams

The frequency at which DTS, CTS and OCR values are to be inserted in the bitstream is application and profile dependent.

λ 7.1.3.7 Example

The example below illustrates the arrival of two Access Units at the Systems Decoder. Due to the constant delay assumption of the model, the arrival times correspond to the point in time when the respective AU have been sent by the transmitter. This point in time must be selected by the transmitter such that the Decoder Buffer never overflows nor underflows. At DTS an AU is instantaneously decoded and the resulting CU(s) are placed in the Composition Memory and remain there until the subsequent CU(s) arrive.



l7.1.4 Buffer Model Specification

l7.1.4.1 Elementary decoder model

The following simplified model is assumed for the purpose of the buffer model specification. Each Elementary Stream is regarded separately. The definitions as given in the previous subclause remain.

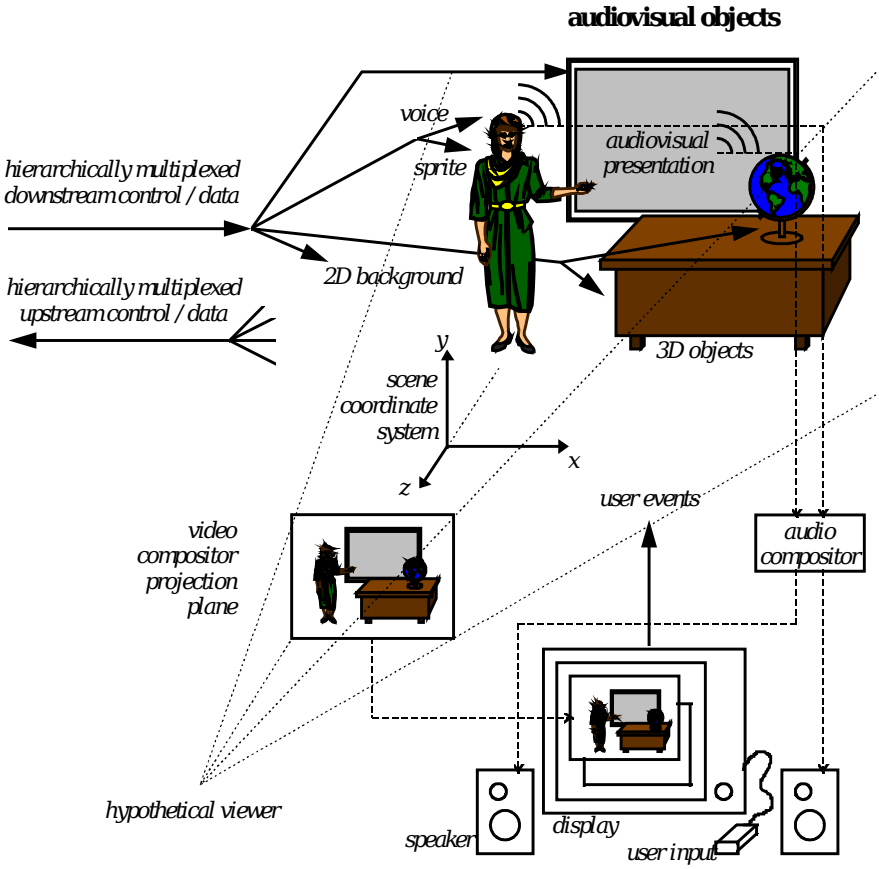


Figure 7-3: Flow diagram for the Systems Decoder Model

λ 7.1.4.2 Assumptions

λ *7.1.4.2.1 Constant end-to-end delay*

Media objects being presented and transmitted in real time, have a timing model in which the end-to-end delay from the encoder input to the decoder output is a constant. This delay is the sum of encoding, encoder buffering, multiplexing, communication or storage, demultiplexing, decoder buffering and decoding delays.

Note that the decoder is free to add a temporal offset (delay) to the absolute values of all time stamps if it copes with the additional buffering needed. However, the temporal difference between two time stamps, that determines the temporal distance between the associated AU or CU, respectively, has to be preserved for real-time performance.

7.1.4.2.2 Demultiplexer

The end-to-end delay between multiplexer output and demultiplexer input is constant.

7.1.4.2.3 Decoding Buffer

- The needed Decoding Buffer size is known by the sender and conveyed to the receiver as specified in Subclause .
- The size of the Decoding Buffer is measured in bytes.
- Decoding Buffers are filled at the rate given by the maximum bit rate for this Elementary Stream if data is available from the demultiplexer and else with rate zero. Maximum bit rate is conveyed in the decoder configuration during set up of each Elementary Stream (see Subclause).

- AL-PDUs are received from the demultiplexer. The AL-PDU Headers are removed at the input to the Decoding Buffers.

7.1.4.2.4 Decoder

- The decoding time is assumed to be zero for the purposes of the Systems Decoder Model.

7.1.4.2.5 Composition Memory

- The size of the Composition Memory is measured in Composition Units.
- The mapping of AU to CU is known implicitly (by the decoder) to the sender and the receiver.

7.1.4.2.6 Compositor

- The composition time is assumed to be zero for the purposes of the Systems Decoder Model.

7.1.4.3 Managing Buffers: A Walkthrough

The model is assumed to be used in a “push” scenario. In case of interactive applications where non-real time content is to be transmitted, flow control by suitable signaling may be established to request Access Units at the time they are needed at the receiver. This is currently not further specified in this document.

The behavior of the SDM elements are modeled as follows:

- The sender signals the required buffer resources to the receiver before starting the transmission. This is done as specified in Subclause either explicitly by requesting buffer sizes for individual Elementary Streams or implicitly by specification of an MPEG-4 profile and level. The buffer size is measured in bytes for the DB.
- The sender models the buffer behavior by making the following assumptions :
  - The Decoding Buffer is filled at the maximum bitrate for this Elementary Stream if data is available.
  - At DTS, an AU is instantaneously decoded and removed from DB.
  - At DTS, a known amount of CUs corresponding to the AU are put in the Composition Memory,
  - The current CU is available to the compositor between its composition time and the composition time of the subsequent CU. If a subsequent CU does not exist, the CU becomes unavailable at the end of lifetime of its Media object.

With these model assumptions the sender may freely use the space in the buffers. For example it may transfer data for several Access Units of a non-real time stream to the receiver and pre-store them in the DB some time before they have to be decoded if there is sufficient space. Then the full channel bandwidth may be used to transfer data of a real time stream just in time afterwards. The Composition Memory may be used, for example, as a reordering buffer to contain decoded P-frames which are needed by the video decoder for the decoding of intermediate B-frames before the arrival of the CTS for the P-frame.

7.2 Scene Description

7.2.1 Introduction

7.2.1.1 Scope

MPEG-4 addresses the coding of objects of various types: Traditional video and audio frames, but also natural video and audio objects as well as textures, text, 2- and 3-dimensional graphic primitives, and synthetic music and sound effects. To reconstruct a multimedia scene at the terminal, it is hence no longer sufficient to encode the raw audiovisual data and transmit it, as MPEG-2 does, in order to convey a video and a synchronized audio channel. In MPEG-4, all objects are multiplexed together at the encoder and transported to the terminal. Once de-multiplexed, these objects are composed at the terminal to construct and present to the end user a meaningful multimedia scene, as illustrated in Figure 7-4. The placement of these elementary Media Objects in space and time is described in what is called the Scene Description layer. The action of putting these objects together in the same representation space is called the Composition of Media Objects. The action of transforming these Media Objects from a common representation space to a specific rendering device (speakers and a viewing window for instance) is called Rendering.

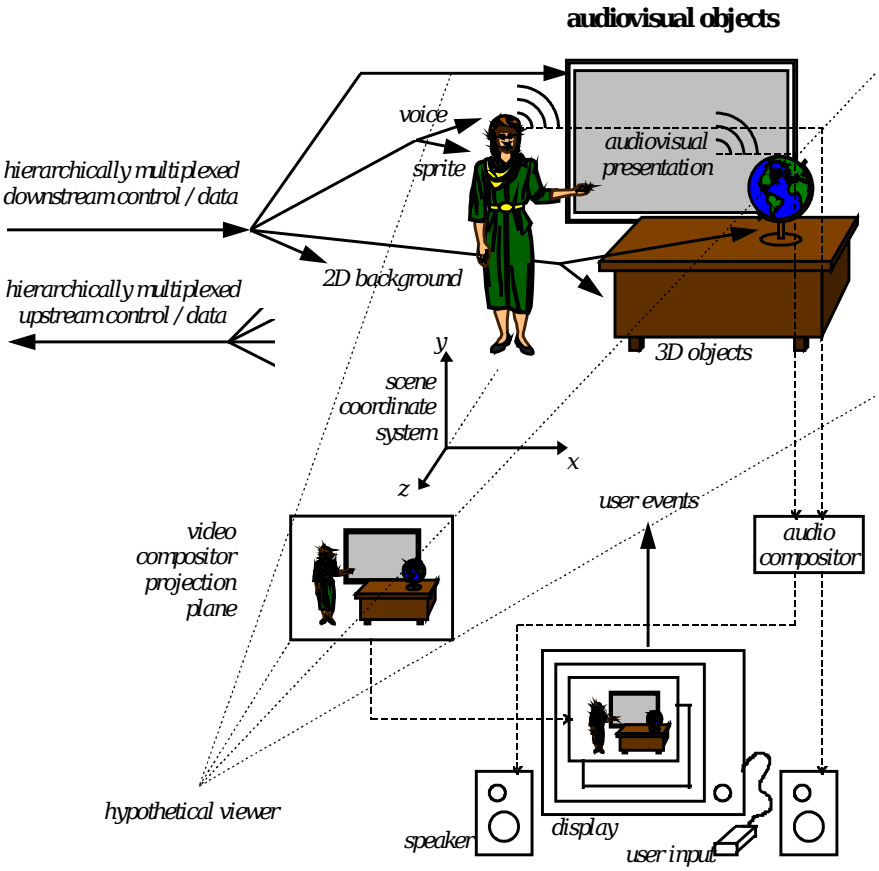


Figure 7-4: An example of an MPEG-4 multimedia scene

The independent coding of different objects may achieve a higher compression rate, but also brings the ability to manipulate content at the terminal. The behaviours of objects and their response to user inputs can thus also be represented in the Scene Description layer, allowing richer multimedia content to be delivered as an MPEG-4 stream.

λ 7.2.1.2 Composition

The intention here is not to describe a standardized way for the MPEG-4 terminal to compose or render the scene at the terminal. Only the syntax that describes the spatio-temporal relationships of Scene Objects is standardized.

λ 7.2.1.3 Scene Description

In addition to providing support for coding individual objects, MPEG-4 also provides facilities to compose a set of such objects into a scene. The necessary composition information forms the scene description, which is coded and transmitted together with the Media Objects which comprise the scene.
In order to facilitate the development of authoring, manipulation and interaction tools, scene descriptions are coded independently from streams related to primitive Media Objects. Special care is devoted to the identification of the parameters belonging to the scene description. This is done by differentiating parameters that are used to improve the coding efficiency of an object (e.g. motion vectors in video coding algorithm), from those used as modifiers of an object's characteristics within the scene (e.g. position of the object in the global scene). In keeping with MPEG-4's objective to allow the modification of this latter set of parameters without having to decode the primitive Media Objects themselves, these parameters form part of the scene description and are not part of the primitive Media Objects. The following sections detail characteristics that can be described with the MPEG-4 scene description.

˄7.2.1.3.1 *Grouping of objects*

An MPEG-4 scene follows a hierarchical structure which can be represented as a Directed Acyclic Graph. Each node of the graph is a scene object, as illustrated in Figure 7-5. The graph structure is not necessarily static; the relationships can change in time and nodes may be added or deleted.

Figure 7-5: Logical structure of the scene

˄7.2.1.3.2 *Spatio-Temporal positioning of objects*

Scene Objects have both a spatial and a temporal extent. Objects may be located in 2-dimensional or 3-dimensional space. Each Scene Object has a local co-ordinate system. A *local co-ordinate system* for an object is a co-ordinate system in which the object has a fixed spatio-temporal location and scale (size and orientation). The local co-ordinate system serves as a handle for manipulating the Scene Object in space and time. Scene Objects are

positioned in a scene by specifying a co-ordinate transformation from the object's local co-ordinate system into a global co-ordinate system defined by its parent Scene Object in the tree. As shown on Figure 7-5, these relationships are hierarchical, therefore the objects are placed in space and time according to their parent.

7.2.1.3.3 Attribute value selection

Individual Scene Objects expose a set of parameters to the composition layer through which part of their behaviour can be controlled by the scene description. Examples include the pitch of a sound, the colour of a synthetic visual object, or the speed at which a video is to be played. A clear distinction should be made between the Scene Object itself , the attributes that enable the placement of such an object in a scene, and any Media Stream that contains coded information representing some attributes of the object (a Scene Object that has an associated Media Stream is called a Media Object). For instance, a video object may be connected to an MPEG-4 encoded video stream, and have a start time and end time as attributes attached to it.

MPEG-4 also allows for user interaction with the presented content. This interaction can be separated into two major categories: client-side interaction and server-side interaction. In this section, we are only concerned by the client side interactivity that can be described within the scene description.

Client-side interaction involves content manipulation which is handled locally at the end-user's terminal, and can be interpreted as the modification of attributes of Scene Objects according to specified user inputs. For instance, a user can click on a scene to start an animation or a video. This kind of user interaction has to be described in the scene description in order to ensure the same behaviour on all MPEG-4 terminals.

l7.2.2 Concepts

l7.2.2.1 Global structure of a BIFS Scene Description

A BIFS scene description is a compact binary format representing a pre-defined set of Scene Objects and behaviours along with their spatio-temporal relationships. The BIFS format contains four kinds of information:

1. The attributes of Scene Objects, which define their audio-visual properties
2. The structure of the scene graph which contains these Scene Objects
3. The pre-defined spatio-temporal changes (or "self-behaviours") of these objects, independent of the user input. For instance, "this red sphere rotates forever at a speed of 5 radians per second, around this axis".
4. The spatio-temporal changes triggered by user interaction. For instance, "start the animation when the user clicks on this object".

These properties are intrinsic to the BIFS format. Further properties relate to the fact that the BIFS scene description data is itself conveyed to the receiver as an Elementary Stream. Portions of BIFS data that become valid at a given point in time are delivered within time-stamped Access Units as defined in Subclause . This streaming nature of BIFS allows modification of the scene description at given points in time by means of BIFS-Update or BIFS-Anim as specified in Subclause . The semantics of a BIFS stream are specified in Subclause .

λ 7.2.2.2 BIFS Scene graph

Conceptually, BIFS scenes represent, as in the ISO/IEC DIS 14772-1:1997, a set of visual and aural primitives distributed in a Direct 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 supports scenes composed of aural primitives as well as:

- l 2D only primitives
- l 3D only primitives
- l A mix of 2D and 3D primitives, in several ways:

- 2D and 3D complete 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 coordinate system in a 3D scene

The following figure describes a typical BIFS scene structure.

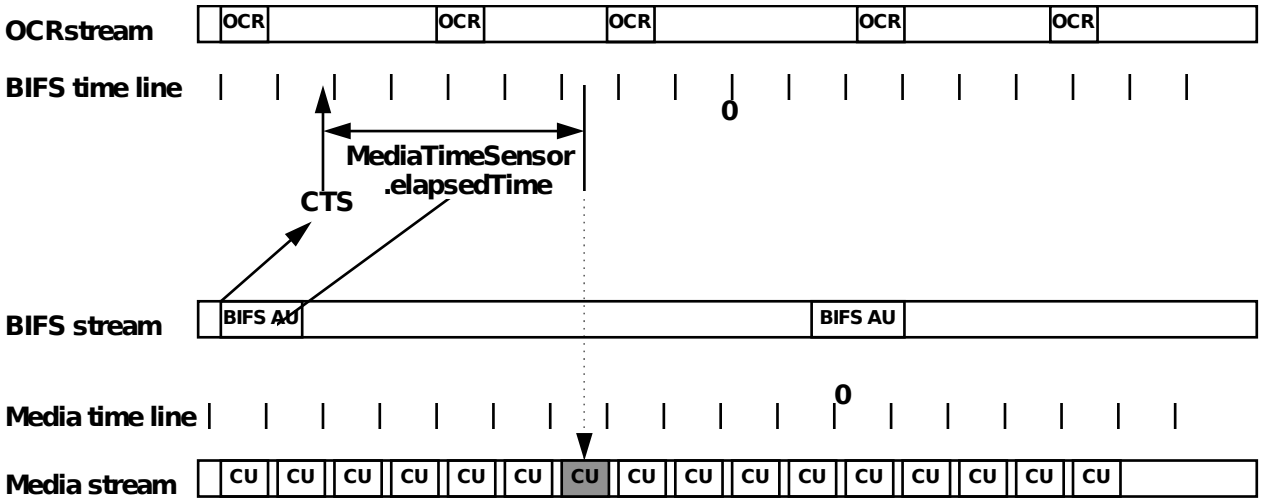


Figure 7-6: A complete scene graph example. We see the hierarchy of 3 different scene graphs: the 2D graphics scene graph, 3D graphics scene graph, and the layers 3D scene graphs. As shown in the picture, the 3D layer-2 view the same scene as 3D-layer1, but the viewpoint may be different. The 3D object-3 is a Appearance node that uses the 2D-Scene 1 as a texture node.

λ 7.2.2.3 2D Coordinate System

For the 2D coordinate system, the origin is positioned at lower left-hand corner of the viewing area, X positive to the right, Y positive upwards. 1.0 corresponds to the width and the height of the rendering area. The rendering area is either the whole screen, when viewing a single 2D scene, or the rectangular area defined by the parent grouping node, or a Composite2DTexture, CompositeMap or Layer2D that embeds a complete 2D scene description.

[1]

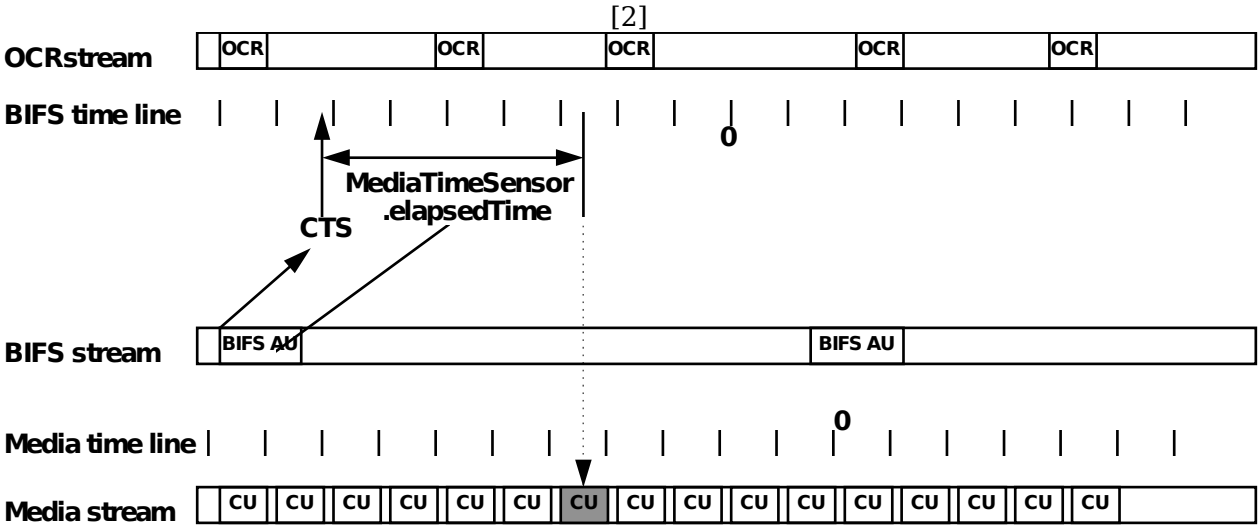


Figure 7-7: 2D Coordinate System

λ 7.2.2.4 3D Coordinate System

The 3D coordinate system is as described in ISO/IEC DIS 14772-1:1997, Section 4.4.5. The following figure illustrates the coordinate system.

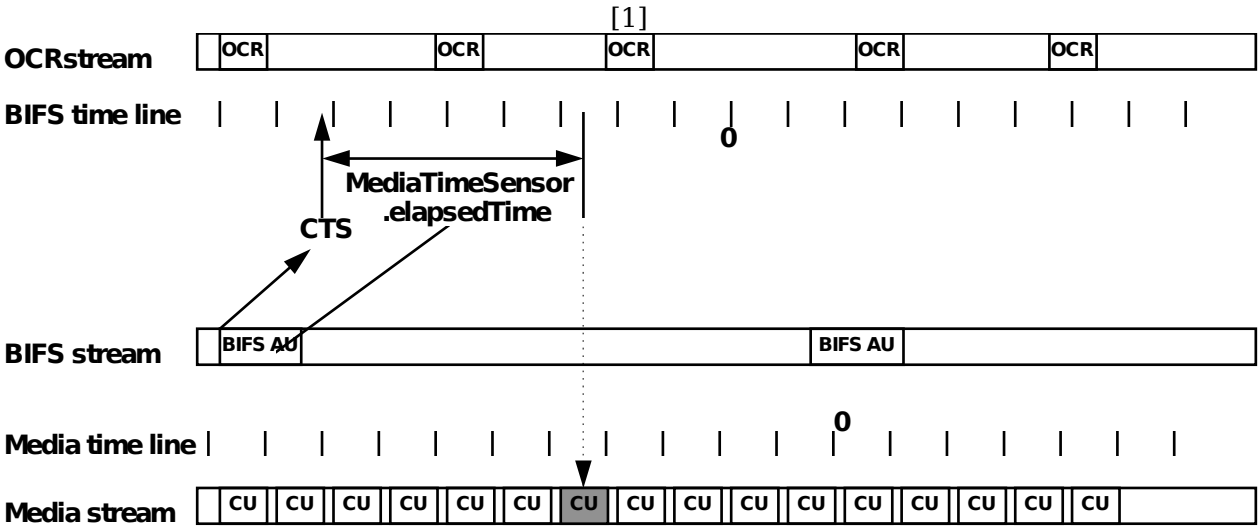


Figure 7-8: 3D Coordinate System

λ 7.2.2.5 Standard Units

As described in ISO/IEC DIS 14772-1:1997, Section 4.4.5, the standard units used in the scene description are the following:

Category	Unit
Distance in 2D	Rendering area width and height
Distance in 3D	Meter
Colour space	RGB [0,1], [0,1] [0,1]

Time	seconds
Angle	radians

Figure 7-9: Standard Units

λ 7.2.2.6 Mapping of scenes to screens

BIFS scenes enable the use of still images and videos by copying, pixel by pixel the output of the decoders to the screen. In this case, the same scene will appear different on screens with different resoultions.

BIFS scenes that do not use these primitives are independent from the screen on which they are viewed.

λ 7.2.2.7 Nodes and fields

λ *7.2.2.7.1 Nodes*

The BIFS scene description consists of a collection of *nodes* which describe the scene and its layout. An 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 and the exact type of the node is specified using a **nodeType** field.

An object may be completely described within the BIFS information, e.g. **Box** with **Appearance**, or may also require streaming data from one or more AV decoders, e.g. **MovieTexture** or **AudioSource**. In the latter case, the node points to an **ObjectDescriptor** which indicates which Elementary Stream(s) is (are) associated with the node, or directly to a URL description (see ISO/IEC DIS 14772-1, Section 4.5.2). **ObjectDescriptors** are denoted in the URL field with the scheme "mpeg4od:<number>", <number> being the **ObjectDescriptorID**.

* *7.2.2.7.2 Fields and Events*

See ISO/IEC DIS 14772-1:1997, Section 5.1.

λ 7.2.2.8 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, single valued begin with SF.

* *7.2.2.8.1 Numerical data and string data types*

For each basic data types, single fields and multiple fields data types are defined in ISO/IEC DIS 14772-1:1997, Section 5.2. Some further restrictions are described herein.

l7.2.2.8.1.1 SFBool

l7.2.2.8.1.2 SFColor/MFColor

l7.2.2.8.1.3 SFFloat/MFFloat

l7.2.2.8.1.4 SFInt32/MFInt32

When **ROUTE**ing values between two SF**Int32**s 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

l7.2.2.8.1.5 SFRotation/MFRotation

l7.2.2.8.1.6 SFString/MFString

l7.2.2.8.1.7 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.

l7.2.2.8.1.8 SFVec2f/MFVec2f

l7.2.2.8.1.9 SFVec3f/MFVec3f

l7.2.2.8.2 Node data types

Nodes in the scene are also represented by a data type, namely SFNode and MFNode types. MPEG-4 has also defined a set of sub-types, such as SFColorNode, SFMaterialNode. These Node Data Types are used for better compression of BIFS scenes to take into account the context to achieve better compression, but are not used at runtime. SFNode and MFNode types are sufficient for internal representations of BIFS scenes.

λ 7.2.2.9 Attaching nodeIDs to nodes

Each node in a BIFS scene graph may have a **nodeID** associated with it, for referencing. ISO/IEC DIS 14772-1:1997, Section 4.6.2 describes the **DEF** semantic which is used to attachnames to nodes. In BIFS scenes, an integer represented as 10 bits is used for **nodeIDs**, allowing for a maximum of 1024 nodes to be simultaneously referenced.

λ 7.2.2.10 Using pre-defined nodes

In the scene graph, nodes may be accessed for future changes of their fields. There are two main sources for changes of the BIFS nodes' fields:

1. The modifications occurring from the ROUTE mechanism, which enables the description of behaviours in the scene
2. The modifications occurring from the BIFS update mechanism (see).

The mechanism for naming and reusing nodes is given in ISO/IEC DIS 14772-1:1997, Section 4.6.3. The following restrictions apply:

1. Nodes are identified by the use of nodeIDs, which are binary numbers conveyed in the BIFS bitstream.
2. The scope of nodeIDs is given in Subclause
3. No two nodes delivered in a single Elementary Stream may have the same nodeID.

λ 7.2.2.11 Scene Structure and Semantics

The BIFS Scene Structure is as described in ISO/IEC DIS 14772-1:1997. However, MPEG-4 includes new nodes that extend the capabilities of the scene graph.

*7.2.2.11.1 2D Grouping Nodes

The 2D grouping nodes enable the ordered drawing of 2D primitives. The 2D Grouping Nodes are:

- Group2D
- Transform2D
- Layout
- Form

*7.2.2.11.22D Geometry Nodes

The 2D Geometry Nodes represent 2D graphic primitives. They are:

- Circle
- Rectangle
- IndexedFaceSet2D
- IndexedLineSet2D

*7.2.2.11.32D Material Nodes

2D Material Nodes have color and transparency fields, and have additional 2D nodes as fields to describe the graphic properties. The following nodes fall into this category:

- Material2D
- LineProperties2D
- ShadowProperties2D

*7.2.2.11.4Face and Body nodes

To offer a complete support for Face and Body animation, BIFS has a set of nodes that defines the Face and Body parameters.

- FBA
- Face
- Body
- FDP
- FBADefTables
- FBADefTransform
- FBADefMesh
- FIT
- FaceSceneGraph

*7.2.2.11.5Mixed 2D/3D Nodes

These nodes that enable the mixing of 2D and 3D primitives.

- Layer2D
- Layer3D
- Composite2Dtexture
- Composite3DTexture
- CompositeMap

l 7.2.2.12 Internal, ASCII and Binary Representation of Scenes

MPEG-4 describes the attributes of Scene Objects using Node structures and fields. These fields can be one of several types (see). To facilitate animation of the content and modification of the objects' attributes in time, within the MPEG-4 terminal, it is necessary to use an internal representation of nodes and fields as described in the node specifications (Subclause). 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 decoders shall not be affected by the way in which they internally represent and transform data; i.e., they shall behave as if their internal representation is as defined herein.

However, at transmission time, different attributes need to be quantized or compressed appropriately. Thus, the binary representation of fields may differ according to the precision needed to represent a given Media Object, or according to the types of fields. The semantic of nodes is described in Subclause , and the binary syntax which represents the binary format as transported in MPEG-4 streams is provided in the Node Coding Tables, in Subclause .

˙7.2.2.12.1 *Binary Syntax Overview*

The Binary syntax represents a complete BIFS scene.

l7.2.2.12.1.1 Scene Description

The whole scene is represented by a binary representation of the scene structure. The binary encoding of the scene structure restricts the VRML Grammar as defined in ISO/IEC DIS 14772-1:1997, Annex A, but still enables representation of any scene observing this grammar to be represented. For instance, all **ROUTEs** are represented at the end of the scene, and a global grouping node is inserted at the top level of the scene.

l7.2.2.12.1.2 Node Description

Node types are encoded according to the context of the node.

l7.2.2.12.1.3 Fields description

Fields are quantized whenever possible. The degradation of the scene can be controlled by adjusting the parameters of the **QuantizationParameter** node.

l7.2.2.12.1.4 ROUTE description

All **ROUTEs** are represented at the end of the scene.

λ 7.2.2.13 BIFS Elementary Streams

The BIFS Scene Description may, in general, be time variant. Consequently, BIFS data is itself of a streaming nature, i.e. it forms an elementary stream, just as any media stream associated with the scene.

˙7.2.2.13.1 *BIFS-Update commands*

BIFS data is encapsulated in BIFS-Update commands. For the detailed specification of all BIFS-Update commands see Subclause . Note that this does not imply that a BIFS-Update command must contain a complete scene description.

˙7.2.2.13.2 *BIFS Access Units*

BIFS data is further composed of BIFS Access Units. An Access Unit groups one or more BIFS-update commands that shall become valid (in an ideal compositor) at a specific point in

time. Access Units in BIFS elementary streams therefore must be labeled and time stamped by suitable means.

*7.2.2.13.3 Requirements on BIFS elementary stream transport

Framing of Access Units for random access into the BIFS stream as well as time stamping must be provided. In the context of the tools specified by this Working Draft of International Standard this is achieved by means of the related flags and the Composition Time Stamp, respectively, in the AL_PDU Header.

*7.2.2.13.4 Time base for the scene description

As for every media stream, the BIFS elementary stream has an associated time base as specified in Subclause . The syntax to convey time bases to the receiver is specified in Subclause . It is possible to indicate on set up of the BIFS stream from which other Elementary Stream it inherits its time base. All time stamps in the BIFS are expressed in **SFTime** format but refer to this time base.

*7.2.2.13.5 Composition Time Stamp semantics for BIFS Access Units

The AL-packetized Stream that carries the Scene Description shall contain Composition Time Stamps (CTS) only. The CTS of a BIFS Access Unit indicates the point in time that the BIFS description in this Access Unit becomes valid (in an ideal compositor). This means that any audiovisual objects that are described in the BIFS Access Unit will ideally become visible or audible exactly at this time unless a different behavior is specified by the fields of their nodes.

*7.2.2.13.6 Multiple BIFS streams

Scene description data may be conveyed in more than one BIFS elementary stream. This is indicated by the presence of one or more Inline/Inline2D nodes in a BIFS scene description that refer to further elementary streams as specified in Subclause /. Therefore multiple BIFS streams have a hierarchical dependency. Note, however, that it is not required that all BIFS streams adhere to the same time base. An example for such an application is a multi-user virtual conferencing scene.

The scope for names (nodeID, objectDescriptorID) used in a BIFS stream is given by the grouping of BIFS streams within one Object Descriptor (see Subclause). Conversely, BIFS streams that are not declared in the same Object Descriptor form separate name spaces. As a consequence, an Inline node always opens a new name space that is populated with data from one or more BIFS streams. It is forbidden to reference parts of the scene outside the name scope of the BIFS stream.

*7.2.2.13.7 Time Fields in BIFS nodes

In addition to the Composition Time Stamps that specify the validity of BIFS Access Units, several time dependent BIFS nodes have fields of type **SFTime** that identify a point in time at which an event happens (change of a parameter value, start of a media stream, etc). These fields are time stamps relative to the time base that applies to the BIFS elementary stream that has conveyed the respective nodes. More specifically this means that any time duration is therefore unambiguously specified.

SFTime fields of some nodes require absolute time values. Absolute time (wall clock time) can not be directly derived through knowledge of the time base, since time base ticks need not have a defined relation to the wall clock. However, the absolute time can be related to the time base if the wall clock time that corresponds to the composition time stamp of the BIFS Access Unit that has conveyed the respective BIFS node is known. This is achieved by an optional wallClockTimeStamp as specified in Subclause . After reception of one such time association, all absolute time references within this BIFS stream can be resolved.

Note specifically that **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.

l7.2.2.13.7.1 Example

The example below shows a BIFS Access Unit that is to become valid at CTS. It conveys a media node that has an associated media stream. Additionally it includes a MediaTimeSensor that indicates an elapsedTime that is relative to the CTS of the BIFS AU. Third a ROUTE node routes Time=(now) to the startTime of the Media Node when the elapsedTime of the MediaTimeSensor has passed. The Composition Unit (CU) that is available at that time CTS+MediaTimeSensor.elapsedTime is the first CU available for composition.

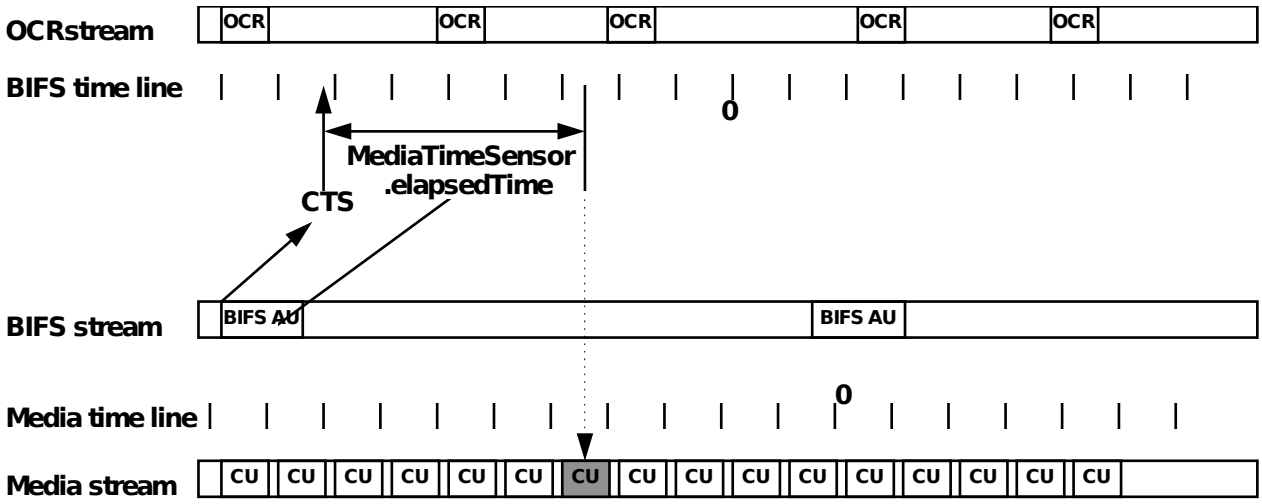


Figure 7-10: Media start times and CTS

*7.2.2.13.8 Time events based on media time

Regular SFTime time values in the scene description allow to trigger events based on the BIFS time base. In order to be able to trigger events in the scene at a specific point on the media time line, a MediaTimeSensor node is specified in Subclause .

λ 7.2.2.14 Sound

Sound nodes are used for building audio scenes in the MPEG-4 decoder terminal from audio sources coded with MPEG-4 tools. The audio scene description is meant to serve two requirements:

- "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 transforms as they are needed artistically.

Sound may be included in either the 2D or 3D scene graphs. In a 3D scene, the sound may be spatially presented to apparently originate from a particular 3D direction, according to the positions of the object and the listener.

The **Sound** node is used to attach sound to 3D and 2D scene graphs. As with visual objects, the audio objects represented by this node has a position in space and time, and are

transformed by the spatial and grouping transforms of nodes hierarchically above them in the scene.

The nodes *below* the **Sound** nodes, however, constitute an *audio subtree.* This subtree is used to describe a particular audio object through the mixing and processing of several audio streams. Rather than representing a hierarchy of spatiotemporal tranformations, the nodes within the audio subtree represent a signal-flow graph that describes how to create the audio object from the sounds coded in the **AudioSource** streams. That is, each audio subtree node (**AudioSource**, **AudioMix**, **AudioSwitch**, **AudioFX)** accepts one or several channels of input sound, and describes how to turn these channels of input sound into one or more channels of output sound. The only sounds presented in the audiovisual scene are those sounds which are the output of audio nodes that are children of a **Sound** node (that is, the "highest" outputs in the audio subtree).

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

7.2.2.14.1 Overview of sound node semantics

This section describes the concepts for normative calculation of the sound objects in the scene in detail, and describes the normative procedure for calculating the sound which is the output of a Sound object given the sounds which are its input.

Recall that the audio nodes present in an audio subtree do not each represent a sound to be presented in the scene. Rather, the audio subtree represents a signal-flow graph which computes a single (possibly multichannel) 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 subtrees, as connected to **Sound** node. This section describes the proper computation of this signal-flow graph and resulting audio object.

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

Each node in the audio subtree has an associated *input buffer* and *output buffer* of sound, except for the **AudioSource** node, which has no input buffer. The CB for the audio source acts as the input buffer of sound for the **AudioSource** with which the decoder is associated. As with CBs, 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 a Sound 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.

l7.2.2.14.1.1 Sample-rate conversion

If the various children of a Sound 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 childrens' 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 at least as high as that of quadratic interpolation, that is, the noise power level due to the interpolation shall be no more than –12dB relative to the power of the signal. Implementors are encouraged to build the most sophisticated resampling capability possible into MPEG-4 terminals.

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 a high computational complexity.

λ 7.2.2.14.1.1.1 Example

Suppose that node **N** has children **M1** and **M2**, all three Sound 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**. Then, 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**.

l7.2.2.14.1.2 Number of output channels

If the **numChan** field of an audio object, 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 subtree calculation, as follows:

1. 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.
2. If the value of the **numChan** field is strictly greater than the number of channels produced, then the "extra" channels shall be set to all 0's in the output buffer.

7.2.2.14.2 Audio-specific BIFS

This section summarizes where issues related specifically to audio, or that have special implications for audio, can be found in this document.

l7.2.2.14.2.1 Audio-related BIFS nodes

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

Node	Purpose	Subclause
AudioClip	Insert an audio clip to scene	
AudioDelay	Insert delay to sound	
AudioMix	Mix sounds	
AudioSource	Define audio source input to scene	
AudioFX	Attach structured audio objects to sound	
AudioSwitch	Switching of audio sources in scene	
Group, Group2D	Grouping of nodes and subtrees in a scene	
ListeningPoint	Define listening point in a scene	
Sound	Define properties of sound	

λ 7.2.2.15 Drawing Order

2D scenes are considered to have zero depth. Nonetheless, it is important to be able to specify the order in which 2D objects are composited. For this purpose, the **Transform2D** node contains a **field** which is called **drawingOrder**.

The **drawingOrder** field provides a mechanism for explicitly specifying the order in which 2D objects are drawn. **drawingOrder** values are floating point numbers and may be negative. By default, **drawingOrder** for all 2D objects is 0 and the following rules include methods for resolving conflicts between multiple objects having the same **drawingOrder**.

1. Objects are drawn in order.
2. The object having the lowest **drawingOrder** is drawn first (taking into account negative values).
3. Objects having the same **drawingOrder** are drawn in the order in which they appear in the scene description.

˄7.2.2.15.1 *Scope of Drawing Order*

The scope of drawing orders, explicit and implicit, is limited to any sub-scene to which they may belong. Note that sub-scenes, as a whole, have a drawing order within the higher level scene or sub-scene to which they belong.

λ 7.2.2.16 Bounding Boxes

Some 2D nodes have bounding box fields. The bounding box gives rendering hints to the implementation and is not necessary. However, when a bounding box is specified, it shall enclose the shape.
The bounding box dimensions are specified by two fields. The bboxCenter specifies the point, in the node's local coordinate, about which the box is centred. The bboxSize fields specify the bounding box's size. A default bboxSize value, (-1, -1), implies that the bounding box is not specified and, if needed, is calculated by the browser.

λ 7.2.2.17 Sources of modification to the scene

λ *7.2.2.17.1 Interactivity and behaviors*

To describe interactivity and behavior of Scene Objects, the event passing mechanism defined in ISO/IEC DIS 14772-1:1997, Section 4.10, is used
Sensors and ROUTEs describe interactivity and behaviors. Sensor nodes sense changes based on user interaction or a change in the scene. Sensor nodes then generate events that 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.

```
ROUTE <name>.<field/eventName> TO <name>.<field/eventName>
```

˄7.2.2.17.2 *External modification of the scene: BIFS Update*

The BIFS-Update mechanism enables to change any property of the scene graph. For instance, Transform nodes can be modified to move objects in space, Material nodes can be changed to modify the objects aspect, fields of geometric nodes can be totally or partially changed to modify the geometry of objects on the fly. Finally, nodes and behaviors can be added or removed.

l7.2.2.17.2.1 Overview

BIFS-Update commands are used to modify a set of properties of the scene at a given time instant in time. However, for continuous changes of the parameters of the scene, the animation scheme as following section needs to be used. Update Commands are grouped into Update Frames to sned several commands in an access unit. The following four basic commands are possible:

1. Insertion.
2. Deletion.
3. Replacement.
4. Replacement of the whole scene

5. Scene Repeat

All first three commands can take three kinds of parameters:

1. A node.
2. A field or an indexed value in a multiple field.
3. A **ROUTE**.

In addition to these three, the Replacement command can also take a specific field as parameter.

The replacement of the whole scene requires a node tree representing a valid BIFS scene. The SceneReplace command is the only random access point in the BIFS stream.

The Repeat Scene command enables to repeat all the updates from the last Replace Scene.

In order to modify the scene the sender must transmit a BIFS-Update frame which contains one or more update commands. The identification of a node in the scene is provided by a **nodeID**. It should be noted that it is the sender's responsibility to provide this **nodeID**, which must be unique (see). A single source of update is assumed. The identification of node field is provided by sending its fieldIndex, its position in the fields list of that node.

The time of application of the update is specified using the time stamp mechanism provided by the Flex Mux.

Figure 7-11: BIFS-Update Commands

l7.2.2.17.2.2 Update examples

The following are examples of the use of BIFS-Update commands:

- BIFS- Update may be used in order to enable progressive loading of a large scene, by adding progressively more and more objects.
- In a Broadcast environment, BIFS-Update may be used to Add a logo, a sub title, or any object on top of the existing Image of a Digital TV.

- In a 3D interactive shared world, BIFS-Update may be used to add an avatar in an existing 3D world.
- In a shared content creation application, BIFS-Update may be used to send to all participants the modifications of sizes, or properties of the scene being created.

- ***7.2.2.17.3External animation of the scene: BIFS-Anim***

The BIFS-Update commands provide a mechanism for describing changes in the BIFS. However, this format is not well-suited for continuous changes of parameters of the scene. A header format called " 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 colour attributes. BIFS-Anim is used for streaming animations.

l7.2.2.17.3.1 Overview

The principle of the BIFS-Anim consists in transmitting the following parameters in order:

1. Configuration parameters of the animation also called the "Animation Mask".
2. These describe the transmitted fields and may specify their quantization and compression parameters. The parameters may be **eventIn** or **exposedField** fields of updatable nodes in the scene, i.e. nodes having a nodeID, that have been assigned a **dynID**. Such fields are called "dynamic" fields. The Animation Mask is composed of several elementary masks defining these parameters.
3. Animation parameters are sent as a set of "Animation Frames".

An Animation Frame contains all the new values of the animation parameters at a specified time, except 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 (the *difference* between the current and previous values is sent) modes.

Animation parameters can be applied to any field of any node of a scene which has been previously declared as a dynamic field by assigning a **dynID**. The types of dynamic fields are all scalar types, single or multiple:

1. SFInt32/MFInt32
2. SFFloat/MFFloat
3. SFRotation/MFRotation
4. SFColor/MFColor

l7.2.2.17.3.2 Animation Mask

Animation Masks represent the fields that will be transmitted in the animation frames for a given animated node, and their associated quantization parameters. In an Animation Mask, the dynamic fields are specified, along with any quantization parameters. For each node of a scene that is to be animated, an Elementary Mask is transmitted that specifies these parameters.

l7.2.2.17.3.3 Animation Frames

Animation Frames specify the values of the dynamic fields of nodes that are being animated in streams. The specific coding is defined in the binary syntax for elementary types. The precise syntax for other types of fields is specified in the binary format description.

l7.2.2.17.3.4 Animation Examples

Typical use of BIFS-Anim include:

- In a broadcast digital TV application, the Animation Stream may be used to animate an existing logo or creating a special effect for a scene transition.
- In a personal communication service, an application using Facial Animation may be completed by animating also the viewpoint, the position and lighting of the faces in the scene.
- In a shared game, the animation stream may be used to animate the position of a user in the scene.

- In a multimedia consultation application, an animation stream may be used to drive the animation of an object in the scene.

7.2.3 BIFS Binary Syntax

7.2.3.1 BIFS Scene and Nodes Syntax

The scene syntax describes the syntax for an entire BIFS scene description.

7.2.3.1.1 BIFSScene

```
class BIFSScene {
      BIFSNodes   nodes ;          // Description of the nodes
      bit(1)      hasROUTEs
      if (hasROUTEs)
            ROUTEs      routes ;    // Description of the ROUTEs
}
```

7.2.3.1.2 BIFSNodes

```
class BIFSNodes{
      bit(1)      hasWorldInfo ;
      if (hasWorldInfo)
            SFNode(SFWorldInfoNode) node ;
                        // Only one WorldInfo node per BIFS description
      SFNode(SFTopNode) node ;
// This a Group2D, Group, Layer2D or Layer3D node
}
```

7.2.3.1.3 SFNode

```
class SFNode(NDT) {
      bit(1) isReused ;       // This bit describes whether this node is
                               // a reused nodes or a newly defined one
                               // This is equivalent to the USE statement
      if (isReused)
      {
            bit(10) nodeID; // The NodeID to be re-used
      }
      else
      {

                  bit (1) predefined;     // Is the node a predefined node?
            if (predefined)
            {
                  NodeType(NDT) NType ;// This represents the type of the node.
                                       // of the node should be inferred from
                                       //the fields type
            }
            bit(1) isUpdatable
            if (isUpdatable)
            {
                  bit(10) nodeID;
            }
            bit(1) MaskAccess ;
            if (MaskAccess) {
                  MaskNodeDescription node ;
            }
```

```
            else {
                  ListNodeDescription node ;
            }
      }
}
```

7.2.3.1.4 MaskNodeDescription

```
Class MaskNodeDescription {
      for (i=0 ; i< NumberOfFields ; i++)
      {
            bit(1) Mask;
            if Mask
                  Field(fieldType) value;// get field value using
                                         // nodeType and fieldReference
                                         // to find out the appropriate
                                         // field type. This field can
                                         // itself be a new node
      }
}
```

7.2.3.1.5 ListNodeDescription

```
Class ListNodeDescrition
{
      bit(1) endFlag;
      while (!EndFlag){
            defID fieldReference;  // this identifies the field
                                         // for this nodeType. Length
                                         // is derived from the number
                                         // of fields in the node of
                                         // type defField and is the
                                         // number of  bits necessary
                                         // to represent the field
                                         // number
            Field(fieldType) value;      // get field value using
                                         // nodeType and
                                         // fieldReference to find out
                                         // the appropriate field type.
                                         // This field can itself be a
                                         // new node
            bit(1) endFlag;
            }
      }
}
```

7.2.3.1.6 NodeType

```
class NodeType(NDT) {
      bit(vlcNbBits) type;            // Accoridng to Node Quantization Tables
}
```

7.2.3.1.7 Field

```
class Field(fieldType) {
      switch fieldType {
      case SFFieldType :
            SFField(fieldType) value;
            break ;
      case MFFieldType :
            MFField(fieldType) value;
            break ;
```

```
	}
}
```

7.2.3.1.8 MFField

```
class MFField(fieldType)  {
	bit (1) isListDescription
	if (isListDescription) {
		MFListDescription(fieldType)  field;
	}
	else
		MFVectorDescription(fieldType)  field;
}

class MFListDescription(fieldType)  {
	do {
		SField(fieldType)	field ;
		bit(1)	moreFields;
	while (moreFields)
}

class MFVectorDescription {
	int (5)	NbBits ;		// Number of bits for the number
					 // of fields
	int(NbBits) numberOfFields;	// Number of Fields
	for (i=0 ; i< ; i++) {
		SFField(fieldType) field ;
	}
}
```

7.2.3.1.9 SFField

```
class SFField(fieldType) {
	switch fieldType {
		case SFNodeType :
			SFNode(fieldType) value;
			break ;
		case SFBoolType :
			SFBool value;
			break ;
		case SFColorType :
			SFColor value;
			break ;
		case SFFloatType :
			SFFloatvalue;
			break ;
		case SFImageType :
			SFImage value;
			break ;
		case SFInt32Type :
			SFInt32 value;
			break ;
		case SFRotationType :
			SFRotation value;
			break ;
		case SFStringType :
			SFString value;
			break ;
		case SFTimeType :
			SFTime value;
```

```
                  break ;
            case SFUrlType :
                  SFUrl value;
                  break ;
            case SFVec2fType :
                  SFVec2f value;
                  break ;
            case SFVec3fType :
                  SFVec3f value;
                  break ;
      }
}
```

l7.2.3.1.9.1 SFBool

```
class SFBool {
bit(1) value;
}
```

l7.2.3.1.9.2 SFColor

```
class SFColor {
      if (QUANTIZED) {
            QuantizedField value;
      }
      else {
            float rValue;
            float gValue;
            float bValue;
      }
}
```

l7.2.3.1.9.3 SFFloat

```
class SFColor {
      if (QUANTIZED) {
            QuantizedField value;
      }
      else {
            float value;
      }
}
```

l7.2.3.1.9.4 SFImage

```
class SFImage {
      uint(10)    width;
      uint(10)    height;
      bit(2)      numComponents;
      uchar(8)    pixels[numComponents*width*height];
}
```

l7.2.3.1.9.5 SFInt32

```
class SFInt32 {
      if (QUANTIZED) {
            QuantizedField value;
      }
      else {
```

```
            int(32)     value;
      }
}
```

l7.2.3.1.9.6 SFRotation

```
class SFRotation {
      if (QUANTIZED) {
            QuantizedField value;
      }
      else {
            float xAxis;
            float yAxis;
            float zAxis;
            float angle;
      }
}
```

l7.2.3.1.9.7 SFString

```
class SFString {
      if (QUANTIZED) {
            QuantizedField value;
      }
      else {
            UTF8String        value;
      }
}
```

l7.2.3.1.9.8 SFTime

```
class SFInt32 {
      if (QUANTIZED) {
            QuantizedField value;
      }
      else {
            float       value;
      }
}
```

l7.2.3.1.9.9 SFUrl

```
class SFUrl {
      bit(1) isOD;
      if (isOD) {
            bit(10)     ODid;
      }
      else {
            SFString    urlValue;
      }
}
```

l7.2.3.1.9.10 SFVec2f

```
class SFVec2f {
      if (QUANTIZED) {
            QuantizedField value;
      }
```

```
      else {
            float value1;
            float value2;
      }
}
```

l7.2.3.1.9.11 SFVec3f

```
class SFVec3f {
      if (QUANTIZED) {
            QuantizedField value;
      }
      else {
            float value1;
            float value2;
            float value3;
      }
}
```

7.2.3.1.10 QuantizedField

```
class QuantizedField {
      switch QUANTType {
            case 1 :    // 3D Positions
                  int(position3DNbBits) x;
                  int(position3DNbBits) y;
                  int(position3DNbBits) z;
                  break ;
            case 2 :    // 2D Positions
                  int(position2DNbBits) x;
                  int(position2DNbBits) y;
                  break ;
            case 3 :    // Drawing Order
                  int(drawOrderNbBits) d;
                  break ;
            case 4 :    // Color
                  int(colorNbBits) r;
                  int(colorNbBits) g;
                  int(colorNbBits) b;
                  break ;
            case 5 :    //TextureCoordinate
                  int(textureCoordinateNbBits) xPos;
                  int(textureCoordinateNbBits) yPos;
                  break ;
            case 6 :    // Angle
                  int(angleNbBits) angle;
                  break ;
            case 7 :    // Scale
                  int(scaleNbBits) xScale;
                  int(scaleNbBits) yScale;
                  if (3D)
                        int(scaleNbBits) zScale;
                  break ;
            case 8 :    //Interpolator Key
                  int(keyNbBits) key;
                  break ;
            case 9 :    // Normal
                  int(3)            octantNb;
                  int(2)            triantNb;
```

```
                  int(normalNbBits) xPos;
                  int(normalNbBits) yPos;
                  break ;
            case 10 :   // Rotation
                  int(3)            octantNb;
                  int(2)            triantNb;
                  int(normalNbBits) xPos;
                  int(normalNbBits) yPos;
                  int(angleNbBits) angle;
                  break ;
            case 11 :   // Object Size 3D
                  int(position3DNbBits) size;
                  break ;
            case 12 :   // Object Size 2D
                  int(position2DNbBits) size;
                  break ;
            case 13 :   // Linear Quantization
                  int(nbBits) value;
                  break ;
      }

}
```

7.2.3.1.11 Field IDs syntax

7.2.3.1.11.1 defID

```
class defID {
      int(vlcNbBits)    id;
}
```

l7.2.3.1.11.2 inID

```
class inID {
      int(vlcNbBits)    id;
}
```

l7.2.3.1.11.3 outID

```
class outID {
      int(vlcNbBits)    id;
}
```

l7.2.3.1.11.4 dynID

```
class dynID {
      int(vlcNbBits)    id;
}
```

7.2.3.1.12 ROUTE syntax

7.2.3.1.12.1 ROUTEs

```
class ROUTEs {
      bit(1)      ListDescription ;
      if (ListROUTEsDesciprion)
            ListROUTEs  nodes ;
      else
            VectorROUTEs      nodes ;
}
```

l7.2.3.1.12.2 ListROUTEs

```
class ListROUTEs{
      do {
            ROUTE       route ;
            bit(1)            moreROUTEs ;
      }
      while (moreROUTEs)

}
```

l7.2.3.1.12.3 VectorROUTEs

```
class VectorROUTEs {
      int(10)     numberOfROUTEs ;  // We allow for a maximum of 1024
                                    //ROUTEs
      for (i=0 ; i< numberOfROUTEs < i++) {
            ROUTE       route ;
      }
}
```

λ 7.2.3.1.12.3.1 ROUTE

```
class ROUTE {
      bit(1) isUpdatable
      if (isUpdatable)
      {
            bit(10) ROUTEID;
      }
      bit(10)     outNodeID ;
      outID outFieldReference;      // this identifies the field
                                    // for the nodeType of.
                                    // outNodeID Length is derived
                                    // from the number of fields
                                    // in the node of this type
                                    // nodetype and is the
                                    // number of  bits necessary
                                    // to represent the field
                                    // number
      bit(10)     inNodeID ;
      inID  inFieldReference;       // Same as above
}
```

λ 7.2.3.2 BIFS-Update Syntax

λ *7.2.3.2.1 Update Frame*

```
class UpdateFrame {
      do {
            UpdateCommand command;  // this is the code of a complete
                              // update command
            bit (1) continue;
      } while (continue)
}
```

7.2.3.2.2 Update Command

```
class UpdateCommand {
bit(3) Command ;        // this is the code of the basic Command
switch Command {
case 0 :
```

```
InsertionCommand insert;
break ;
case 1 :
DeletionCommand delete ;
break ;
case 2 :
ReplacementCommand replace ;
break ;
case 3 :
SceneReplaceCommand sceneReplace ;
break ;
      case 4:
            break:                  // RepeatCommand, empty
}
```

7.2.3.2.3 Insertion Command

```
class InsertionCommand {
      bit(2) parameterType ; // this is the code of the basic Command
      switch parameterType {
            case ‘00’ :
                  NodeInsertion nodeInsert;
                  break ;
            case ‘10’ :
                  IndexedValueInsertion idxInsert;
                  break ;
            case ‘11’ :
                  ROUTEInsertion ROUTEInsert;
                  break ;
      }
}
```

l7.2.3.2.3.1 Node Insertion

```
class NodeInsertion {
      bit(10) nodeID ;        // This is the ID of the grouping node to which
                              // the node is added
      bit(2) insertionPosition;    // The position in the children field
      switch (insertionPosition) {
      case ‘00’ :             // insertion at a specified position
            bit (8) position; // the position in the children field
            SFNode(NDT) node; // the node to be inserted
            break;
      case ‘10’:              // insertion at the beginning of the field
            SFNode(NDT) node; // the node to be inserted
            break;
      case ‘11’:              // insertion at the end of the field
            SFNode(NDT) node; // the node to be inserted
            break;
      }
}
```

l7.2.3.2.3.2 IndexedValue Insertion

```
class IndexedValueInsertion {
      bit(10) nodeID ;        // This is the ID of the node to be modified
      inID id;                // the field to be changed. x is inferred from
                              // the number of exposed and event in fields
                              // of the node, according to the specification
      bit(2) insertionPosition;    // The position of the value in the field
```

```
      switch (insertionPosition) {
      case ‘00’ :             // insertion at a specified position
            bit (16) position;      // the absolute position in the field
            SFField(fieldType) value;     // the value to be inserted
            break;
      case ‘10’:              // insertion at the beginning of the field
            SFField(fieldType) value;     // the value to be inserted
            break;
      case ‘11’:              // insertion at the end of the field
            SFField(fieldType) value;     // the value to be inserted
            break;
      }
}
```

l7.2.3.2.3.3 ROUTE Insertion

```
class ROUTEInsertion {
      bit(10) departureNodeID ;           // the ID of the departure node
                                    // it still needs to be clarified
                                    // how ROUTEs are designated
      outID departureID;            // the index of the departure field
      bit(10) arrivalNodeID ;       // the ID of the arrival node
      inID arrivalID;         // the index of the arrival field
}
```

7.2.3.2.4 Deletion Command

```
class DeletionCommand {
bit(2) parameterType ;  // this is the code of the basic Command
switch parameterType {
case ‘00’ :
            NodeDeletion nodeDelete;
break ;
case ‘10’ :
            IndexedValueDeletion idxDelete;
break ;
case ‘11’ :
            ROUTEDeletion ROUTEDelete;
break ;
}
}
```

l7.2.3.2.4.1 Node Deletion

```
class NodeDeletion {
      bit(10) nodeID ;        // the ID of the node to be deleted
}
```

l7.2.3.2.4.2 IndexedValue Deletion

```
class IndexedValueDeletion {
      bit(10) nodeID ;        // This is the ID of the node to be modified
      inID id;                // the field to be deleted.
      bit(2) deletionPosition;      // The position in the children field
      switch (deletionPosition) {
      case ‘00’ :             // deletion at a specified position
            bit (16) position;      // the absolute position in the field
            SFField(fieldType) value;     // the new value
            break;
      case ‘10’:              // deletion at the beginning of the field
            SFField(fieldType) value;     // the new value
```

```
            break;
      case ‘11’:              // deletion at the end of the field
            SFField(fieldType) value;     // the new value
            break;
      }
}
```

l7.2.3.2.4.3 ROUTE Deletion

```
class ROUTEDeletion {
      bit(10) ROUTEID ;        // the ID of the ROUTE to be deleted
}
```

7.2.3.2.5 Replacement Command

```
class ReplacementCommand {
bit(2) parameterType ;  // this is the code of the basic Command
switch parameterType {
case 0 :
NodeReplacement nodeReplace;
break ;
case 1:
      FieldReplacement fieldReplace;
      break;
case 2 :
IndexedValueReplacement idxReplace;
break ;
case 3 :
ROUTEReplacement ROUTEReplace;
break ;
}
}
```

l7.2.3.2.5.1 Node Replacement

```
class NodeReplacement {
      bit(10) nodeID ;             // the ID of the node to be replaced
      SFNode(SFWorldNode) node;          // the new node
}
```

l7.2.3.2.5.2 Field Replacement

```
class FieldReplacement {
      bit(10) nodeID ;       // This is the ID of the node to be modified
      inID id;               // the index of the field to be replaced
      Field(fieldType) value;// the new field value, either single or list
}
```

l7.2.3.2.5.3 IndexedValue Replacement

```
class IndexedValueReplacement {
      bit(10) nodeID ;       // This is the ID of the node to be modified
      inField id;       // the index of the field to be replaced
      bit(2) replacementPosition;  // The position in the children field
      switch (replacementPosition) {
      case ‘00’ :             // replacement at a specified position
            bit (16) position;      // the absolute position in the field
            SFField(fieldType) value;          // the new value
            break;
      case ‘10’:              // replacement at the beginning of the field
            SFField(fieldType) value;     // the new value
            break;
```

```
      case ‘11’:              // replacement at the end of the field
            SFField(fieldType) value;     // the new value
            break;
      }
}
```

l7.2.3.2.5.4 ROUTE Replacement

```
class ROUTEReplacement {
      bit(10) ROUTEID ;             // the ID of the ROUTE to be replaced
      bit(10) departureNodeID ;           // the ID of the departure node
      outID departureID;            // the index of the departure field
      bit(10) arrivalNodeID ;       // the ID of the arrival node
      inID arrivalID;         // the index of the arrival field
}
```

l7.2.3.2.5.5 Scene Replacement

```
class SceneReplacementCommand {
      BIFSScene scene;        // the current scene graph is completely
                              // replaced by this new node which should
                        // contains a whole new scene graph
}
```

λ 7.2.3.3 BIFS-Anim Syntax

λ *7.2.3.3.1 BIFS AnimationMask*

λ 7.2.3.3.1.1 AnimationMask

```
Class AnimationMask
{
      do {
      ElementaryMask eMask;
            bit(1) Continue ;
      } while (Continue) ;
}
```

l7.2.3.3.1.2 Elementary mask

```
Class ElementaryMask {
      int (10) nodeID ;       // The nodeID of the node to be animated
      switch (NodeType) {
            case FDPtype:     // No initial mask for face
                  break;
            case BDPtype:     // No initial mask for body
                  break;
            default:
                  InitialFieldsMask mask ;
                                    // Describes which of the dynamic
                                    // fields are animated and their
                                    // parameters
      }
}
```

l7.2.3.3.1.3 InitialFieldsMask

```
class InitialFieldsMask {
      bit(1) animatedFields[numFields]; // read binary mask of dynamic
                                    // fields for this kind of nodes.
```

```
                                    //numField is the number of Dynamic fields
                                    // of the animated node.
                                    // 0 if field is animated, 1 otherwise
      for(int i=0; i < numFields; i++) {
            if(animatedFields[i]) { // is this field animated ?
                  if(field[i] instanceof (Mfield)) {
                        // Do we have a multiple field ?
                        bit(1) isTotal ; // if 1, all the elements of
                                         // the Multiple fields are
                                         // animated
                        if( ! isTotal) {  // animate specified indices
                              do {
                                    uint(32) index;
                                    bit(1) Continue ;
                              } while (Continue) ;
                  }
                  InitialAnimQP QP[i];
                                    // read initial quantization
                                    // parameters QP[i] for field i
            }
```

l7.2.3.3.1.4 }InitialAnimQP

```
InitialAnimQP {
      Switch (category)
            case 0 :                // Position 3D
                  float(32)   Ix3Min;
                  float(32)   Iy3Min;
                  float(32)   Iz3Min;
                  float(32)   Ix3Max;
                  float(32)   Iy3Max ;
                  float(32)   Iz3Max;
                  float(32)   Px3Min;
                  float(32)   Py3Min;
                  float(32)   Pz3Min;
                  float(32)   Px3Max;
                  float(32)   Py3Max ;
                  float(32)   Pz3Max;
                  int (5)     position3DNbBits ;
                  break ;
            case 1 :                // Position 2D
                  float(32)   Ix2Min ;
                  float(32)   Iy2Min ;
                  float(32)   Ix2Max ;
                  float(32)   Iy2Max ;
                  float(32)   Px2Min ;
                  float(32)   Py2Min ;
                  float(32)   Px2Max ;
                  float(32)   Py2Max ;
                  int (5)     position2DNbBits ;
                  break ;
            case 2 :                // SFColor
                  float(32)   IcMin ;
                  float(32)   IcMax ;
                  float(32)   PcMin ;
                  float(32)   PcMax ;
                  int (5)     colorNbBits ;
                  break ;
            case 3:                       // Angles
                  float(32)   IangleMin ;
```

```
                float(32)   PangleMax ;
                float(32)   IangleMin ;
                float(32)   PangleMax ;
                int (5)     angleNbBits ;
                break ;
          case 4:                       // Normals
                int(5)      normalNbBits ;
                break ;
          case 5:                       // Scale
                float(32)   IscaleMin ;
                float(32)   IscaleMax ;
                float(32)   PscaleMin ;
                float(32)   PscaleMax ;
                int(5)      scaleNbBits;
                break ;
          case 6:                       // Rotation
                bit(1)      hasAxis;
                if (hasAxis) {
                      int (5)     axisNbBits ;
                }
                bit(1)      hasAngle;
                if (hasAngle) {
                      float(32)   IangleMin ;
                      float(32)   IangleMax ;
                      float(32)   PangleMin ;
                      float(32)   PangleMax ;
                      int (5)     angleNbBits ;
                }
                break ;
          case 7:                       // Size of objects
                float(32)   IsizeMin ;
                float(32)   PsizeMax ;
                int (5)     sizeNbBits ;
                break ;
}
```

7.2.3.3.2 Animation Frame Syntax

7.2.3.3.2.1 AnimationFrame

```
class AnimationFrame
{
      AnimationFrameHeader    header;
      AnimationFrameData      data;
}
```

l7.2.3.3.2.2 AnimationFrameHeader

```
class AnimationFrameHeader {
      if(next_bit() == '00000000000000000000000')
      {
            uint(32) AnimationStartCode ; // synchronization code for Intra
      }
      bit(1) isIntra;
      bit(1) nodeMask[numNodes];   // Mask for setting which of the
                                                  // nodes are animated
      if (isIntra) {
            bit(1) isFrameRate;
            if (isFrameRate) {
```

```
                  FrameRate rate;   // decode_frame_rate() function of FBA
            }
            bit(1) isTimeCode;
            if (isTimeCode) {
                  uint(18) timeCode;
            }
      }
      bit(1) hasSkipFrames;
      if (hasSkipFrames)
            SkipFrames  skip;       // decode_skip_frames() in FBA
}
```

l7.2.3.3.2.3 AnimationFrameData

```
Class AnimationFrameData (nodeMask){
for(int i=0; i<numNodes; i++) {    // for each animated nodes
            if(nodeMask[i]) {      // do we have values for node i ?
            switch (nodeType) {
            {                 case FaceorBody :
                        FBAFrameData data;     // As defined in visual CD
                        Break;
                  case IndexedFaceSet2D:
                        Mesh2DframeData data;  //As defined in visual CD
                        Break,
                  default:    // All other types of nodes
                        for(int j=0; j < numFields[i] ; j++) {
                                          // for each animated
                                          // dynamic field
                                    AnimationField  AField;
                                          // The syntax of the animated
                                          // field. This depends on the
                                          // field and node type
                                    }
                        }
                  }
            }
      }
}
```

l7.2.3.3.2.4 AnimationField

```
class AnimationField
{
      if(animatedFields[j]) {// is this field animated ?
                              if (isIntra) {
                                    bit(1) hasQP ;
                                          // Do we send new
                                          //quantizationparameters ?
                                   if(hasQP)
                                          {     // read new QP for field
                                                // of the current node
                                                // QuantizationParameter
                                          AnimQP QP;
                                         }
                                    AnimIValue value;
                                                // read intra-coded
                                                //value of field
                              }
```

```
                        else
                              AnimPValue value;
                                    // read predicted-coded
                                    //value of field
                  }
            }
}
```

l7.2.3.3.2.5 AnimQP

```
AnimQP {
      Switch (category)
            case 0:
                  bit (1) IMinMax ;
                  if (IMinMax)
                  {
                        float(32)   Ix3Min;
                        float(32)   Iy3Min;
                        float(32)   Iz3Min;
                        float(32)   Ix3Max;
                        float(32)   Iy3Max ;
                        float(32)   Iz3Max;
                  }
                  bit (1) PMinMax ;
                  if (PMinMax)
                  {
                        float(32)   Px3Min;
                        float(32)   Py3Min;
                        float(32)   Pz3Min;
                        float(32)   Px3Max;
                        float(32)   Py3Max ;
                        float(32)   Pz3Max;
                  }
                  bit (1) hasNbBits ;
                  if (hasNbBits) {
                        int (5)     position3DNbBits ;
                  }
                  break ;
            case 1 :          // Position 2D
                  bit (1) IMinMax ;
                  if (IMinMax)
                  {
                        float(32)   Ix2Min ;
                        float(32)   Iy2Min ;
                        float(32)   Ix2Max ;
                        float(32)   Iy2Max ;
                  }
                  bit (1) PMinMax ;
                  if (PMinMax)
                  {
                        float(32)   Px2Min ;
                        float(32)   Py2Min ;
                        float(32)   Px2Max ;
                        float(32)   Py2Max ;
                  }
                  bit (1) hasNbBits ;
                  if (hasNbBits) {
                        int (5)     position2DNbBits ;
                  }
                  break ;
```

```
case 2 :                // SFColor
      bit (1) IMinMax ;
      if (IMinMax)
      {
            float(32)   IcMin ;
            float(32)   IcMax ;
      }
      if (PMinMax)
       if (PMinMax)
      {
            float(32)   PcMin ;
            float(32)   PcMax ;
      }
      bit (1) hasNbBits ;
      if (hasNbBits) {
            int (5)     colorNbBits ;
      }
       break ;
case 3:                       // Angles
      bit (1) IMinMax ;
      if (IMinMax)
      {
            float(32)   IangleMin ;
            float(32)   IangleMax ;
      }
      bit (1) PMinMax ;
      if (PMinMax)
      {
            float(32)   PangleMin ;
            float(32)   PangleMax ;
      }
       bit (1) hasNbBits ;
      if (hasNbBits) {
            int (5)     NbBits ;
      }
       break ;
case 4:                       // Normals
      int(5)      normalNbBits ;
      break ;
case 5:                       // SFRotation 3D
      bit (1) IMinMax ;
      if (IMinMax)
      {
            float(32)   IscaleMin ;
            float(32)   IscaleMax ;
      }
       bit (1) PMinMax ;
      if (PMinMax)
      {
            float(32)   PscaleMin ;
            float(32)   PscaleMax ;
      }
      bit (1) hasNbBits ;
      if (hasNbBits) {
            int (5)     scaleNbBits ;
      }
       break ;
case 6:                       // Rotation
      bit(1) hasAngle;
```

```
                  bit(1) hasNormal;
                  bit (1) IMinMax ;
                  if (IMinMax)
                  {
                        float(32)   IangleMin ;
                        float(32)   IangleMax ;
                  }
                  bit (1) PMinMax ;
                  if (PMinMax)
                  {
                        float(32)   PangleMin ;
                        float(32)   PangleMax ;
                  }
                  bit (1) hasNbBits ;
                  if (hasNbBits) {
                        if (hasAngle)
                              int(5)      angleNbBits ;
                        if (hasNormal)
                              int(5)      normalNbBits ;
                  }
                  break ;
            case 7:                       // Size of objects
                  bit (1) IMinMax ;
                  if (IMinMax)
                  {
                        float(32)   IsizeMin ;
                        float(32)   IsizeMax ;
                  }
                  bit (1) PMinMax ;
                  if (PMinMax)
                  {
                        float(32)   PsizeMin ;
                        float(32)   PsizeMax ;

                  }
                  bit (1) hasNbBits ;
                  if (hasNbBits) {
                        int (5)     sizeNbBits ;
                  }
                  break ;
}
```

l7.2.3.3.2.6 AnimationIValue

```
class AnimationIValue {
      switch DYNType {
            case 0 :    // 3D Positions
                  int(vlcNbBits) x;
                  int(vlcNbBits) y;
                  int(vlcNbBits) z;
                  break ;
            case 1 :    // 2D Positions
                  int(vlcNbBits) x;
                  int(vlcNbBits) y;
                  break ;
            case 2 :    // Color
                  int(vlcNbBits) r;
                  int(vlcNbBits) g;
                  int(vlcNbBits) b;
                  break ;
```

```
            case 3 :    // Angle
                  int(vlcNbBits) angle;
                  break ;
            case 5 :    // Scale
                  int(vlcNbBits) xScale;
                  int(vlcNbBits) yScale;
                  if (3D)
                        int(vlcNbBits) zScale;
                  break ;
            case 6 :    // Rotation
                  if (hasAxis) {
                        int(3)           octantNb;
                        int(2)           triantNb;
                        int(vlcNbBits)   xPos;
                        int(vlcNbBits)   yPos;
                  }
                  if (hasAngle) {
                        int(vlcNbBits) angle;
                  }
                  break ;
            case 7 :    // Object Size 3D
                  int(vlcNbBits) size;
                  break ;
      }
}
```

l7.2.3.3.2.7 AnimationPValue

```
class AnimationPsValue {
      switch DYNType {
            case 0 :    // 3D Positions
                  int(vlcNbBits) x;
                  int(vlcNbBits) y;
                  int(vlcNbBits) z;
                  break ;
            case 1 :    // 2D Positions
                  int(vlcNbBits) x;
                  int(vlcNbBits) y;
                  break ;
            case 2 :    // Color
                  int(vlcNbBits) r;
                  int(vlcNbBits) g;
                  int(vlcNbBits) b;
                  break ;
            case 3 :    // Angle
                  int(vlcNbBits) angle;
                  break ;
            case 5 :    // Scale
                  int(vlcNbBits) xScale;
                  int(vlcNbBits) yScale;
                  if (3D)
                        int(vlcNbBits) zScale;
                  break ;
            case 6 :    // Rotation
                  if (hasAxis) {
                        int(3)           octantNb;
                        int(2)           triantNb;
                        int(vlcNbBits)   xPos;
                        int(vlcNbBits)   yPos;
```

```
                }
                if (hasAngle) {
                      int(vlcNbBits) angle;
                }
                break ;
          case 7 :    // Object Size 3D
                int(vlcNbBits) size;
                break ;
      }
}
```

l7.2.4 BIFS Decoding Process and Semantic

l7.2.4.1 BIFS Scene and Nodes Decoding Process

The scene syntax describes the syntax for an entire BIFS scene description. However, this syntax is always used in combination with a BIFS-Update command, as described in the next section.

7.2.4.1.1 BIFS Scene

The BIFS Scene represents the global scene. A BIFS Scene is always associated to a BIFS-Update ReplaceScene command. The BIFS Scene is tructured in the following way:

- The Nodes
- The ROUTEs

- ***7.2.4.1.2 BIFS Nodes***

The BIFS Nodes represent all the nodes contain in the scene. The BIFS nodes are organized in the following way:

- First, a **WorldInfo** node is optionally sent.
- Second, one of the four Top nodes is sent. The top node can be either a **Group2D** , **Layer2D**, **Group** or **Layer3D** node.

- ***7.2.4.1.3 SFNode***

The **SFNode** represents a general node representation. The encoding depends on the Node Data Type of the node (NDT).
A reused node means that a **USE** statement is present, and hence the node is ony a pointer to an already **DEF'ed** node signaled by the **nodeID**.
If **isReused** is **FALSE**, the node is entirely defined. The **predefined** flag enables to distinguish between nodes defined in the current BIFS spec, and nodes defined in future extensions.
Next the **NodeType** is inferred from the Node Data Type, as explained in the following section.
The **isUpdatable** flag enables to assign a **nodeID** to the node.
The node definition is then sent, either with a **MaskNodeDescription**, or with a **ListNodeDescription**.

7.2.4.1.4 MaskNodeDescription

In the **MaskNodeDescription**, a **Mask** indicates for each def field of the node (according to the node type), if the field is defined or not. Fields are sent in the order of the specification of the semantinc of the node. According to the order of the fields, the field type is known and used to decode the field.

7.2.4.1.5 ListNodeDescription

In the **ListNodeDescription**, fields are directly addressed by their **fieldReference**. The reference is sent as a **defID** and its parsing depends on the node type, as explained in the defID section.

7.2.4.1.6 NodeType

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 following explains how to determine the exact type of node from the nodeType:

1. The data type of the field parsed indicates the Node Data Type: **SF2DNode**, **SFColorNode**, and so on. The first 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 inferred
3. Finally, the **nodeType** gives the nature of the node to be parsed.

A typical example is the **Shape** node.

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
```

If a texture is applied on the geometry, a texture field will be transmitted. Currently, the MovieTexture, the PixelTexture and ImageTexture, Composite2Dtexture and Composite3DTexture are available. This means the nodeType for the texture node can be coded using 3 bits.

7.2.4.1.7 Field

A **field** is encoded according to its type: single or multiple. A multiple field is a collection of single fields.

7.2.4.1.8 MFField

Multiple fields can be encoded. **MFField** types can be encoded with a list-like or mask-like description. The *MaskFieldDescription* or *ListFieldDescription* may be used and the choice is normally made according to the number of elements in the multiple field.

7.2.4.1.9 SFField

Single fields are coded according to the type of the field. The fields have a default syntax that specifies a raw encoding when no quantization is applied. The quantization parameters are read from a special node called **QuantizationParameter**. For quantization, the following categories have been identified.

0	None
1	3D Position
2	2D positions

3	depth
4	SFColor
5	Texture Coordinate
6	Angle
7	Scale
8	Interpolator keys
9	Normals
10	Rotations (9+5)
11	Object Size 3D (1)
12	Object Size 2D (2)
13	Linear Scalar Quantization
14	Reserved

For each field that may be quantized, a quantization parameter is assigned, refering to the table of quantization types. Along with quantization parameters, min and max values are specified for each filed of each node.

The scope of the quantization is only a single BIFS access unit. The quantization schemes fall into four main categories:

1. No Quantization (0): If no **QuantizationParameter** nodes are sent, no quantization is applied.
2. Generic quantization scheme: In that case, a linear scalar quantization scheme is applied. The parameters that set this quantization scheme are conveyed in **QuantizationParameter** nodes. The use of a **QuantizationParameter** node enables to set the **QUANTIZATION** boolean (present in the **QuantizedField** syntax) to TRUE.
3. Linear Quantization Scheme (13): In that case, a fixed number of bits is used to quantize the field. The number of bits used is specified in the field declaration, as well as the minimum and maximum values.
4. Specific Quantization schemes can be applied (14) when requested. In that case, the syntax is documented directly in the node.

*7.2.4.1.10 QuantizedField

If the **QUANTIZATION** boolean is TRUE, a linear scalar quantization is applied on fields. For each category of fields, a min and max value is given, as well as the number of bits to represnt the value. For a value v to be quantified, define Nb the number of allowed bits, (V_{min},V_{max}) its minimal and maximal values. Then, the quantized value v_q is defined by:

For the inverse quantization:

The min value used for quantization is the largest of the values defined in the QuantizationParameters node and the specific field parameters as described in the node table.

The max value used for quantization is the smallest of the values defined in the QuantizationParameters node and the specific field parameters, as described in the node table.

If the value assigned is infinite, the meaning is the max or min value for the numerical type encoded (32 bit integer, ANSI C float,..).

Note that in the case of a vectorial field value 2 cases may arise:

1. The min and max values given Smin and Smax are scalar, in which case Vmin and Vmax given by the multiplication of the constant 1 vector (a vector with all components set to 1), with Smin and Smax. This is in particular true for the scale factors.
2. The min and max values are given in a vectorial form, in which these values are directly Vmin and Vmax. This is in particular the case of the 2D and 3D positions.

There are 3 special cases in the categories specified in the previous tabular:

- For normals, the quantization method is the following: A normal is a set of 3 floating values representing a vector in 3-d space with unit length. The quantization process first divides the unit sphere into eight octants. The signs of the 3 coordinates of the nomal determine the octant and the first 3 bits of the quantized normal. Then each octant is further symetrically divided into 3 ‘triants’ of equal size (a triant is a quadrilateral on the sphere). The index of the most significant coordinate (the one with the largest absolute value) determines the triant and the 2 next bits. Each triant is then mapped into a unit square. Finally each axis of the square is evenly subdivided into $2^{QNORMAL}$ so that position within a triant can be associated with a couple (a_q, b_q), where a_q and b_q have integer values between 0 and $2^{normalNbBits}$ -1 The mapping of the triant {x>0, y>0, z>0, x>z, x>y} into a unit square is , . The inverse mapping is , ,

.

-
  - The mapping is defined similarly for the other triants. bits wll be used to designate the octant used. 2 bits will be used to designate the triant. The parameter normalNbBits specify that we code the normal value on a square grid with $2^{normalNbBits}$ elements on each axis. Normals will be thus be coded with normalNbBits+5 in total.
- fields of type **SFRotation** are made of 4 floats: 3 for an axis of rotation and 1 for an angle. For this field, two quantizers are used: one for the axis of rotation which is a normal and one for the angle.
- For the values related to the sizes of the primitives, such as the **Sphere**, **Circle**, **Cone** nodes, the distance of the diagonal of the bounding box specified by the position min and max values is taken into account for the Vmax value. The minimal value is considered to be zero. Hence the Vmax value can be represented as the euclidian distance of diagonal of the surrounding bounding box, given by , where and are the vectorial 2D or 3D min and max positions.
- For values quantized with scheme 13, the number of bits used for quantization is specified in the node tables.
- For fields named url, a specific encoding is used. A flag indicates whether an object descriptor is used, or a url described as a SFString.

- For SFImage types, the width and height of the Image are sent. **numComponents** defines the image type. The 4 following types are enabled:
  1. If the value is ‘00’, then a grey scale image is defined.
  2. If the value is ‘01’, a grey scale with alpha channel is used.
  3. If the value is ‘10’, then an r, g, b image is used.
  4. If the value is ‘11’, then an r,g, b image with alpha channel is used.

7.2.4.1.11 Field and Events IDs Decoding Process

Four different fieldIDs have been identified to refer to fields in the nodes. All field Ids are encoded with a variable number of bit. For each field of each node, the binary values of the field Ids are defined in the node tables.

l7.2.4.1.11.1 DefID

The **defID**s correspond to the Ids for the fields defined with nodes declaration. They correspond to exposedField and field types.

l7.2.4.1.11.2 inID

The **inID**s correspond to the Ids for the events and fields that can be modified from outside the node. They correspond to exposedField and eventIn types.

l7.2.4.1.11.3 outID

The **outID**s correspond to the Ids for the events and fields that can be output from the node. They correspond to exposedField and eventOut types.

l7.2.4.1.11.4 dynID

The **dynID**s correspond to the Ids for fields that can be animated using the BIFS-Anim scheme. They correspond to a subset of the fields designated by **inID**s.

7.2.4.1.12 ROUTE Decoding Process

ROUTEs are encoded using list or vector descriptions, as multiple fields and nodes. ROUTEs, as nodes, can be assigned an ID. **inID** and **outID** are used for the ROUTE syntax.

λ 7.2.4.2 BIFS-Update Decoding Process

λ *7.2.4.2.1 Update Frame*

An **UpdateFrame** is a collection of BIFS update commands, and corresponds to one access unit. The **UpdateFame** is the only valid syntax for carrying BIFS scenes in an access unit.

7.2.4.2.2 Update Command

For each **UpdateCommand**, a 3 bit flag, **command**, signals one of the 5 basic commands.

7.2.4.2.3 Insertion Command

There are four basic insertion commands, signalled by a 2 bit flag.

l7.2.4.2.3.1 Node Insertion

A node can be inserted in the children field of a grouping node. The node can be inserted at the beginning, at the end, or at a specified position in the children list. This in particular useful for 2D nodes. The **NodeDataType (NDT)** of the inserted node is known from the NDT of the children field in which the node is inserted.

l7.2.4.2.3.2 IndexedValue Insertion

The field in which the value is inserted must a multiple value type of field. The field is signalled with an **inID**. The **inID** is parsed using the table for the **Node Type** of the node in which the value is inserted, which is inferred from the **nodeID**.

l7.2.4.2.3.3 ROUTE Insertion

A ROUTE is inserted in the list of ROUTEs simply by specifying a new ROUTE.

7.2.4.2.4 Deletion Command

There are three types of deletion commands, signalled by a 2 bit flag.

l7.2.4.2.4.1 Node Deletion

The node deletion is simply signalled by the **nodeID** of the node to be deleted. When deleting a node, all fields are also deleted, as well as all **ROUTE**s related to the node or its fields.

l7.2.4.2.4.2 IndexedValue Deletion

This command enables to delete an element of a multiple value field. As for the insertion, it is possible to delete at a specified position, at the beginning or at the end.

l7.2.4.2.4.3 ROUTE Deletion

Deleting a ROUTE is simply performed by giving the ROUTE ID. This is similar to the deleting of a node.

*7.2.4.2.5 Replacement Command

There are 3 replacement commands, signalled by a 2 bits flag.

l7.2.4.2.5.1 Node Replacement

When a node is replaced, all teh ROUTEs pointing to this node are 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 a request to the Node Data Type of the replaced node.

l7.2.4.2.5.2 Field Replacement

Replacing a field enables to replace a given field of an existing node. The node in which the field is replaced is signalled with the **nodeID**. The field is signalled with an **inID**, which is encoded according to the Node Type of the changed node. If the replaced field is a node, then the same consequences as for a node replacement are assumed.

l7.2.4.2.5.3 IndexedValue Replacement

The **IndexedValueReplacement** command enables to modify 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.

l7.2.4.2.5.4 ROUTE Replacement

Replacing a ROUTE deletes the replaced ROUTE and replaces it with the new specified ROUTE.

l7.2.4.2.5.5 Scene Replacement

Replacing a new scene simply consists in replacing entirely the scene with a new **BIFSScene** scene. When used inside an Inline command, the semantic means replacing the sub scene which is previously empty. This thus simply inserts a new sub scene as expected in an Inline node.
In a BIFS Stream, the SceneReplacement commands are the only random access points in the BIFS streams.

l7.2.4.2.5.6 Scene Repeat

The SceneRepeat command enables to repeat all updates since the last random access point in the BIFS stream.

λ 7.2.4.3 BIFS-Anim Decoding Process

The dynamic fields are quantized and coded by a predictive coding scheme as shown in Figure 7-12. For each parameter to be coded in the current frame, the decoded value of this parameter in the previous frame is used as the prediction. Then the prediction error, i.e., the difference between the current parameter and its prediction, is computed and coded by variable length coding. This predictive coding scheme prevents the coding error from accumulating.

Figure 7-12: Encoding dynamic fields

Each dynamic field, that is field that can be animated is assigned, through the **InitialAnimQP** or **AnimQP** some quantization parameters than enable to control the quality and precision of the reconstructed animation stream.

The decoding process performs the reverse operations, by applying first an adaptive arithmetic decoder, then the inverse quantization and adding the previous field, in predictive (P) mode, or taking the new value directly in Intra (I) mode.

7.2.4.3.1 BIFS AnimationMask

The **AnimationMask** sets up the parameters for an animation. In particular, it specifies the fields and the nodes to be animated in the scene and their parameters. The Mask is sent in the **ObjectDescriptor** pointing to the BIFS-Anim stream.

l7.2.4.3.1.1 AnimationMask

The **AnimationMask** of **ElementaryMask** for animated nodes and their associated parameters.

l7.2.4.3.1.2 Elementary mask

The **ElementaryMask** links an **InitialFieldsMask** with a node specified by its **nodeID**.The InitialFieldsMask is not used for FDP, BDP or IndexedFaceSet2D nodes.

l7.2.4.3.1.3 InitialFieldsMask

The InitialFieldsMask specifies which fields of a given node are animated. In the case of a multiple field, either all the fields or a selected list of fields are animated.

l7.2.4.3.1.4 InitialAnimQP

The initial quantization masks are defined according to the categories of fields addressed. In the nodes specification, it is specified for each field whether it is a dynamic field or no, and in the case which type of quantization and coding scheme is applied. The fields are grouped in the following category for animation:

0	3D Position
1	2D positions
2	SFColor
3	Angle
4	Normals
5	Scale
6	Rotations3D (3+4)
7	Object Size

For each type of quantization, the min and max values for I and P mode, as well as the number of bits to be used for each type is specified. For the rotation, it is possible to choose to animate the angle and/or the axis with the **hasAxis** and **hasAngle** bits. When the flags are set to TRUE, the validity of the flag is for the current parsed frame; and untill the next **AnimQP** that sets the flag to a different value.

7.2.4.3.2 Animation Frame Decoding Process

7.2.4.3.2.1 AnimationFrame

The **AnimationFrame** is the **Access Unit** for the BIFS-Anim stream. It contains the **AnimationFrameHeader**, which specifies some timing, and selects which nodes are being animated in the list of animated nodes, and the **AnimationFrameData**, which contains the data for all nodes being animated.

l7.2.4.3.2.2 AnimationFrameHeader

In the **AnimationFrameHeader**, a start code is sent optionally at each I or P frame. Additionally, a mask for nodes being animated is sent. The mask has the length of the number of nodes specified in the **AnimationMask**. A 1 in the header specifies that the node is animated for that frame, 0 that is is not animated in the current frame. In the header, if in **Intra** mode, some additional timing information are also specified. The timing information follows the syntax of the Facial Animation specification in the Visual MPEG-4 Specification. Finally, it is possible to skip a number of **AnimationFrame** by using the **FrameSkip** syntax specified in the afore mentioned document.

l7.2.4.3.2.3 AnimationFrameData

The **AnimationFrameData** corresponds to the field data for the nodes being animated. In the case of an IndexedFaceSet2D, a face, or a body, the syntax used is the one of the MPEG-4 Visual Specification. In other cases, for each field animated node and for each animated field the **AnimationField** is sent. **NumFields [i]** represents the number of animated fields for node i.

l7.2.4.3.2.4 AnimationField

In an **AnimationField**, if in **Intra** mode, a new **QuantizationParameter** value is optionally sent. Then comes the I or P frame.

All numerical parameters as defined in the categories below follow the same coding scheme. This scheme is identical to the FBA animation stream, except for the quantization parameters:

- In P (Predictive) mode: for each new value to send, we code its difference with the preceding value. Values are quantized with a uniform scalar scheme, and then coded with an adaptive arithmetic encoder, as described in ISO/IEC CD 14496-2.
- In I (Intra) mode: values of dynamic fields are directly quantized and coded with the same arithmetic adaptive coding scheme

The syntax for all the numerical field animation is the same for all types of fields. The category corresponds to the table below:

0	3D Position
1	2D positions
2	SFColor
3	Angle
4	Normals
5	Scale
6	Rotations3D (3+4)
7	Object Size or Scalar

l7.2.4.3.2.5 AnimQP

The **AnimQP** is identical to the **InitialAnimQP**, except that it enables to send min and max values as well as number of bits for quantization optionally, for each type of fields.

l7.2.4.3.2.6 AnimationIValue

Intra Values are coded as described in the Animation field section

l7.2.4.3.2.7 AnimationPValue

Predictive values are coded as described in the AnimationField section.

l7.2.5 Nodes Semantic

l7.2.5.1 Shared Nodes

l7.2.5.1.1 Shared Nodes Overview

The Shared nodes are those nodes which may be used in both 2D and 3D scenes.

7.2.5.1.2 Shared MPEG-4 Nodes

The following nodes are specific to MPEG-4.

l7.2.5.1.2.1 AnimationStream

l7.2.5.1.2.1.1 Semantic Table

AnimationStream {

exposedField	SFBool	loop	FALSE
exposedField	SFFloat	speed	1
exposedField	SFTime	startTime	0
exposedField	SFTime	stopTime	0
exposedField	MFString	url	[""]
eventOut	SFBool	isActive	FALSE

}

λ 7.2.5.1.2.1.2 Main Functionality

The **AnimationStream** node is a node aimed at controlling interactively an animation stream as defined in the BIFS-Animation format. The syntax and semantic is almost the same as the MovieTexture node which controls a video stream.

λ 7.2.5.1.2.1.3 Detailed Semantic

The **loop exposedField**, when TRUE, specifies that the video sequence shall play continuously. Having displayed the final available time VOP available, it shall begin the next loop by playing the first VOP. When **loop** is FALSE, playback shall occur once.

The **speed exposedField** controls playback speed. If a **AnimationStream** is inactive when the sequence is first loaded and the speed is non-negative, then frame 0 shall be used as the texture. If a **AnimationStream** is inactive when the sequence is first loaded and the speed is negative, then the last frame of the sequence shall be used as the texture. A **AnimationStream** shall display frame 0 if **speed** is 0. For positive values of **speed**, the frame an active **AnimationStream** will display at time now corresponds to the frame at movie time (i.e., in the movie's local time system with frame 0 at time 0, at speed = 1):

fmod (now - startTime, duration/speed)

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

duration + fmod(now - startTime, duration/speed).

When a **AnimationStream** becomes inactive, the frame corresponding to the time at which the **MovieTexture** became inactive shall persist as the texture. The **speed exposedField** indicates how fast the movie should 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. A negative **speed** specifies that the video sequence shall play backwards. However, content creators should note that this may not work for streaming movies or very large movie files.

The **startTime exposedField** specifies the moment at which the animation sequence shall begin to play.

The **stopTime exposedField** specifies the moment at which the animation sequence shall stop playing.

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

The **duration_changed eventOut** shall be sent when the length (in time) of the animation sequence has been determined. Otherwise, it shall be set to -1.

The **isActive eventOut** shall be sent as TRUE when the animation stream is playing. Otherwise, it shall be set to FALSE.

l7.2.5.1.2.2 AudioDelay

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.

λ 7.2.5.1.2.2.1 Semantic Table

AudioDelay {

exposedField	MFNode	**children**	NULL
exposedField	SFTime	**delay**	0
field	SFInt32	**numChan**	1
field	MFInt32	**phaseGroup**	NULL

}

λ 7.2.5.1.2.2.2 Main Functionality

This node is used to delay a group of sounds, so that they start and stop playing later than specified in the AudioSource nodes.

λ 7.2.5.1.2.2.3 Detailed Semantics

The **children** array specifies the nodes affected by the delay.

The **delay** field specifies the delay to apply to each chld.

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 .

l7.2.5.1.2.3 AudioMix

l7.2.5.1.2.3.1 Semantic Table

AudioMix {			
exposedField	MFNode	**children**	NULL
exposedField	SFInt32	**numInputs**	1
exposedField	MFFloat	**matrix**	NULL
field	SFInt32	**numChan**	1
field	MFInt32	**phaseGroup**	NULL
}			

λ 7.2.5.1.2.3.2 Main Functionality

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.

λ 7.2.5.1.2.3.3 Detailed Semantics

The **children** field specifies which nodes' outputs to mix together.

The **numInputs** field specifies the number of input channels. It should 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 , specifying a "2 into 3" mix, the value of

the **matrix** field should 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 .

λ 7.2.5.1.2.3.4 Calculation

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

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

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

where **children**[*i*][*j*] represents the *j*th output sample of the *i*th channel of the input buffer.

l7.2.5.1.2.4 AudioSource

l7.2.5.1.2.4.1 Semantic Table

AudioSource {

exposedField	MFString	**url**	NULL
exposedField	SFFloat	**pitch**	1
exposedField	SFTime	**startTime**	0
exposedField	SFTime	**stopTime**	0
field	SFInt32	**numChan**	1
field	MFInt32	**phaseGroup**	NULL

}

λ 7.2.5.1.2.4.2 Main Functionality

This node is used to add sound to an MPEG-4 scene. See the ISO/IEC CD 14496-3:1997 for information on the various audio tools available for coding sound.

λ 7.2.5.1.2.4.3 Detailed Semantics

The **objectDescriptorID** field specifies which decoded bitstream to include.
The **pitch** field controls the playback pitch for the Parametric and Structured Audio decoders. 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 controls the Parametric decoder directly; it is available as the globalPitch variable in the Structured Audio decoder. See the Structured Audio section of the Audio WD for more details.

The **startTime** field specifies a time at which to start the audio playing.

The **stopTime** field specifies a time at which to turn off the Sound. Sounds which have limited extent in time turn themselves off when finished. If the stopTime field is 0, the Sound continues until it is finished or plays forever.

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).

λ 7.2.5.1.2.4.4 Calculation

The audio output from the decoder according to the bitstream(s) referenced in the specified object descriptor is placed in the output buffer for this node.

For audio sources decoded using the Structured Audio decoder (CD 14496-3 Subpart 5) Profile 3 or Profile 4, several variables from the scene description must be mapped into standard names in the orchestra. See CD 14496-3, Subclause 5.11.

l7.2.5.1.2.5 AudioFX

l7.2.5.1.2.5.1 Semantic Table

AudioFX {			
exposedField	MFNode	**children**	NULL
exposedField	SFString	**orch**	""
exposedField	SFString	**score**	""
exposedField	MFFloat	**params**	NULL
field	SFInt32	**numChan**	1
field	MFInt32	**phaseGroup**	NULL
}			

λ 7.2.5.1.2.5.2 Main Functionality

The AudioFX node is used to allow arbitrary signal-processing functions to be included and applied to the child inputs. Any function which operates on an array of signals and returns another array of signals may be written in this language.

λ 7.2.5.1.2.5.3 Detailed Semantics

The **children** array contains the nodes operated upon by this effect 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, the MPEG-4 orchestra language. This code block should contain an orchestra header and some instrument definitions, and conform to the bitstream syntax of the **orchestra** class as defined in ISO/IEC CD 14496-3, Subpart 5, sections 5.1 and 5.4.

The **score** string may contain a tokenized score for the given orchestra written in SASL, the MPEG-4 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 CD 14496-3, Subclause 5.1.

The **params** field allows BIFS updates 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 CD 14496-3, Subclause 5.11.

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 .

λ 7.2.5.1.2.5.4 Calculation

The node is evaluated according to the semantics of the orchestra code contained in the **orch** field. See ISO/IEC CD 14496-3 Subpart 5 for the normative sections of this process, especially ISO/IEC CD 14496-3 Subclause 5.11. Within the orchestra code, the multiple channels of input sound are placed on the global bus input; first, all channels of the first child, then all the channels of the second child, and so on. The orchestra header should '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 CD 14496-3 Subclause 5.3.3.3.

l7.2.5.1.2.6 AudioSwitch

l7.2.5.1.2.6.1 Semantic Table

AudioSwitch {			
exposedField	MFNode	**children**	NULL
exposedField	MFInt32	**WhichChoice**	NULL
field	SFInt32	**NumChan**	1
field	MFInt32	**PhaseGroup**	NULL
}			

λ 7.2.5.1.2.6.2 Main Functionality

The **AudioSwitch** node is used to select one from a set of audio nodes. One input is passed through unchanged; the rest are ignored.

λ 7.2.5.1.2.6.3 Detailed Semantics

The **children** field specifies a list of child options.

The **which** field specifies which channels should be passed through. If **which[i]** is 1, then the ith child channel should 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 .

λ 7.2.5.1.2.6.4 Calculation

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 **which[0] + which[1] + ... + which[*j*] =** *i*.

l7.2.5.1.2.7 Conditional

l7.2.5.1.2.7.1 Semantic Table

Conditional {			
EventIn	SFBool	**activate**	FALSE
EventIn	SFBool	**reverseActivate**	FALSE
ExposedField	SFString	**buffer**	""
EventOut	SFBool	**isActive**	FALSE
}			

λ 7.2.5.1.2.7.2 Main Functionality

A **Conditional** node interprets a buffered bit stream when it is activated. This allows events to trigger node updates, deletions, and other modifications to the scene. The buffered bit stream is interpreted as if it had just been received. The typical use of this node for the implementation of the action of a button is the following: 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.

λ 7.2.5.1.2.7.3 Detailed Semantics

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 are interpreted as BIFS updates. These updates are not time-stamped; they are executed at the time of the event.

l7.2.5.1.2.8 MediaTimeSensor

l7.2.5.1.2.8.1 Semantic Table

MediaTimeSensor {

exposedField	SFNode	**Media**	NULL
field	SFNode	**Timer**	NULL

}

λ 7.2.5.1.2.8.2 Main Functionality

The MediaTimeSensor node provides a mechanism to attach a media stream to a Timer node, slaving the timer to the media stream’s timebase.

λ 7.2.5.1.2.8.3 Detailed Semantics

The MediaTimeSensor is a way to link a TimeSensor clock to a specific streaming media clock. The media field is a pointer to a node that is linked to a streaming media. All the SFTime values in the attached Timer node will then be interpreted as values related to the conveyed Time Base of the pointed stream. This enables in particular to start an animation after a given time that a media stream is streaming, whether it has been stopped or not. If the value of media is NULL, then the time events in the TimeSensor will be refering to the Time Base used by the BIFS stream.

l7.2.5.1.2.9 QuantizationParameter

l7.2.5.1.2.9.1 Semantic Table

QuantizationParameter {

field	SFBool	**isLocal**	FALSE
field	SFBool	**position3DQuant**	TRUE
field	SFVec3f	**position3DMin**	-?, -?, -?
field	SFVec3f	**position3DMax**	+?, +?, +?
field	SFInt32	**position3DNbBits**	16
field	SFBool	**position2DQuant**	TRUE
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
field	SFFloat	**colorMax**	1
field	SFInt32	**colorNbBits**	8
field	SFBool	**textureCoordinateQuant**	TRUE
field	SFFloat	**textureCoordinateMin**	0
field	SFFloat	**textureCoordinateMax**	1
field	SFInt32	**textureCoordinateNbBits**	16
field	SFBool	**angleQuant**	TRUE

field	SFFloat	**angleMin**	0
field	SFFloat	**angleMax**	2.?
field	SFInt32	**angleNbBits**	16
field	SFBool	**scaleQuant**	TRUE
field	SFFloat	**scaleMin**	0
field	SFFloat	**scaleMax**	+?
field	SFInt32	**scaleNbBits**	8
field	SFBool	**keyQuant**	TRUE
field	SFFloat	**keyMin**	0
field	SFFloat	**keyMax**	1
field	SFInt32	**keyNbBits**	8
field	SFBool	**normalQuant**	TRUE
field	SFInt32	**normalNbBits**	8

}

λ 7.2.5.1.2.9.2 Main Functionality

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 a local field to apply the quantization only to the node following the QuantizationParameter node. The significant advantage given by using a node structure for declaring the quantization parameters lies in the possibility to **DEF** and **USE** the QuantizationParameter Node.

λ 7.2.5.1.2.9.3 Detailed Semantics

λ 7.2.5.1.2.9.4 Example

The following example illustrates these possibilities :

```
DEF Q1 QuantizationParameter // defines a local
{                                  // quantization node
      isLocal           TRUE
}

DEF Q2 QuantizationParameter             // define a global
{                                           quantization node
position3Dmin           -5 -5 -5    // bounds for 3D positions
      position3Dmax             5  5  5
      positon3DNbBits   8                // bits to encode 3D positions
      colorNbBits       6                // bits to encode colors
}

Transform
{
      translation 10 10 10                     // coded by Q2.position3D
      children [
            Shape{
                  geometry  Cube {}
                  Appearance Appearance{
                        DEF Mat Material Material{
                              emissiveColor 1 0 0    // coded by
                        }                               Q2.color
                  }
            QuantizationParameter USE Q1
            Shape{
                  geometry  Cone {}
```

```
                  Appearance Appearance{
                        DEF Mat Material Material{
                              emissiveColor 0 1 0    // coded by
                        }                                Q1.color
                  }
            Shape{
                  geometry  Sphere {}
                  Appearance Appearance{
                        DEF Mat Material Material{
                              emissiveColor 0 0 1    // coded by
                        }                              Q2.color
                  }
            ]
}
```

l7.2.5.1.2.10 StreamingText

l7.2.5.1.2.10.1 Semantic Table

StreamingText {			
ExposedField	MFString	**url**	NULL
ExposedField	SFNode	**fontStyle**	NULL
ExposedField	SFBool	**ucs_2**	FALSE
}			

λ 7.2.5.1.2.10.2 Main Functionality

The **StreamingText** node defines a time dependent streaming text and parameters used to specify the text properties.

λ 7.2.5.1.2.10.3 Detailed Semantics

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

The **ucs_2** field selects UCS-2 format when TRUE and UTF-8 format when FALSE. The characters in the data stream will be specified using the 2-byte Basic Multilingual Plane (BMP) specifications (UCS-2) or the UCS Transformation Format 8 (UTF-8) as given by ISO/IEC 10646. In the BMP (or UCS-2), the basic Latin characters are covered within the span of 0x0020 - 0x007E. For example, the letter “a” is encoded as 0x0061 while the letter “A” is encoded as 0x0041. Numerals 0-9 are covered by representations 0x0030-0x0039. The 2-byte (UCS-2 or UTF-8) representation also covers a variety of international character sets, for example, basic and extended Greek, Cyrillic, basic and extended Latin, Hiragana, Katakana, etc. Mathematical operators and characters are also covered within this set. In UTF-8, the UCS-2 representations spanning 0x0000 - 0x007F are mapped to the 1-byte representation 0x00-0x7F. For example, the letter “a” is represented as 0x61 while the letter “A” is represented as 0x41. A 2-byte UTF-8 representation is necessary for other characters beyond this span within the BMP space. The StreamingText node supports the formatting capability of character value 13 representing CR-LF.

The **fontStyle** field contains one **FontStyle2D** node that specifies the font type, font size, font style, spacing and language.

l7.2.5.1.2.11 Valuator

l7.2.5.1.2.11.1 Semantic Table

Valuator {			
eventIn	SFBool	**inSFBool**	NULL
eventIn	SFColor	**inSFColor**	NULL

eventIn	MFColor	**inMFColor**	NULL
eventIn	SFFloat	**inSFFloat**	NULL
eventIn	MFFloat	**inMFFloat**	NULL
eventIn	SFInt32	**inSFInt32**	NULL
eventIn	MFInt32	**inMFInt32**	NULL
eventIn	SFRotation	**inSFRotation**	NULL
eventIn	MFRotation	**inMFRotation**	NULL
eventIn	SFString	**inSFString**	NULL
eventIn	MFString	**inMFString**	NULL
eventIn	SFTime	**inSFTime**	NULL
EventIn	SFVec2f	**inSFVec2f**	NULL
EventIn	MFVec2f	**inMFVec2f**	NULL
eventIn	SFVec3f	**inSFVec3f**	NULL
eventIn	MFVec3f	**inMFVec3f**	NULL
exposedField	SFBool	**outSFBool**	FALSE
exposedField	SFColor	**outSFColor**	0, 0, 0
exposedField	MFColor	**outMFColor**	NULL
exposedField	SFFloat	**outSFFloat**	0
exposedField	MFFloat	**outMFFloat**	NULL
exposedField	SFInt32	**outSFInt32**	0
exposedField	MFInt32	**outMFInt32**	NULL
exposedField	SFRotation	**outSFRotation**	0
exposedField	MFRotation	**outMFRotation**	NULL
exposedField	SFString	**outSFString**	""
exposedField	MFString	**outMFString**	NULL
exposedField	SFTime	**outSFTime**	0
exposedField	SFVec2f	**outSFVec2f**	0, 0
exposedField	MFVec2f	**outMFVec2f**	NULL
exposedField	SFVec3f	**outSFVec3f**	0, 0, 0
exposedField	MFVec3f	**outMFVec3f**	NULL

}

λ 7.2.5.1.2.11.2 Main Functionality

A Valuator node can receive an event of any type, and this reception will trigger the **eventOut** of an event of any kind with a constant value. It can be seen as an event type adapter. One use of this node is the modification of the **SFInt whichChoice** field of a **Switch** node by an event. There is no interpolator or sensor node with an **SFInt 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 which **outInt** field is set to **1** and routed to the **whichChoice** field of the Switch. The return to the **0** state needs another **Valuator** node.

λ 7.2.5.1.2.11.3 Detailed Semantics

Upon reception of an event on any of the **in<typeIn>** fields, on each **out<typeOut>** connected to a **ROUTE**, an event will be generated. The value of this event is the current value of the **out<typeOut>** field. Note: the value of the **out<typeOut>** event bears no relationship with the value of the **in<typeIn>** event, even if **typeIn** and **typeOut** are the same. As such, this node does not do type casting.

7.2.5.1.3 Shared VRML Nodes

The following nodes have their semantic specified in ISO/IEC DIS 14772-1:1997 with further restrictions and extensions defined herein.

l7.2.5.1.3.1 Appearance

l7.2.5.1.3.1.1 Semantic Table

Appearance {			
exposedField	SFNode	**material**	NULL
exposedField	SFNode	**texture**	NULL
exposedField	SFNode	**textureTransform**	NULL
}			

l7.2.5.1.3.2 AudioClip

The **AudioClip** node is used to provide an interface for short snippets of audio to be used in an interactive scene, such as sounds triggered as “auditory icons” upon mouse clicks. It buffers up the audio generated by its children, so that it can provide random restart capability upon interaction.

λ 7.2.5.1.3.2.1 Semantic Table

AudioClip {			
exposedField	SFBool	**loop**	FALSE
exposedField	SFFloat	**pitch**	1
exposedField	SFTime	**startTime**	0
exposedField	SFTime	**stopTime**	0
exposedField	MFString	**url**	NULL
eventOut	SFTime	**duration_changed**	NULL
eventOut	SFBool	**isActive**	FALSE
}			

λ 7.2.5.1.3.2.2 Main Functionality

The **AudioClip** node provides a special “buffering” interface to streaming audio, to convert it into a non-streaming form so that it can be used interactively, such as for auditory feedback of event triggers or other interactive “sound effect” processes.

λ 7.2.5.1.3.2.3 Detailed Semantics

The semantics of this node are the semantics of the VRML node with the same name, with the following exceptions and additions:

The **children** field specifies one or more child Sounds to use as the sound clip corresponding to this node.

λ 7.2.5.1.3.2.4 Calculation

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, if **isReady** is set, the sound output begins at the beginning of the clip buffer and **isActive** is set to 1. At each time step thereafter, the value of the output buffer is 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 is reached, if **loop** is set, the audio begins again from the beginning of the clip buffer; if not, the playback ends.

The clip buffer is calculated as follows: when the node is instantiated, **isReady** is set to 0, and for the first **length** *[units]*, the audio input to this node is copied into the clip buffer; that is, after *t* seconds, where *t* < **length**, audio sample number *t* * **S** of channel *i* in the buffer contains 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** *[units]*, the input to this node has no effect and **isReady** is set to 1.

When the playback ends, either because **stopTime** is reached or because the end of the clip buffer is reached for a non-looping clip, the **isActive** fireld is set to 0.

When the playback is not active, the output of the node is all 0s. If the **startTime** is reached before the **isReady** field is set, the output of the node is all 0s.

If **pitch** is negative, the buffer is played backward, beginning with the last segment.

l7.2.5.1.3.3 Color

l7.2.5.1.3.3.1 Semantic Table

Color {			
exposedField	MFColor	**color**	NULL
}			

l7.2.5.1.3.4 ColorInterpolator

l7.2.5.1.3.4.1 Semantic Table

ColorInterpolator {			
eventIn	SFFloat	**set_fraction**	NULL
exposedField	MFFloat	**key**	NULL
exposedField	MFColor	**keyValue**	NULL
eventOut	SFColor	**value_changed**	NULL
}			

l7.2.5.1.3.5 FontStyle

l7.2.5.1.3.5.1 Semantic Table

FontStyle {			
field	MFString	**family**	["SERIF"]
field	SFBool	**horizontal**	TRUE
field	MFString	**justify**	["BEGIN"]
field	SFString	**language**	""
field	SFBool	**leftToRight**	TRUE
field	SFFloat	**size**	1
field	SFFloat	**spacing**	1
field	SFString	**style**	"PLAIN"
field	SFBool	**topToBottom**	TRUE
}			

l7.2.5.1.3.6 ImageTexture

l7.2.5.1.3.6.1 Semantic Table

ImageTexture {			
exposedField	MFString	**url**	NULL
field	SFBool	**repeatS**	TRUE
field	SFBool	**repeatT**	TRUE
}			

λ 7.2.5.1.3.6.2 Detailed Semantics

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

l7.2.5.1.3.7 MovieTexture

l7.2.5.1.3.7.1 Semantic Table

MovieTexture {

exposedField	SFBool	**loop**	FALSE
exposedField	SFFloat	**speed**	1
exposedField	SFTime	**startTime**	0
exposedField	SFTime	**stopTime**	0
exposedField	MFString	**url**	NULL
field	SFBool	**repeatS**	TRUE
field	SFBool	**repeatT**	TRUE
eventOut	SFTime	**duration_changed**	NULL
eventOut	SFBool	**isActive**	NULL

}

λ 7.2.5.1.3.7.2 Detailed Semantics

startTime and **stopTime** refer to the time base of the Scene Description and are relative to the Decoding Time Stamp of the BIFS Access Unit that contained this node. In that case the currently available Composition Unit when **startTime** is reached will be presented.

If the Scene Description and the attached Elementary Stream(s) refer to the same time base then this method allows precise reference to each Composition Unit. Otherwise this may not be possible.

The **loop exposedField**, when TRUE, specifies that the video sequence shall play continuously. Having displayed the final available time VOP, it shall begin the next loop by playing the first VOP. When **loop** is FALSE, playback shall occur once.

Loop shall be FALSE when the attached sequence is received from a source that prohibits looping.

The **speed exposedField** controls playback speed. A MovieTexture shall display frame 0 if speed is 0. For positive values of speed, the frame an active MovieTexture will display at time now corresponds to the frame at movie time (i.e., in the movie's local time system with frame 0 at time 0, at speed = 1):

fmod (now - startTime, duration/speed)

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

duration + fmod(now - startTime, duration/speed).

Time now is relative to the decoding time stamp of the BIFS Access Unit that contains this node, in the same time basis as startTime.

Speed shall have the value 1.0 in applications where the control of the speed is not possible.

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. (CH: What do we do if it is not 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 should 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. A negative **speed** specifies that the video sequence shall play backwards. However, content creators should note that this may not work for streaming movies or very large movie files.

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

l7.2.5.1.3.8 ScalarInterpolator

l7.2.5.1.3.8.1 Semantic Table

ScalarInterpolator {			
eventIn	SFFloat	**set_fraction**	NULL
exposedField	MFFloat	**key**	NULL
exposedField	MFFloat	**keyValue**	NULL
eventOut	SFFloat	**value_changed**	NULL
}			

l7.2.5.1.3.9 Shape

l7.2.5.1.3.9.1 Semantic Table

Shape {			
exposedField	SFNode	**appearance**	NULL
exposedField	SFNode	**geometry**	NULL
}			

l7.2.5.1.3.10 Sound

l7.2.5.1.3.10.1 Semantic Table

Sound {			
exposedField	SFVec3f	**direction**	0, 0, 1
exposedField	SFFloat	**intensity**	1
exposedField	SFVec3f	**location**	0, 0, 0
exposedField	SFFloat	**maxBack**	10
exposedField	SFFloat	**maxFront**	10
exposedField	SFFloat	**minBack**	1
exposedField	SFFloat	**minFront**	1
exposedField	SFFloat	**priority**	0
exposedField	SFNode	**source**	NULL
field	SFBool	**spatialize**	TRUE
}			

λ 7.2.5.1.3.10.2 Main Functionality

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 subtree to the rest of an audiovisual 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 MPEG-4 audio scene may be filtered and mixed before being spatially composited into the scene.

λ 7.2.5.1.3.10.3 Detailed Semantics

The semantics of this node are as the semantics of the VRML node of the same name, with the following exceptions and additions:

The **source** field allows the connection of an audio source containing the sound.

The **spatialize** field determines whether the Sound should be spatialized. If this flag is set, the Sound should 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 discussed in Subclause .

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 discussion in Subclause .

λ 7.2.5.1.3.10.4 Nodes above the Sound node

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.

λ 7.2.5.1.3.10.5 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]

a4[4] = a1[4]

a3[5] = a1[5].

l7.2.5.1.3.11 Switch

l7.2.5.1.3.11.1 Semantic Table

Switch {			
exposedField	MFNode	**choice**	NULL
exposedField	SFInt32	**whichChoice**	-1
}			

l7.2.5.1.3.12 Text

l7.2.5.1.3.12.1 Semantic Table

Text {			
exposedField	SFString	**string**	""
exposedField	MFFloat	**length**	NULL
exposedField	SFNode	**fontStyle**	NULL
exposedField	SFFloat	**maxExtent**	0
}			

l7.2.5.1.3.13 TextureCoordinate

l7.2.5.1.3.13.1 Semantic Table

TextureCoordinate {			
exposedField	MFVec2f	**point**	NULL
}			

l7.2.5.1.3.14 TextureTransform

l7.2.5.1.3.14.1 Semantic Table

TextureTransform {			
exposedField	SFVec2f	**center**	0, 0
exposedField	SFFloat	**rotation**	0
exposedField	SFVec2f	**scale**	1, 1
exposedField	SFVec2f	**translation**	0, 0
}			

l7.2.5.1.3.15 TimeSensor

l7.2.5.1.3.15.1 Semantic Table

TimeSensor {			
exposedField	SFTime	**cycleInterval**	1
exposedField	SFBool	**enabled**	TRUE
exposedField	SFBool	**loop**	FALSE
exposedField	SFTime	**startTime**	0
exposedField	SFTime	**stopTime**	0
eventOut	SFTime	**cycleTime**	NULL
eventOut	SFFloat	**fraction_changed**	NULL
eventOut	SFBool	**isActive**	NULL
eventOut	SFTime	**time**	NULL
}			

l7.2.5.1.3.16 TouchSensor

l7.2.5.1.3.16.1 Semantic Table

TouchSensor {			
exposedField	SFBool	**enabled**	TRUE
eventOut	SFVec3f	**hitNormal_changed**	NULL
eventOut	SFVec3f	**hitPoint_changed**	NULL
eventOut	SFVec2f	**hitTexCoord_changed**	NULL
eventOut	SFBool	**isActive**	NULL
eventOut	SFBool	**isOver**	NULL

eventOut	SFTime	**touchTime**	NULL

}

l7.2.5.1.3.17 WorldInfo

l7.2.5.1.3.17.1 Semantic Table

WorldInfo {

field	MFString	**info**	NULL
field	SFString	**title**	""

}

λ 7.2.5.2 2D Nodes

λ *7.2.5.2.1 2D Nodes Overview*

The 2D nodes are those nodes which may be used in 2D scenes and with nodes that permit the use of 2D nodes in 3D scenes.

7.2.5.2.2 2D MPEG-4 Nodes

7.2.5.2.2.1 Background2D

7.2.5.2.2.1.1 Semantic Table

Background2D {

eventIn	SFBool	**set_bind**	NULL
exposedField	MFString	**url**	NULL
eventOut	SFBool	**isBound**	NULL

}

λ 7.2.5.2.2.1.2 Main Functionality

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.

λ 7.2.5.2.2.1.3 Detailed Semantics

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 again.

The **url** specifies the stream used for the backdrop.
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).

This is not a geometry node and the top-left corner of the image is displayed at the top-left corner of the screen, regardless of the current transformation. Scaling and/or rotation do not have any effect on this node.

λ 7.2.5.2.2.1.4 Example

Changing the background for 5 seconds.

```
Group2D {
```

```
     children [
          …
DEF TIS TimeSensor {
StartTime 5.0
StopTime  10.0
}
DEF BG1 Background2D {…
            }
]
}
ROUTE TIS.isActive TO BG1.set_bind
```

l7.2.5.2.2.2 Circle

l7.2.5.2.2.2.1 Semantic Table

Circle {

exposedField	SFFloat	**radius**	1

}

λ 7.2.5.2.2.2.2 Main Functionality

This node draws a circle.

λ 7.2.5.2.2.2.3 Detailed Semantics

The radius field determines the radius of the rendered circle.

l7.2.5.2.2.3 Coordinate2D

l7.2.5.2.2.3.1 Semantic Table

Coordinate2D {

exposedField	MFVec2f	**point**	NULL

}

λ 7.2.5.2.2.3.2 Main Functionality

This node defines a set of 2D coordinates to be used in the coord field of geometry nodes.

λ 7.2.5.2.2.3.3 Detailed Semantics

The point field contains a list of points in the 2D coordinate space. See Subclause .

l7.2.5.2.2.4 Curve2D

l7.2.5.2.2.4.1 Semantic Table

Curve2D {

exposedField	SFNode	**points**	NULL
exposedField	SFInt32	**fineness**	0

}

λ 7.2.5.2.2.4.2 Main Functionality

This node is used to include 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:

,

where *x[j]* is the j^{th} x coordinate of the internal polygon, *n* is the number of points in the control polygon, *xc[i]* is the i^{th} 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 symmetrical formula yields the y coordinates.

λ 7.2.5.2.2.4.3 Detailed Semantics

The points field lists the vertices of the control polygon. The fineness field contains the number of points in the internal polygon which constitutes the Bezier interpolation of the control polygon. fineness should sensibly be greater than the number of points in the control polygon.

λ 7.2.5.2.2.4.4 Example

The following defines a 20-points Bezier approximation of a 4-points polygon.

```
geometry Curve2D {
      points Coordinate2D {
            point [ -10.00 0.00 0.00 50.00 15.00 25.00 25.00 15.00 ]
            fineness 20
      }
}
```

l7.2.5.2.2.5 DiscSensor

l7.2.5.2.2.5.1 Semantic Table

DiscSensor {

exposedField	SFBool	**autoOffset**	TRUE
exposedField	SFVec2f	**center**	0, 0
exposedField	SFBool	**enabled**	TRUE
exposedField	SFFloat	**maxAngle**	-1
exposedField	SFFloat	**minAngle**	-1
exposedField	SFFloat	**offset**	0
eventOut	SFBool	**isActive**	NULL
eventOut	SFRotation	**rotation_changed**	NULL
eventOut	SFVec3f	**trackPoint_changed**	NULL

}

λ 7.2.5.2.2.5.2 Main Functionality

This sensor enables to rotate the object in the 2D plane around an axis with a coordinate specified in the local coordinate system.

λ 7.2.5.2.2.5.3 Detailed Semantics

The detailed semantic is the one of the ISO/IEC DIS 14772-1:1997 Section 6.15, restricted to a 2D case.

l7.2.5.2.2.6 Form

l7.2.5.2.2.6.1 Semantic Table

Form {			
field	MFNode	**children**	NULL
exposedField	SFVec2f	**size**	-1, -1
field	MFInt32	**groups**	NULL
field	MFInt32	**constraint**	NULL
}			

λ 7.2.5.2.2.6.2 Main Functionality

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.

λ 7.2.5.2.2.6.3 Detailed Semantics

The **children MF2DNode** shall specify a list of nodes that are laid out. Note that the children's position is implicit and that order is important.

The **width** field specifies the width of the layout frame.

The **height** field specifies the 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 are numbered from 1 to n, 0 being reserved for a reference to the layout itself. One group is a list of child indices, terminated by a -1.

The **constraints** field specifies the list of constraints. One constraint is constituted by a **constraint type**, optionally followed by a distance, followed by the indices of the objects and groups it is to be applied on and terminated by a –1. The numbering scheme is:

- 0 for a reference to the layout,
- 1 to n for a reference to one of the children,
- n+1 to n+m for a reference to one of the m specified groups.

Constraints belong to two categories: alignment and distribution constraints.

Table 7-1: Alignment Constraints

Alignment Constraints	Type Index	Effect
AL: Align Left edges	0	The xmin of constrained components become equal to the xmin of the left-most component.
AH: Align centers Horizontally	1	The (xmin+xmax)/2 of constrained components become equal to the (xmin+xmax)/2 of the group of constrained components.
AR: Align Right edges	2	The xmax of constrained components become equal to the xmax of the right-most component.
AT: Align Top edges	3	The ymax of all constrained components become equal to the ymax of the top-most component.
AV: Align centers Vertically	4	The (ymin+ymax)/2 of constrained components become equal to the (ymin+ymax)/2 of the group of constrained components.
AB: Align Bottom edges	5	The ymin of constrained components become equal to the ymin of the bottom-most component.
ALspace: Align Left edges by specified space	6	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	7	The xmax of the second and following components become equal to the xmax of the right-most component minus the specified space.
ATspace: Align Top edges by specified space	8	The ymax of the second and following components become equal to the ymax of the top-most component minus the specified space.
ABspace: Align Bottom edges by specified space	9	The ymin of the second and following components become equal to the ymin of the bottom-most 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 7-2: Distribution Constraints

Distribution Constraints	Type Index	Effect
SH: Spread Horizontally	10	The differences between the xmin of each component and the xmax of the previous one become all equal. The first and the last component should be constrained horizontally already.
Shin: Spread Horizontally in container	11	The differences between the xmin of each component and the xmax of the previous one become all equal. References are the edges of the layout.
SHspace: Spread Horizontally by specified space	12	The difference between the xmin of each component and the xmax of the previous one become all equal to the specified space.
SV: Spread Vertically	13	The differences between the ymin of each component and the ymax of the previous one become all equal. The first and the last component should be constrained vertically already.
Svin: Spread Vertically in container	14	The differences between the ymin of each component and the ymax of the previous one become all equal. References are the edges of the layout.
SVspace: Spread Vertically by specified space	15	The difference between the ymin of each component and the ymax of the previous one become all equal to the specified space.

All objects start at the center of the layout. The constraints are then applied once in sequence.

λ 7.2.5.2.2.6.4 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 [
                              Shapes2D { use OBJ1 }
```

```
                        Shapes2D { use OBJ2 }
                        Shapes2D { use OBJ3 }
                        Shapes2D { use OBJ4 }
                        Shapes2D { use OBJ5 }
                    ]
                    Width 50 Height 55
                    Groups [ 1 3 -1]
                        Constraints [ 7 6 -1 9 1 -1 3 0 2 -1 0 0 2 -1 12 6
                        0 3 –1
                         12 7 0 4 -1 15 7 0 4 -1 13 -2 0 5 -1 14 -2 0 5 –1
                    ]
                }
            ]
        }
}
```

l7.2.5.2.2.7 Group2D

l7.2.5.2.2.7.1 Semantic Table

Group2D {			
eventIn	MFNode	**addChildren**	NULL
eventIn	MFNode	**removeChildren**	NULL
exposedField	MFNode	**children**	NULL
field	SFVec2f	**bboxCenter**	0, 0
field	SFVec2f	**bboxSize**	-1, -1
}			

λ 7.2.5.2.2.7.2 Main Functionality

The **Group2D** node is one of the grouping 2D nodes. It is, itself, a 2D scene. A **Group2D** node contains children nodes without introducing any transformation.

λ 7.2.5.2.2.7.3 Detailed Semantics

The **addChildren eventIn** specifies a list of 2D objects that must be added to the Group2D node.
The **removeChildren eventIn** specifies a list of 2D objects that must be removed from the Group2D node.
The **children** field is the current list of 2D objects contained in the Group2D node.
The **bboxCenter** field specifies the center of the bounding box and the **bboxSize** field specifies the width and the height of the bounding box. It is possible not to transmit the **bboxCenter** and **bboxSize** fields, but if they are transmitted, the corresponding box must contain all the children. The behaviour of the terminal in other cases is not specified. The bounding box semantic is described in more detail in Subclause .

λ 7.2.5.2.2.7.4 Example

This example illustrates a means of avoiding any 2D objects of a group to be, simultaneously, in any other one by **ROUTEing** the children of the first to the **removeChildren eventIn** of the second:

```
DEF GSrc Group2D {
   children [
    …
   ]
}
DEF GDst Group2D {
    children [
```

```
    ...
   ]
}
ROUTE GSrc.children  TO GDst.removeChildren
```

l7.2.5.2.2.8 Image2D

l7.2.5.2.2.8.1 Semantic Table

Image2D {

exposedField	MFString	**url**	NULL

}

λ 7.2.5.2.2.8.2 Main Functionality

This node includes an image in its native size, transmitted in a stream, in a 2D scene. It is different from an **ImageTexture** image in that the image is not scaled to fit the underlying geometry on which it is texture mapped.

λ 7.2.5.2.2.8.3 Detailed Semantics

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

This is not a geometry node and by default, the bottom-left corner of the image is drawn at (0,0). The **Image2D** node is not affected by rotation or scaling.

l7.2.5.2.2.9 IndexedFaceSet2D

l7.2.5.2.2.9.1 Semantic Table

IndexedFaceSet2D {

eventIn	MFInt32	**set_colorIndex**	NULL
eventIn	MFInt32	**set_coordIndex**	NULL
eventIn	MFInt32	**set_texCoordIndex**	NULL
exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL
exposedField	SFNode	**texCoord**	NULL
field	MFInt32	**colorIndex**	NULL
field	SFBool	**colorPerVertex**	TRUE
field	SFBool	**convex**	TRUE
field	MFInt32	**coordIndex**	NULL
field	MFInt32	**texCoordIndex**	NULL

}

λ 7.2.5.2.2.9.2 Main Functionality

The **IndexedFaceSet2D** node represents a 2D shape formed by constructing 2D faces (polygons) from points specified in the **coord** field.

λ 7.2.5.2.2.9.3 Detailed Semantics

IndexedFaceSet2D shall be specified in the local coordinate system and shall be affected by parent transformations. Each face of an **IndexedFaceSet2D** node shall have at least three vertices which do not coincide. Each polygon defined by the vertices of a face shall not be self-intersecting. The faces of a **IndexedFaceSet2D** node shall not overlap each other.

l7.2.5.2.2.10 IndexedLineSet2D

l7.2.5.2.2.10.1 Semantic Table

IndexedLineSet2D {			
eventIn	MFInt32	**set_colorIndex**	NULL
eventIn	MFInt32	**set_coordIndex**	NULL
exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL
field	MFInt32	**colorIndex**	NULL
field	SFBool	**colorPerVertex**	TRUE
field	MFInt32	**coordIndex**	NULL
}			

λ 7.2.5.2.2.10.2 Main Functionality

The IndexedLineSet node causes a collection of lines or polygons (depending on the properties2D node) to be rendered.

λ 7.2.5.2.2.10.3 Detailed Semantics

The coord field lists the vertices of the lines. When coordIndex is empty, the order of vertices is 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 follows with **IndexedLineSet**.

l7.2.5.2.2.11 Inline2D

l7.2.5.2.2.11.1 Semantic Table

Inline2D {			
exposedField	MFString	**url**	NULL
field	SFVec2f	**bboxCenter**	0, 0
field	SFVec2f	**bboxSize**	-1, -1
}			

λ 7.2.5.2.2.11.2 Main Functionality

Inline2D allows the inclusion of a 2D scene from an external source in the current 2D scene graph.

λ 7.2.5.2.2.11.3 Detailed Semantics

The **url** field specifies the data source to be used (see). The external source must contain a valid 2D BIFS scene, and may include BIFS-Updates and BIFS-Anim frames.
The **bboxCenter** and **bboxSize** semantics are specified in Subclause

λ 7.2.5.2.2.11.4 Example

Including a 2D scene from the BIFS stream whose the decriptor index is 5.

```
Group2D {
        ....
        Inline2D {
                        url "mpeg4od:5"
        }
        ...
}
```

l7.2.5.2.2.12 Layout

l7.2.5.2.2.12.1 Semantic Table

Layout {			
exposedField	MFNode	**children**	NULL
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
exposedField	SFBool	**smoothScroll**	FALSE
exposedField	SFBool	**loop**	FALSE
exposedField	SFBool	**scrollVertical**	TRUE
exposedField	SFFloat	**scrollRate**	0
eventIn	MFNode	**addChildren**	NULL
eventIn	MFNode	**removeChildren**	NULL
}			

λ 7.2.5.2.2.12.2 Main functionality

The **Layout** node specifies the placement (layout) of its children in various alignment modes as specified, for the Text children, by their FontStyle fields, and for non-text children, by the fields horizontal, justify, leftToRight, topToBottom and spacing present in this node. It also includes the ability to scroll its children horizontally or vertically.

λ 7.2.5.2.2.12.3 Detailed Semantics

The **children MF2DNode** shall specify a list of nodes that are laid out. 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 smaller-than-the-layout pieces. The preferred places for breaking a Text are spaces, tabs, 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 ISO/IEC DIS 14772-1:1997, Subclause 6.20, for complete semantics.

The **scrollRate** field specifies the scroll rate per second specified in the units determined by the CoordinateSystem node. 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 scroll out of the scroll frame will be deleted. When loop is TRUE, then the set of children is supposed to constitute the entire space to be scrolled. That space is considered as a cylinder. When scrollVertical is TRUE and loop is TRUE and scrollRate is

negative (top-to-bottom scrolling), then the bottom-most object will reappear on top of the layout frame as soon as the top-most object has scrolled entirely into the layout frame.

The **scrollVertical** field specifies whether the scrolling is done vertically or horizontally. When set to TRUE, the scrolling rate shall be understood as a vertical scrolling rate, and a positive rate shall mean scrolling to the top. When set to FALSE, the scrolling rate shall be understood 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.

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 anything else, 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 edge of following lines are placed at regular intervals of value spacing ⌶ 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 have only a width. Column size is the maximum width over all objects.

λ 7.2.5.2.2.12.4 Example

If wrap is FALSE:

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.

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:

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).

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 0, 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:

If scrollRate is positive, then the scrolling direction is left-to-right, and vice-versa.

If scrollVertical is FALSE:

If scrollRate is positive, then the scrolling direction is bottom-to-top, and vice-versa.

l7.2.5.2.2.13 LineProperties

l7.2.5.2.2.13.1 Semantic Table

LineProperties {

exposedField	SFColor	**lineColor**	0, 0, 0
exposedField	SFInt32	**lineStyle**	0
exposedField	SFFloat	**width**	1

}

λ 7.2.5.2.2.13.2 Main Functionality

The LineProperties node specifies line parameters used in 2D rendering. The fields apply to certain geometry2DNodes only.

λ 7.2.5.2.2.13.3 Detailed Semantics

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

The lineStyle field contains the number of the line style to apply to lines: the allowed values are:

- 0 = solid
- 1 = dashed
- 2 = dotted
- 3 = dashed-dotted
- 4 = dashed-dashed-dotted
- 5 = dashed-dotted-dotted

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.

l7.2.5.2.2.14 Material2D

l7.2.5.2.2.14.1 Semantic Table

Material2D {

exposedField	SFColor	**diffuseColor**	0.8, 0.8, 0.8
exposedField	SFBool	**filled**	FALSE

```
    exposedField  SFNode      linePropsNULL
    exposedField  SFNode      shadowsPropsNULL
    exposedField  SFFloat     transparency0
}
```

λ 7.2.5.2.2.14.2 Main Functionality

The **Material2D** node determines the characteristics of a rendered **Shape2D**. This node only appears inside an **Appearance** field, which only appears inside a **Shape2D** node.

λ 7.2.5.2.2.14.3 Detailed Semantics

The **diffuseColor** field specifies the color of the Shape.

The **filled** field determines if rendered nodes are filled or drawn using lines. This field affects **IndexedFaceSet2D**, **Circle** and **Rectangle**.

The **linePr**ops field contains information about line rendering. If the field is null, lines are drawn solid with a width of one pixel, in the color specified in **diffuseColor**. See the **LineProperties** node (see Subclause) for more information.

The **shadowProps** field contains information about the presence of shadows for the 2D geometries. If the field is null, then no shadow is drawn. See the **ShadowProperties** node for more information.

The **transparency** field specifies the transparency of the Shape.

l7.2.5.2.2.15 VideoObject2D

l7.2.5.2.2.15.1 Semantic Table

VideoObject2D		SF2DNode, SFStreamingNode		**10000, ?**			
field name		*field type*		*field data type*		*Q*	*A*
defID	***inID***	***outID***	***dynID***	*default value*	*min value*	*max value*	
loop		exposedField		SFBool			
000	**000**	**000**		FALSE			
speed		exposedField		SFFloat		0	
001	**001**	**001**		1.0	0	+I	
startTime		exposedField		SFTime			
010	**010**	**010**		0			
stopTime		exposedField		SFTime			
011	**011**	**011**		0			
url		exposedField		MFString			
100	**100**	**100**		[]			
duration_changed		eventOut		SFFloat			
		101		-1			
isActive		eventOut		SFBool			
		110		FALSE			

λ 7.2.5.2.2.15.2 Main Functionality

The **VideoObject2D** node includes a video sequence using its natural size into a 2D scene. It is different from an **MovieTexture** image in that the video sequence is not scaled to fit the underlying geometry on which it is texture mapped.

λ 7.2.5.2.2.15.3 Detailed Semantics

As soon as the movie is started, a **duration_changed eventOut** is sent. This indicates the duration of the movie in seconds. This **eventOut** value can be read to determine the duration

of a movie. A value of "-1" implies the movie has not yet loaded or the value is unavailable for some reason.
The **loop**, **startTime**, and **stopTime exposedFields** and the **isActive eventOut**, and their effects on the VideoObject2D node, are discussed in detail in the section describing “Time dependent nodes". The cycle of a VideoObject2D node is the length of time in seconds for one playing of the movie at the specified speed.
The **speed exposedField** indicates how fast the movie shall be played. A speed of 2 indicates the movie plays twice as fast. The **duration_changed eventOut** is not affected by the speed **exposedField**. **set_speed** events are ignored while the movie is playing. A negative speed implies that the movie will play backwards.
If a **VideoObject2D** node is inactive when the movie is first loaded, frame 0 of the movie texture is displayed if speed is non-negative or the last frame of the movie texture is shown if speed is negative. A **VideoObject2D** node shall display frame 0 if speed = 0. For positive values of speed, an active **VideoObject2D** node displays the frame at movie time t as follows (i.e., in the movie's local time system with frame 0 at time 0 with speed = 1):

t = (now - startTime) modulo (duration/speed)

If speed is negative, the VideoObject2D node displays the frame at movie time:

t = duration - ((now - startTime) modulo ABS(duration/speed))

When a **VideoObject2D** node becomes inactive, the frame corresponding to the time at which the **VideoObject2D** became inactive will remain visible.
The **url** field specifies the data source to be used (see).

l7.2.5.2.2.16 PlaneSensor2D

l7.2.5.2.2.16.1 Semantic Table

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	SFVec2f	**trackPoint_changed**	NULL

}

λ 7.2.5.2.2.16.2 Main Functionality

This sensor detects pointer device dragging and enables the dragging of objects on the 2D rendering plane.

λ 7.2.5.2.2.16.3 Detailed Semantics

The semantic is a restricted case for 2D of the PlaneSensor as defined in ISO/IEC DIS 14772-1:1997.

l7.2.5.2.2.17 PointSet2D

l7.2.5.2.2.17.1 Semantic Table

PointSet2D {

exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL

}

λ 7.2.5.2.2.17.2 Main Functionality

λ 7.2.5.2.2.17.3 Detailed Semantics

λ *7.2.5.2.2.18* *Position2DInterpolator*

λ 7.2.5.2.2.18.1 Semantic Table

Position2DInterpolator {			
eventIn	SFFloat	**set_fraction**	NULL
exposedField	MFFloat	**key**	NULL
exposedField	MFVec2f	**keyValue**	NULL
eventOut	SFVec2f	**value_changed**	NULL
}			

λ 7.2.5.2.2.18.2 Main Functionality

λ 7.2.5.2.2.18.3 Detailed Semantics

This interpolator works as the other interpolators.

l7.2.5.2.2.19 Proximity2DSensor

l7.2.5.2.2.19.1 Semantic Table

Proximity2DSensor {			
exposedField	SFVec2f	**center**	0, 0
exposedField	SFVec2f	**size**	0, 0
exposedField	SFBool	**enabled**	TRUE
eventOut	SFBool	**isActive**	NULL
eventOut	SFVec2f	**position_changed**	NULL
eventOut	SFFloat	**orientation_changed**	NULL
eventOut	SFTime	**enterTime**	NULL
eventOut	SFTime	**exitTime**	NULL
}			

λ 7.2.5.2.2.19.2 Main Functionality

λ 7.2.5.2.2.19.3 Detailed Semantics

λ *7.2.5.2.2.20* *Rectangle*

λ 7.2.5.2.2.20.1 Semantic Table

Rectangle {			
exposedField	SFVec2f	**size**	2, 2
}			

λ 7.2.5.2.2.20.2 Main Functionality

This node renders a rectangle.

λ 7.2.5.2.2.20.3 Detailed Semantics

The **size** field specifies the horizontal and vertical size of the rendered rectangle.

l7.2.5.2.2.21 ShadowProperties

ShadowProperties {

exposedField	SFVec2f	**shadowPos**	5, 5
exposedField	SFColor	**shadowColor**	0, 0, 0

}

λ 7.2.5.2.2.21.1 Main Functionality

The ShadowProperties node specifies shadow parameters used in 2D rendering. The fields apply to certain geometry2DNodes only.

λ 7.2.5.2.2.21.2 Detailed Semantics

The shadowPos determines an offset for the shadow of a rendered object.

The shadowColor determines the color with which the shadow the shadow should be drawn.

l7.2.5.2.2.22 Switch2D

l7.2.5.2.2.22.1 Semantic Table

Switch2D {

exposedField	MFNode	**choice**	NULL
exposedField	SFInt32	**whichChoice**	-1

}

λ 7.2.5.2.2.22.2 Main functionality

The Switch2D grouping node traverses zero or one of the 2D nodes specified in the **choice** field. All nodes under a Switch2D continue to receive and send events regardless of the choice of the traversed one.

λ 7.2.5.2.2.22.3 Detailed Semantics

The **choice** field specifies the switchable nodes list.
The **whichChoice** field specifies the index of the child to traverse, with the first child having index 0. If whichChoice is less than zero or greater than the number of nodes in the choice field, nothing is chosen.

l7.2.5.2.2.23 Transform2D

l7.2.5.2.2.23.1 Semantic Table

Transform2D {

eventIn	MFNode	**addChildren**	NULL
eventIn	MFNode	**removeChildren**	NULL
exposedField	SFVec2f	**center**	0, 0
exposedField	MFNode	**children**	NULL
exposedField	SFFloat	**rotationAngle**	0
exposedField	SFVec2f	**scale**	1, 1
exposedField	SFFloat	**scaleOrientation**	0
exposedField	SFFloat	**drawingOrder**	0
exposedField	SFVec2f	**translation**	0, 0
field	SFVec2f	**bboxCenter**	0, 0
field	SFVec2f	**bboxSize**	-1, -1

}

λ 7.2.5.2.2.23.2 Main Functionality

The **Transform2D** node allows the translation, rotation and scaling of 2D objects which form the children of an **Transform2D** node.

λ 7.2.5.2.2.23.3 Detailed Semantics

The **bboxCenter** and **bboxSize** semantics are specified in Subclause .

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 co-ordinate 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 **drawingOrder** field specifies the order in which this node's children are drawn *with respect to other objects in the scene* (see) . When a **Transform2D** node has more than one child node, its children are drawn in order. The exception to this rule occurs when one or more child node has an explicit **drawingOrder**. In this case, the explicit **drawingOrder** is respected for that child node without affecting the implicit drawing order of its siblings.

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.

l7.2.5.2.2.24 VideoObject2D

Semantic Table

VideoObject2D {

exposedField	SFBool	**loop**	FALSE
exposedField	SFFloat	**speed**	1
exposedField	SFTime	**startTime**	0
exposedField	SFTime	**stopTime**	0
exposedField	MFString	**url**	NULL
eventOut	SFFloat	**duration_changed**	-1
eventOut	SFBool	**isActive**	FALSE

}

1.4.2.18.2 Main Functionality

The VideoObject2D node includes a video sequence using its natural size into a 2D scene. It is different from an MovieTexture image in that the video sequence is not scaled to fit the underlying geometry on which it is texture mapped.

1.4.2.18.3 Detailed Semantics

As soon as the movie is started, a **duration_changed eventOut** is sent. This indicates the duration of the movie in seconds. This **eventOut** value can be read to determine the duration of a movie. A value of "-1" implies the movie has not yet loaded or the value is unavailable for some reason.

The **loop**, **startTime**, and **stopTime exposedFields** and the **isActive eventOut**, and their effects on the VideoObject2D node, are discussed in detail in the section describing "Time

dependent nodes". The cycle of a VideoObject2D node is the length of time in seconds for one playing of the movie at the specified speed.

The **speed exposedField** indicates how fast the movie shall be played. A speed of 2 indicates the movie plays twice as fast. The **duration_changed eventOut** is not affected by the speed **exposedField**. **set_speed** events are ignored while the movie is playing. A negative speed implies that the movie will play backwards.

If a **VideoObject2D** node is inactive when the movie is first loaded, frame 0 of the movie texture is displayed if speed is non-negative or the last frame of the movie texture is shown if speed is negative. A **VideoObject2D** node shall display frame 0 if speed = 0. For positive values of speed, an active **VideoObject2D** node displays the frame at movie time t as follows (i.e., in the movie's local time system with frame 0 at time 0 with speed = 1):

t = (now - startTime) modulo (duration/speed)

If speed is negative, the VideoObject2D node displays the frame at movie time:

t = duration - ((now - startTime) modulo ABS(duration/speed))

When a **VideoObject2D** node becomes inactive, the frame corresponding to the time at which the **VideoObject2D** became inactive will remain visible.
The **url** field specifies the data source to be used (see).

λ 7.2.5.3 3D Nodes

λ *7.2.5.3.1 3D Nodes Overview*

The 3D nodes are those nodes which may be used in 3D scenes.

*7.2.5.3.2 3D MPEG-4 Nodes

The following nodes are specific to MPEG-4.

l7.2.5.3.2.1 ListeningPoint

l7.2.5.3.2.1.1 Semantic Table

ListeningPoint {			
eventIn	SFBool	**set_bind**	NULL
exposedField	SFBool	**jump**	TRUE
exposedField	SFRotation	**orientation**	0, 0, 1, 0
exposedField	SFVec3f	**position**	0, 0, 10
field	SFString	**description**	""
eventOut	SFTime	**bindTime**	NULL
eventOut	SFBool	**isBound**	NULL
}			

λ 7.2.5.3.2.1.2 Main Functionality

The ListeningPoint 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.

λ 7.2.5.3.2.1.3 Detailed Semantics

λ *7.2.5.3.2.2 FBA*

λ 7.2.5.3.2.2.1 Semantic Table

FBA {

exposedField	SFNode	**face**	NULL
exposedField	SFNode	**body**	NULL

}

λ 7.2.5.3.2.2.2 Main Functionality

This node contains one face and one body. They reside in the same coordinate system. The face is subject to body motion.

λ 7.2.5.3.2.2.3 Detailed Semantics

face contains a Face node

body contains a Body node, not yet defined

l7.2.5.3.2.3 Face

l7.2.5.3.2.3.1 Semantic Table

Face {

exposedField	SFNode	**fit**	NULL
exposedField	SFNode	**fdp**	NULL
exposedField	SFNode	**fap**	NULL
eventOut	MFNode	**renderedFace**	NULL

}

λ 7.2.5.3.2.3.2 Main Functionality

Organizes definition and animation of a face. The FAP node is mandatory, the FDP node, definening 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 node is not specified, the default face model of the decoder is used.

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

λ 7.2.5.3.2.3.3 Detailed Semantics

fit Specifies the FIT node. When this field is non-null, the decoder should use the FIT compute the maximal set of FAPs before using the FAPs to compute the mesh.

fdp contains an FDP node

fap contains an FAP node

objectDescriptorID ID of the object descriptor linking to the stream carrying facial parameters

renderedFace Scenegraph of the face after it is rendered (all FAP’s applied)

*7.2.5.3.2.3.3.1 Interpolating FAPs

By including an FIT node in the FIT field, the encoder can specify an FAP interpolation graph and a set of functions that can be usedto interpolate FAP values from one set of FAPs to another set of FAPs.

The FAP interpolation graph (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 child node can be interpolated from parent node provided that all FAPs in the parent node are available. A FAP may appear in several nodes, and a node may have multiple parents.

For a node which 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 1. Each node has an ID. The numerical label on each incoming link indicates the order of these links.

Figure 7-13: An example FIG

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_0 to Node_N-1
          for (ordered Node_i's parent Node_k) {
              if (FAPs in Node_k have been interpolated or are available) {
                    interpolate Node_i from Node_k;  //using interpolation function table here
                    interpolation_count ++;
                    goto Point_1;
               }
           }
           Point_1: ;
        }
  } while (interpolation_count != 0);
```

Both the encoder and the decoder shall use the above interpolation process.

l7.2.5.3.2.4 FIT

l7.2.5.3.2.4.1 Semantic Table

FIT {			
exposedField	MFInt32	**FAPs**	NULL
exposedField	MFInt32	**Graph**	NULL
exposedField	MFInt32	**numeratorExp**	NULL
exposedField	MFInt32	**denominatorExp**	NULL
exposedField	MFInt32	**numeratorTerms**	NULL
exposedField	MFInt32	**denominatorTerms**	NULL
exposedField	MFFloat	**numeratorCoefs**	NULL
exposedField	MFFloat	**denominatorCoefs**	NULL
}			

λ 7.2.5.3.2.4.2 Main Functionality

Defines the rational functions that determine the interpolation of FIG-child FAP tables from their parents' FAPs. The graph structure is determined using the **graph** field.

Each directed link in an 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. These sets are defined by the FAPs field.

Then, there are m interpolation functions denoted as:
$f_1 = I_1(F_1, F_2, ..., F_n)$;
$f_2 = I_2(F_1, F_2, ..., F_n)$;
...
$f_m = I_m(F_1, F_2, ..., F_n)$;

Each interpolation function I() is in a rational polynomial form

,

where and are the numbers of polynomial products, and are the coefficient of the *i*th product. and are the power of in the *i*th product. Since rational polynomials form a complete functional space, any possible finite interpolation function can be represented in this form to any given precision.

Encoder should send an interpolation function table which contains all , , , , , to decoder for a predictable interpolation between FAPs.

The interpolation function described here can also be applied to define FAP-to-mesh interpolation for a face animation definition table (FAT). The FAT currently specified in W1825 represents each interpolation function as

,

where each is approximated by a piece-wise linear curve. This representation is a special case of the method described above.

λ 7.2.5.3.2.4.3 Detailed Semantics

FAPs a list of indices specifying which animation parameters form sets of FAPs. Each set of

FAP indeces is terminated by a -1.

Graph A list of pairs of intergers, specifying a directed arc between sets of

FAPs. The intergers refer to sets specified in the graph field. When more than one arc terminates at the same set, that is, when the second value in the pair is repeated, the arcs have precedence determined by their order in this field.

numeratorTerms The number of terms in the polynomials in the numerators of the rational functions controlling the parameter value. This field corresponds to *K*. Each parameter in FAPs must have one degree specified.

denomenatorTerms The number of terms in the polynomials in the denominator of the rational functions controlling the parameter value. This field corresponds to *P*. Each parameter in FAPs must have one degree specified.

numeratorExp The exponents of the polynomial terms in the numerator of the rational function controlling the parameter value. This list corresponds to . The list should have *n***K* terms in row order for each parameter specified in the FAPs field.

denomenatorExp The exponents of the polynomial terms in the denominator of the rational function controlling the parameter value. This list corresponds to . The list should have *n***P* terms in row order order for each parameter specified in the FAPs field.

numeratorCoefs The coefficients of the polynomial terms in the numerator of the rational function controlling the parameter value. This list corresponds to . The list should have *K* terms for each parameter specified in the FAPs field.

DenomenatorCoefs The coefficients of the polynomial terms in the numerator of the rational function controlling the parameter value. This list corresponds to . The list should have *P* terms order for each parameter specified in the FAPs field.

λ 7.2.5.3.2.4.4 Example

```
FIT {
        FAPs [ 4 -1 5]
        numeratorPower 2
        denominatorPower 2
        numeratorExp  [0 0 0 0 1 2 3 4]
        denominatorExp  [0 0 0 0 1 0 0 1]
        numeratorCoefs [5 6]
        denominatorCoefs[7 8]
}
```

This FIT defines how the lowering of the top midlip (a set of one FAP, the 4th), is modified in terms of the 5th FAP.

I(f1,f2,f3,f4)= (5 +6* f1*f2^2*f3^3*f4^4) / (7 + 8 * f1* f4)

l7.2.5.3.2.5 FAP

l7.2.5.3.2.5.1 Semantic Table

FAP {

exposedField	SFInt32	**visemeSelect**	0
exposedField	SFInt32	**expressionSelect**	0
exposedField	SFInt32	**viseme**	0
exposedField	SFInt32	**expression**	0
exposedField	SFInt32	**open_jaw**	0
exposedField	SFInt32	**lower_t_midlip**	0
exposedField	SFInt32	**raise_b_midlip**	0
exposedField	SFInt32	**stretch_l_corner**	0
exposedField	SFInt32	**stretch_r_corner**	0
exposedField	SFInt32	**lower_t_lip_lm**	0
exposedField	SFInt32	**lower_t_lip_rm**	0
exposedField	SFInt32	**lower_b_lip_lm**	0
exposedField	SFInt32	**lower_t_lip_rm**	0
exposedField	SFInt32	**raise_l_cornerlip**	0
exposedField	SFInt32	**raise_r_cornerlip**	0
exposedField	SFInt32	**thrust_jaw**	0
exposedField	SFInt32	**shift_jaw**	0
exposedField	SFInt32	**push_b_lip**	0
exposedField	SFInt32	**push_t_lip**	0
exposedField	SFInt32	**depress_chin**	0
exposedField	SFInt32	**close_t_l_eyelid**	0
exposedField	SFInt32	**close_t_r_eyelid**	0
exposedField	SFInt32	**close_b_l_eyelid**	0
exposedField	SFInt32	**close_b_r_eyelid**	0
exposedField	SFInt32	**yaw_l_eyeball**	0
exposedField	SFInt32	**yaw_r_eyeball**	0
exposedField	SFInt32	**pitch_l_eyeball**	0
exposedField	SFInt32	**pitch_r_eyeball**	0
exposedField	SFInt32	**thrust_l_eyeball**	0
exposedField	SFInt32	**thrust_r_eyeball**	0
exposedField	SFInt32	**dilate_l_pupil**	0
exposedField	SFInt32	**dilate_r_pupil**	0
exposedField	SFInt32	**raise_l_i_eyebrow**	0
exposedField	SFInt32	**raise_r_i_eyebrow**	0
exposedField	SFInt32	**raise_l_m_eyebrow**	0
exposedField	SFInt32	**raise_r_m_eyebrow**	0
exposedField	SFInt32	**raise_l_o_eyebrow**	0
exposedField	SFInt32	**raise_r_o_eyebrow**	0
exposedField	SFInt32	**squeeze_l_eyebrow**	0
exposedField	SFInt32	**squeeze_r_eyebrow**	0
exposedField	SFInt32	**puff_l_cheek**	0
exposedField	SFInt32	**puff_r_cheek**	0
exposedField	SFInt32	**lift_l_cheek**	0
exposedField	SFInt32	**lift_r_cheek**	0
exposedField	SFInt32	**shift_tongue_tip**	0
exposedField	SFInt32	**raise_tongue_tip**	0
exposedField	SFInt32	**thrust_tongue_tip**	0
exposedField	SFInt32	**raise_tongue**	0
exposedField	SFInt32	**tongue_roll**	0
exposedField	SFInt32	**head_pitch**	0
exposedField	SFInt32	**head_yaw**	0
exposedField	SFInt32	**head_roll**	0
exposedField	SFInt32	**lower_t_midlip**	0
exposedField	SFInt32	**raise_b_midlip_o**	0
exposedField	SFInt32	**stretch_l_cornerlip**	0
exposedField	SFInt32	**stretch_r_cornerlip_o**	0
exposedField	SFInt32	**lower_t_lip_lm_o**	0
exposedField	SFInt32	**lower_t_lip_rm_o**	0
exposedField	SFInt32	**raise_b_lip_lm_o**	0

exposedField	SFInt32	**raise_b_lip_rm_o**	0
exposedField	SFInt32	**raise_l_cornerlip_o**	0
exposedField	SFInt32	**raise_r_cornerlip_o**	0
exposedField	SFInt32	**stretch_l_nose**	0
exposedField	SFInt32	**stretch_r_nose**	0
exposedField	SFInt32	**raise_nose**	0
exposedField	SFInt32	**bend_nose**	0
exposedField	SFInt32	**raise_l_ear**	0
exposedField	SFInt32	**raise_r_ear**	0
exposedField	SFInt32	**pupil_l_ear**	0
exposedField	SFInt32	**pupil_r_ear**	0

}

λ 7.2.5.3.2.5.2 Main Functionality

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 parameters see MPEG-4 Visual (ISO/IEC 14496-2).

λ 7.2.5.3.2.5.3 Detailed Semantics

visemeSelect	selects a viseme
expressionSelect	selects an expression
FAP1Value	specifies the intensity for the viseme defined by **visemeSelect**.
FAP2Value	specifies the intensity for the expression defined by **expressionSelect.**
FAP3Value, FAP4Value, ..., FAP68Value	values for FAP 3, FAP 4, ...,FAP 68.

A FAP of value +I is assumed to be uninitialized.

l7.2.5.3.2.6 FDP

l7.2.5.3.2.6.1 Semantic Table

FDP {

exposedField	SFNode	**featurePointsCoord**	NULL
exposedField	SFNode	**textureCoord4FeaturePoints**	NULL
exposedField	SFNode	**calibrationMesh**	NULL
exposedField	SFNode	**faceTexture**	NULL
exposedField	MFNode	**FBADefTables**	NULL
exposedField	MFNode	**faceSceneGraph**	NULL

}

λ 7.2.5.3.2.6.2 Main Functionality

The FDP node defines the face model to be used at the receiver. Two options are supported:

- l calibration information is downloaded, so that the proprietary face of the receiver can be configured using facial feature points and optionally a 3D mesh or texture. In this case, the field featurePointsCoord has to be set. The field calibrationMesh is optional. If set, the calibrationMesh has to show a face in neutral position and the featurePointsCoord define the position of the feature points in the calibrationMesh. The calibrationMesh can be used by the decoder to calibrate the

shape of its own face model.If the face model has a texture map, the fields faceTexture and textureCoord4FeaturePoints have to be set.

- a face model is downloaded with the animation definition of the Facial Animation Parameters hence the fields FBADefTables and faceSceneGraph have to be set. This face model replaces the proprietary face model in the receiver. The faceSceneGraph has to have the face in its neutral position (all FAPs 0). Therefore, a decoder may decide to use the faceSceneGraph as a calibration mesh. If desired, the faceSceneGraph contains the texture map of the face. Fields other than FBADefTables and faceSceneGraph will not be evaluated if the coder uses this downloaded model. However, the encoder is also required to send information according to option 1 (at least featurePointsCoord) in order to allow a low-complexity decoder to adapt their own face model.

l 7.2.5.3.2.6.3 Detailed Semantics

featurePointsCoord contains a Coordinate node. Specifies feature points for the calibration of the proprietary face. 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 (e.g. feature point 3.14 before feature point 4.1).

textureCoord4FeaturePoints contains a TextureCoordinate node. Specifies the texture coordinates for the feature points.

calibrationMesh contains an IndexedFaceSet node. Specifies a 3D mesh for the calibration of the proprietary face model. All fields in the IndexedFaceSet node can be used as calibration information.

faceTexture contains an ImageTexture or PixelTexture node. Specifies texture to be applied on the proprietary face model.

FBADefTables contains FBADefTable nodes. If a face model is downloaded, the behavior of FAPs is defined in this field.

faceSceneGraph contains a Group node. Grouping node for face model rendered in the compositor. It has to contain the face model. The effect of Facial Animation Parameters is defined in the 'FBADefTables'field.

l7.2.5.3.2.7 FBADefTable

l7.2.5.3.2.7.1 Semantic Table

FBADefTable {			
field	SFInt32	**fapID**	1
field	SFInt32	**highLevelSelect**	1
exposedField	MFNode	**tables**	NULL
}			

λ 7.2.5.3.2.7.2 Main Functionality

Defines the behavior of an animation parameter on a downloaded face model by specifying displacement vectors for moved vertices inside IndexedFaceSet objects and/or specifying the field of Transform nodes to be updated.

λ 7.2.5.3.2.7.3 Detailed Semantics

fapID specifies the FAP, for which the animation behavior is defined in the 'tables' field.

highLevelSelect specifies the type of viseme or expression, if fapID is 1 or 2. In other cases this

field has got no meaning.

Tables contains an FBADefTransform or FBADefMesh node.

l7.2.5.3.2.8 FBADefTransform

l7.2.5.3.2.8.1 Semantic Table

FBADefTransform {			
field	SFNode	**faceSceneGraphNode**	NULL
field	SFInt32	**fieldId**	1
field	SFRotation	**fieldValue**	0, 0, 1, 0
}			

λ 7.2.5.3.2.8.2 Main Functionality

Defines, which field of an FBADefTransform is updated by an Facial Animation Parameter.

λ 7.2.5.3.2.8.3 Detailed Semantics

faceSceneGraph Node ID of the Transform node for which the animation is defined. The node must be part of faceScenegraph as defined in the FDP node.

FieldId specifies, which field in the Transform node is updated by the FAP during animation. Possible fields are translation, rotation, scale.

FieldValue is of type **SFVec3f** (if fieldId references the translation or scale field) or **SFRotation** (if fieldRef references the rotation field). The new node value of the Transform is in the case of :

fieldId==(translation or scale): NodeValue:= FAPValue+fieldValue

fieldId==(rotation): NodeValue $Theta_{new}$:= FAPValue+$Theta_{old}$

l7.2.5.3.2.9 FBADefMesh

l7.2.5.3.2.9.1 Semantic Table

FBADefMesh {			
field	SFNode	**faceSceneGraphNode**	NULL
field	MFInt32	**intervalBorders**	NULL
field	MFInt32	**coordIndex**	NULL
field	MFVec3f	**displacements**	NULL
}			

λ 7.2.5.3.2.9.2 Main Functionality

Defines the piece-wine linear motion trajectories for vertices of the IndexedFaceSet objects of the faceSceneGraph of the FDP node, which is deformed by an Facial Animation Parameter.

λ 7.2.5.3.2.9.3 Detailed Semantics

faceSceneGraph Node ID of the IndexedFaceSet node for which the animation is defined. The node must be part of faceSceneGraph as defined in the FDP node.

IntervalBorders interval borders for the piece-wise linear approximation in increasing order. Exactly one interval border must have the value 0.

coordIndex a list of indices into the Coordinate node of the IndexedFaceSet node specified by nodeId.

displacements for each vertex indexed in the coordIndex field, displacement vectors are given

for the intervals defined in the intervalBorders field. There must be exactly (num(IntervalBorders)-1)*num(coordIndex) values in this field.

Example:

FBADefMesh {

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 FBADefMesh 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 $500*(1\ 0\ 0)^T+100*(1.5\ 0\ 4)^T=(650\ 0\ 400)^T$. If the FAPValue is outside the given intervals, the boundary intervals are extended to +I or -I, as appropriate.

*7.2.5.3.3 3D VRML Nodes

Some nodes have their semantic specified in ISO/IEC DIS 14772-1:1997 with further restrictions and extensions defined herein.

l7.2.5.3.3.1 Background

l7.2.5.3.3.1.1 Semantic Table

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

λ 7.2.5.3.3.1.2 Detailed Semantics

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

l7.2.5.3.3.2 Billboard

l7.2.5.3.3.2.1 Semantic Table

Billboard {			
eventIn	MFNode	**addChildren**	NULL
eventIn	MFNode	**removeChildren**	NULL
exposedField	SFVec3f	**axisOfRotation**	0, 1, 0
exposedField	MFNode	**children**	NULL
field	SFVec3f	**bboxCenter**	0, 0, 0

field	SFVec3f	**bboxSize**	-1, -1, -1

}

l7.2.5.3.3.3 Box

l7.2.5.3.3.3.1 Semantic Table

Box {

field	SFVec3f	**size**	2, 2, 2

}

l7.2.5.3.3.4 Collision

l7.2.5.3.3.4.1 Semantic Table

Collision {

eventIn	MFNode	**addChildren**	NULL
eventIn	MFNode	**removeChildren**	NULL
exposedField	MFNode	**children**	NULL
exposedField	SFBool	**collide**	TRUE
field	SFVec3f	**bboxCenter**	0, 0, 0
field	SFVec3f	**bboxSize**	-1, -1, -1
field	SFNode	**proxy**	NULL
eventOut	SFTime	**collideTime**	NULL

}

l7.2.5.3.3.5 Cone

l7.2.5.3.3.5.1 Semantic Table

Cone {

field	SFFloat	**bottomRadius**	1
field	SFFloat	**height**	2
field	SFBool	**side**	TRUE
field	SFBool	**bottom**	TRUE

}

l7.2.5.3.3.6 Coordinate

l7.2.5.3.3.6.1 Semantic Table

Coordinate {

exposedField	MFVec3f	**point**	NULL

}

l7.2.5.3.3.7 CoordinateInterpolator

l7.2.5.3.3.7.1 Semantic Table

CoordinateInterpolator {

eventIn	SFFloat	**set_fraction**	NULL
exposedField	MFFloat	**key**	NULL
exposedField	MFVec3f	**keyValue**	NULL
eventOut	MFVec3f	**value_changed**	NULL

}

l7.2.5.3.3.8 Cylinder

l7.2.5.3.3.8.1 Semantic Table

Cylinder {			
field	SFBool	**bottom**	TRUE
field	SFFloat	**height**	2
field	SFFloat	**radius**	1
field	SFBool	**side**	TRUE
field	SFBool	**top**	TRUE
}			

l7.2.5.3.3.9 DirectionalLight

l7.2.5.3.3.9.1 Semantic Table

DirectionalLight {			
exposedField	SFFloat	**ambientIntensity**	0
exposedField	SFColor	**color**	1, 1, 1
exposedField	SFVec3f	**direction**	0, 0, -1
exposedField	SFFloat	**intensity**	1
exposedField	SFBool	**on**	TRUE
}			

l7.2.5.3.3.10 ElevationGrid

l7.2.5.3.3.10.1 Semantic Table

ElevationGrid {			
eventIn	MFFloat	**set_height**	NULL
exposedField	SFNode	**color**	NULL
exposedField	SFNode	**normal**	NULL
exposedField	SFNode	**texCoord**	NULL
field	MFFloat	**height**	NULL
field	SFBool	**ccw**	TRUE
field	SFBool	**colorPerVertex**	TRUE
field	SFFloat	**creaseAngle**	0
field	SFBool	**normalPerVertex**	TRUE
field	SFBool	**solid**	TRUE
field	SFInt32	**xDimension**	0
field	SFFloat	**xSpacing**	1
field	SFInt32	**zDimension**	0
field	SFFloat	**zSpacing**	1
}			

l7.2.5.3.3.11 Extrusion

l7.2.5.3.3.11.1 Semantic Table

Extrusion {			
eventIn	MFVec2f	**set_crossSection**	NULL
eventIn	MFRotation	**set_orientation**	NULL
eventIn	MFVec2f	**set_scale**	NULL
eventIn	MFVec3f	**set_spine**	NULL
field	SFBool	**beginCap**	TRUE
field	SFBool	**ccw**	TRUE
field	SFBool	**convex**	TRUE

field	SFFloat	**creaseAngle**	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

}

l7.2.5.3.3.12 Group

l7.2.5.3.3.12.1 Semantic Table

Group {

EventIn	MFNode	**addChildren**	NULL
EventIn	MFNode	**removeChildren**	NULL
ExposedField	MFNode	**children**	NULL
Field	SFVec3f	**bboxCenter**	0, 0, 0
Field	SFVec3f	**bboxSize**	-1, -1, -1

}

λ 7.2.5.3.3.12.2 Detailed Semantics

If multiple subgraphs containing audio content (ie, **Sound** nodes) are children of a **Group** node, the sounds are combined as follows:

If all of the children have equal numbers of channels, or are each a spatially-presented Sound, the Sound outputs of the children sum to create the audio output of this node.

If the children do not have equal numbers of audio channels, or some children, but not all, are spatially presented sounds, the semantics are TBD.

l7.2.5.3.3.13 IndexedFaceSet

l7.2.5.3.3.13.1 Semantic Table

IndexedFaceSet {

eventIn	MFInt32	**set_colorIndex**	NULL
eventIn	MFInt32	**set_coordIndex**	NULL
eventIn	MFInt32	**set_normalIndex**	NULL
eventIn	MFInt32	**set_texCoordIndex**	NULL
exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL
exposedField	SFNode	**normal**	NULL
exposedField	SFNode	**texCoord**	NULL
field	SFBool	**ccw**	TRUE
field	MFInt32	**colorIndex**	NULL
field	SFBool	**colorPerVertex**	TRUE
field	SFBool	**convex**	TRUE
field	MFInt32	**coordIndex**	NULL
field	SFFloat	**creaseAngle**	0
field	MFInt32	**normalIndex**	NULL
field	SFBool	**normalPerVertex**	TRUE
field	SFBool	**solid**	TRUE
field	MFInt32	**texCoordIndex**	NULL

}

λ 7.2.5.3.3.13.2 Main Functionality

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.

λ 7.2.5.3.3.13.3 Detailed Semantics

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 followed by a -1.

If the **coordIndex** field is NULL, the vertices of the **coord** field are laid out in their respective order to specify one face.

If the **color** field is NULL and there is a **Material** defined for the **Appearance** affecting this **IndexedFaceSet**, then the **emissiveColor** of the **Material** shall be used to draw the faces.

l7.2.5.3.3.14 IndexedLineSet

l7.2.5.3.3.14.1 Semantic Table

IndexedLineSet {

eventIn	MFInt32	**set_colorIndex**	NULL
eventIn	MFInt32	**set_coordIndex**	NULL
exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL
field	MFInt32	**colorIndex**	NULL
field	SFBool	**colorPerVertex**	TRUE
field	MFInt32	**coordIndex**	NULL

}

l7.2.5.3.3.15 Inline

l7.2.5.3.3.15.1 Semantic Table

Inline {

exposedField	MFString	**url**	NULL
field	SFVec3f	**bboxCenter**	0, 0, 0
field	SFVec3f	**bboxSize**	-1, -1, -1

}

λ 7.2.5.3.3.15.2 Detailed Semantics

The **url** field specifies the data source to be used (see). The external source must contain a valid BIFS scene, and may include BIFS-Updates and BIFS-Anim frames

l7.2.5.3.3.16 LOD

l7.2.5.3.3.16.1 Semantic Table

LOD {

exposedField	MFNode	**level**	NULL

field	SFVec3f	**center**	0, 0, 0
field	MFFloat	**range**	NULL
field	MFFloat	**fpsRange**	NULL

}

l7.2.5.3.3.17 Material

l7.2.5.3.3.17.1 Semantic Table

Material {

exposedField	SFFloat	**ambientIntensity**	0.2
exposedField	SFColor	**diffuseColor**	0.8, 0.8, 0.8
exposedField	SFColor	**emissiveColor**	0, 0, 0
exposedField	SFFloat	**shininess**	0.2
exposedField	SFColor	**specularColor**	0, 0, 0
exposedField	SFFloat	**transparency**	0

}

l7.2.5.3.3.18 Normal

l7.2.5.3.3.18.1 Semantic Table

Normal {

exposedField	MFVec3f	**vector**	NULL

}

l7.2.5.3.3.19 NormalInterpolator

l7.2.5.3.3.19.1 Semantic Table

NormalInterpolator {

eventIn	SFFloat	**set_fraction**	NULL
exposedField	MFFloat	**key**	NULL
exposedField	MFVec3f	**keyValue**	NULL
eventOut	MFVec3f	**value_changed**	NULL

}

l7.2.5.3.3.20 OrientationInterpolator

l7.2.5.3.3.20.1 Semantic Table

OrientationInterpolator {

eventIn	SFFloat	**set_fraction**	NULL
exposedField	MFFloat	**key**	NULL
exposedField	MFRotation	**keyValue**	NULL
eventOut	SFRotation	**value_changed**	NULL

}

l7.2.5.3.3.21 PointLight

l7.2.5.3.3.21.1 Semantic Table

PointLight {

exposedField	SFFloat	**ambientIntensity**	0
exposedField	SFVec3f	**attenuation**	1, 0, 0

exposedField	SFColor	**color**	1, 1, 1
exposedField	SFFloat	**intensity**	1
exposedField	SFVec3f	**location**	0, 0, 0
exposedField	SFBool	**on**	TRUE
exposedField	SFFloat	**radius**	100

}

l7.2.5.3.3.22 PointSet

l7.2.5.3.3.22.1 Semantic Table

PointSet {

exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL

}

l7.2.5.3.3.23 PositionInterpolator

l7.2.5.3.3.23.1 Semantic Table

PositionInterpolator {

eventIn	SFFloat	**set_fraction**	NULL
exposedField	MFFloat	**key**	NULL
exposedField	MFVec3f	**keyValue**	NULL
eventOut	SFVec3f	**value_changed**	NULL

}

l7.2.5.3.3.24 ProximitySensor

l7.2.5.3.3.24.1 Semantic Table

ProximitySensor {

exposedField	SFVec3f	**center**	0, 0, 0
exposedField	SFVec3f	**size**	0, 0, 0
exposedField	SFBool	**enabled**	TRUE
eventOut	SFBool	**isActive**	NULL
eventOut	SFVec3f	**position_changed**	NULL
eventOut	SFRotation	**orientation_changed**	NULL
eventOut	SFTime	**enterTime**	NULL
eventOut	SFTime	**exitTime**	NULL

}

l7.2.5.3.3.25 Sphere

l7.2.5.3.3.25.1 Semantic Table

Sphere {

field	SFFloat	**radius**	1

}

l7.2.5.3.3.26 SpotLight

l7.2.5.3.3.27 Semantic Table

SpotLight {

exposedField	SFFloat	**ambientIntensity**	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
exposedField	SFVec3f	**location**	0, 0, 0
exposedField	SFBool	**on**	TRUE
exposedField	SFFloat	**radius**	100

}

l7.2.5.3.3.28 Transform

l7.2.5.3.3.28.1 Semantic Table

Transform {

eventIn	MFNode	**addChildren**	NULL
eventIn	MFNode	**removeChildren**	NULL
exposedField	SFVec3f	**center**	0, 0, 0
exposedField	MFNode	**children**	NULL
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
field	SFVec3f	**bboxCenter**	0, 0, 0
field	SFVec3f	**bboxSize**	-1, -1, -1

}

λ 7.2.5.3.3.28.2 Detailed Semantics

If some of the child subgraphs contain audio content (ie, 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. The spatialized outputs of the children nodes sum equally to produce the output at this node.

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.

l7.2.5.3.3.29 Viewpoint

l7.2.5.3.3.29.1 Semantic Table

Viewpoint {

eventIn	SFBool	**set_bind**	NULL
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**	NULL
eventOut	SFBool	**isBound**	NULL

}

λ 7.2.5.3.3.29.2 Detailed Semantics

Currently, scenes may contain only one **Viewpoint** node. This **Viewpoint** may not be a child node.

λ 7.2.5.4 Mixed 2D/3D Nodes

λ *7.2.5.4.1 Mixed 2D/3D Nodes Overview*

Mixed 2D and 3D nodes enable to build scenes made up of 2D and 3D primitives together. In particular, it is possible to view simultaneously several rendered 2D and 3D scenes, to use a rendered 2D or 3D scene as a texture map, or to render a 2D scene in a 3D local coordinate plane.

7.2.5.4.2 2D/3D MPEG-4 Nodes

7.2.5.4.2.1 Layer2D

*7.2.5.4.2.1.1 Semantic Table

Layer2D {

exposedField	MFNode	**children**	NULL
exposedField	MFNode	**childrenLayer**	NULL
exposedField	SFVec2f	**size**	-1, -1
exposedField	SFVec2f	**translation**	0, 0
exposedField	SFInt32	**depth**	0

}

λ 7.2.5.4.2.1.2 Main Functionality

The Layer2D node is a transparent rendering rectangle region on the screen where a 2D scene is shown. The Layer2D is part of the layers hierarchy, and can be composited in a 2D environment with depth.

λ 7.2.5.4.2.1.3 Detailed Semantics

Layer 2D and Layer3D nodes enable to compose in a 2D space with depth multiple 2D and 3D scenes. This enables, for instance; to have 2D interfaces to a 2D scene; or 3D interfaces to a 2D scene, or to view a 3D scene from different view points in the same scene.
Interaction with objects drawn inside a Layer node is enabled only on the top most layer of the scene at a given position of the rendering area. This means that it is impossible to interact with an object behind another layer.
The **`children`** field can have as value any 2D children nodes that defines a 2D scene. As for any 2D grouping node, the order of children is the order of drawing for 2D primitives.

The **`childrenLayer`** field can take either a 2D or 3D layer node as value.

The layering of the 2D and 3D layers is specified by the **`translation`** and **`depth`** fields. The units of 2D scenes are used for the translation parameter. The **`size`** parameter is given in floating point number which expresses a fraction of the width and height of the parent Layer. In case of a layer at the root of the hierarchy, the fraction is a fraction of the screen rendering area. A size of -1 in one direction, means that the **`Layer`** 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`**, the behaviours are defined exactly as for objects which 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 behaviours triggered by the shared sensors as well as

other behaviours that apply on objects shared between several layers apply on all layers containing these objects.

All the 2D objects under a same **Layer2D** node form a single composed object. This composed object is viewed by other objects as 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 children of A.

l7.2.5.4.2.2 Layer3D

l7.2.5.4.2.2.1 Semantic Table

Layer3D {

exposedField	MFNode	**children**	NULL
exposedField	MFNode	**childrenLayer**	NULL
exposedField	SFVec2f	**translation**	0, 0
exposedField	SFInt32	**depth**	0
exposedField	SFVec2f	**size**	-1, -1
eventIn	SFNode	**background**	NULL
eventIn	SFNode	**fog**	NULL
eventIn	SFNode	**navigationInfo**	NULL
eventIn	SFNode	**viewpoint**	NULL

}

λ 7.2.5.4.2.2.2 Main Functionality

The Layer3D node is a transparent rendering rectangle region on the screen where a 3D scene is shown. The Layer3D is part of the layers hierarchy, and can be composited in a 2D environment with depth

λ 7.2.5.4.2.2.3 Detailed Semantics

Layer2D and 3D are composed as described in the Layer2D specification. For navigation in a Layer3D, the same principle as for interaction applies. It is possible to navigate any Layer3D node that appears as the front layer at a given position on the rendering area. A terminal must provide a way to select an active layer among all the displayed Layer3D. Once this layer is selected, the navigation acts on this layer. For instance; if a mouse pointing device is used, the active layer for navigation may be the front layer under the starting position of the mouse dragging action for navigating the 3D scene.
The **children** field can have as value any 3D children nodes that define a 3D scene.

The **childrenLayer** field can have either a 2D or 3D layer as values. The layering of the 2D and 3D layers is specified by the **translation** and **depth** fields. The **translation** field is expressed, as in the case of the Layer2D in terms of 2D units.

The **size** parameter has the same semantic and units as in the Layer2D.

A Layer3D stores the stack of bindable leaf nodes of the children scene of the layer. All bindable leaf nodes are **eventIn** fields of the **Layer3D** node. At run-time, these fields take the value of the currently bound bindable leaf nodes for the 3D scene that is a child of the **Layer3D** node. This will allow, for instance, to set a current viewpoint to a **Layer3D**, in response to some event. Note that this cannot be achieved by a direct use of the `set_bind` **eventIn** of the Viewpoint nodes since scenes or nodes can be shared between different layers. If a set_bind TRUE event is sent to the set_bind eventIn of any of the bindable leaf nodes, then all Layer3D nodes having this node as a children node will set this node as the current bindable leaf node.

In the case where a 3D scene or object is shared between several **Layer3D**, the behaviours are defined exactly as for objects which are multiply referenced using the DEF/USE mechanism. . A sensor triggers an event whenever the sensor is triggered in any of the **Layer3D** in which it is contained. The behaviours triggered by the shared sensors as well as other behaviours that apply on objects shared between several layers apply on all layers containing these objects.

All the 3D objects under a same **Layer3D** node form a single composed object. This composed object is viewed by other objects as a single object.

Figure 7-14: Three Layer2D and Layer3D examples. Layer2D are signaled by a plain line, Layer3D with 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 overlaping Layer3D.

l7.2.5.4.2.3 Composite2DTexture

l7.2.5.4.2.3.1 Semantic Table

Composite2DTexture {			
exposedField	MFNode	**children**	NULL
exposedField	SFVec2f	**size**	-1, -1
}			

λ 7.2.5.4.2.3.2 Main Functionality

The composite 2D texture node represents a texture mapped onto a 3D object which is composed of a 2D scene.

λ 7.2.5.4.2.3.3 Detailed Semantics

All the behaviors and user interaction are enabled when using a Composite2DTexture. However, no sensor can be then attached on the object on which the texture is projected.
The **children2D** field of type **MF2DNode** is the list of 2D children nodes that defines the 2D scene to be mapped onto the 3D object. As for any 2D grouping node, the order of children is the order of drawing for 2D primitives.

The **size** field specifies the size in pixels of this map. If left as default value, an undefined size will be used. This is a clue for the content creator to define the quality of the texture mapping.

Figure 7-15: A Composite2DTexture example. The 2D scene is projected on the 3D cube

l7.2.5.4.2.4 Composite3DTexture

l7.2.5.4.2.4.1 Semantic Table

Composite3DTexture {			
exposedField	MFNode	**children**	NULL
exposedField	SFVec2f	**size**	-1, -1
eventIn	SFNode	**background**	NULL
eventIn	SFNode	**fog**	NULL
eventIn	SFNode	**navigationInfo**	NULL
eventIn	SFNode	**Viewpoint**	NULL
}			

λ 7.2.5.4.2.4.2 Main Functionality

The composite 3D texture node represents a texture mapped onto a 3D object which is composed of a 3D scene.

λ 7.2.5.4.2.4.3 Detailed Semantics

Behaviors and user interaction are enabled when using a Composite3DTexture. However, no user, navigation is possible on the textured scene. Moreover, no sensor can be attached to the object on which the texture is mapped.

The **children** field of type MF3DNode is the list of 3D root and children nodes that define the 3D scene to be mapped onto the 3D object.

The size field specifies the size in pixels of this map. If left as default value, an undefined size will be used. This is a clue for the content creator to define the quality of the texture mapping.

The 4 following fields represent the current values of the bindable leaf nodes used in the 3D scene. The semantic is the same as in the case of the Layer3D node. This node can only be used as a texture field of an Appearence node.

Figure 7-16: A Composite3Dtexture example: The 3D view of the earth is projected onto the 3D cube

l7.2.5.4.2.5 CompositeMap

l7.2.5.4.2.5.1 Semantic Table

CompositeMap {			
exposedField	MFNode	**children2D**	NULL
exposedField	SFVec2f	**sceneSize**	-1, -1
}			

λ 7.2.5.4.2.5.2 Main Functionality

λ 7.2.5.4.2.5.3 The CompositeMap node is used for a 2D scene appearing on a plane in a 3D scene. A similar functionnality can be achieved with the CompositeTexture2D, but when using the CompsiteMap, you do not have to use texture projection to draw the 2D scene in the local XY planeDetailed Semantics

When using a composite Map, all the behaviors of 2D objects are similar to those of the 2D scene as drawn in a 2D layer.

The **children** field of type **MF2DNode** is the list of 2D root and children nodes that define the 2D scene to be rendered in the local XY coordinate plane. As for any 2D grouping node, the order of children is the order of drawing for 2D primitives.

The **sceneSize** field specifies the size in the local 3D coordinate system of the rendering area where the 2D composited scene needs to be rendered. If left as default value, the scene rendering area is defined as the rectangle which diagonal is delimited by the origin of the local coordinate system and point (1,1,0) in the local coordinate system.

Figure 7-17: A CompositeMap example: The 2D scene as defined in Fig. yyy composed of an image, a logo, and a text, is drawn in the local X,Y plane of the back wall.

l7.2.6 Node Coding Parameters

l7.2.6.1 Table Semantic

The Node Data Type tables contain the following parameters:

- Name of NDT
- Number of nodes in NDT
- Number of nbits to address a node in NDT
- NodeID for each Node

- Number of DEF, IN, OUT and DYN fields as well as the number of bits to encode them.

The Node Coding Tables contain the following information:

- Name of the node
- List of NDT to which the node belongs.
- List of corresponding NodeIDs.
- For each field:
  - The Def, In, Out and/or Dyn IDs
  - The min and max values of the field are specified (used for quantization)
  - The quantization scheme to apply
- The animation scheme to apply, for dynamic fields

7.2.6.2 Node Data Type tables

7.2.6.2.1 SF2DNode

SF2DNode		**24 nodes**				**5 bits**			
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
AnimationStream	00000	5	3b	5	3b	6	3b	0	0b
MediaTimeSensor	00001	2	1b	1	0b	1	0b	0	0b
QuantizationParameter	00010	35	6b	0	0b	0	0b	0	0b
Valuator	00011	16	4b	32	5b	16	4b	0	0b
ColorInterpolator	00100	2	1b	3	2b	3	2b	0	0b
ScalarInterpolator	00101	2	1b	3	2b	3	2b	0	0b
Shape	00110	2	1b	2	1b	2	1b	0	0b
TimeSensor	00111	5	3b	5	3b	9	4b	0	0b
TouchSensor	01000	1	0b	1	0b	7	3b	0	0b
Background2D	01001	1	0b	2	1b	2	1b	0	0b
DiscSensor	01010	6	3b	6	3b	9	4b	0	0b
Form	01011	4	2b	1	0b	1	0b	0	0b
Group2D	01100	3	2b	3	2b	1	0b	0	0b
Image2D	01101	1	0b	1	0b	1	0b	0	0b
Inline2D	01110	3	2b	1	0b	1	0b	0	0b
Layout	01111	12	4b	14	4b	12	4b	0	0b
PlaneSensor2D	10000	5	3b	5	3b	6	3b	0	0b
Position2DInterpolator	10001	2	1b	3	2b	3	2b	0	0b
Proximity2DSensor	10010	3	2b	3	2b	8	3b	0	0b
unused	10011								
Switch2D	10100	2	1b	2	1b	2	1b	0	0b
Transform2D	10101	9	4b	9	4b	7	3b	4	2b
VideoObject2D	10110	5	3b	5	3b	7	3b	0	0b
Conditional	10111	1	0b	3	2b	2	1b	0	0b

7.2.6.2.2 SF3DNode

SF3DNode		**31 nodes**				**5 bits**			
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
AnimationStream	00000	5	3b	5	3b	6	3b	0	0b
MediaTimeSensor	00001	2	1b	1	0b	1	0b	0	0b
QuantizationParameter	00010	35	6b	0	0b	0	0b	0	0b
Valuator	00011	16	4b	32	5b	16	4b	0	0b
ColorInterpolator	00100	2	1b	3	2b	3	2b	0	0b

ScalarInterpolator	00101	2	1b	3	2b	3	2b	0	0b
Shape	00110	2	1b	2	1b	2	1b	0	0b
Sound	00111	10	4b	9	4b	9	4b	5	3b
Switch	01000	2	1b	2	1b	2	1b	0	0b
TimeSensor	01001	5	3b	5	3b	9	4b	0	0b
TouchSensor	01010	1	0b	1	0b	7	3b	0	0b
ListeningPoint	01011	4	2b	4	2b	5	3b	2	1b
FBA	01100	2	1b	2	1b	2	1b	0	0b
Background	01101	9	4b	10	4b	10	4b	4	2b
Billboard	01110	4	2b	4	2b	2	1b	0	0b
Collision	01111	5	3b	4	2b	3	2b	0	0b
CoordinateInterpolator	10000	2	1b	3	2b	3	2b	0	0b
DirectionalLight	10001	5	3b	5	3b	5	3b	4	2b
Group	10010	3	2b	3	2b	1	0b	0	0b
Inline	10011	3	2b	1	0b	1	0b	0	0b
LOD	10100	4	2b	1	0b	1	0b	0	0b
NormalInterpolator	10101	2	1b	3	2b	3	2b	0	0b
OrientationInterpolator	10110	2	1b	3	2b	3	2b	0	0b
PointLight	10111	7	3b	7	3b	7	3b	5	3b
PositionInterpolator	11000	2	1b	3	2b	3	2b	0	0b
ProximitySensor	11001	3	2b	3	2b	8	3b	0	0b
SpotLight	11010	10	4b	10	4b	10	4b	4	2b
Transform	11011	8	3b	8	3b	6	3b	5	3b
Viewpoint	11100	5	3b	5	3b	6	3b	3	2b
CompositeMap	11101	2	1b	2	1b	2	1b	0	0b
Conditional	11110	1	0b	3	2b	2	1b	0	0b

7.2.6.2.3 SFAppearanceNode

SFAppearanceNode	**1 node**								
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
Appearance	0	3	2b	3	2b	3	2b	0	0b

7.2.6.2.4 SFAudioNode

SFAudioNode	**6 nodes**								
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
AudioDelay	000	4	2b	2	1b	2	1b	0	0b
AudioMix	001	5	3b	3	2b	3	2b	1	0b
AudioSource	010	6	3b	4	2b	4	2b	0	0b
AudioFX	011	6	3b	4	2b	4	2b	0	0b
AudioSwitch	100	4	2b	2	1b	2	1b	0	0b
AudioClip	101	5	3b	5	3b	7	3b	0	0b

7.2.6.2.5 SFColorNode

SFColorNode	**1 node**								
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
Color	0	1	0b	1	0b	1	0b	1	0b

7.2.6.2.6 SFCoordinate2DNode

SFCoordinate2DNode	**1 node**								
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
Coordinate2D	0	1	0b	1	0b	1	0b	1	0b

7.2.6.2.7 SFCoordinateNode

SFCoordinateNode	1 node		0 bit					
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
Coordinate	0	1	0b	1	0b	1	0b	1	0b

7.2.6.2.8 SFFAPNode

SFFAPNode	1 node		0 bit					
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
FAP	0	70	7b	70	7b	70	7b	0	0b

7.2.6.2.9 SFFBADefNode

SFFBADefNode	2 nodes			1 bit					
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
FBADefMesh	0	4	2b	0	0b	0	0b	0	0b
FBADefTransform	1	3	2b	0	0b	0	0b	0	0b

7.2.6.2.10 SFFBADefTableNode

SFFBADefTableNode	1 node		0 bit					
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
FBADefTable	0	3	2b	1	0b	1	0b	0	0b

7.2.6.2.11 SFFDPNode

SFFDPNode	1 node		0 bit					
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
FDP	0	6	3b	6	3b	6	3b	0	0b

7.2.6.2.12 SFFaceNode

SFFaceNode	1 node		0 bit					
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
Face	0	3	2b	3	2b	4	2b	0	0b

7.2.6.2.13 SFFitNode

SFFitNode	1 node		0 bit					
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
FIT	0	8	3b	8	3b	8	3b	0	0b

7.2.6.2.14 SFFontStyleNode

SFFontStyleNode	1 node		0 bit					
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
FontStyle	0	9	4b	0	0b	0	0b	1	0b

7.2.6.2.15 SFGeometryNode

SFGeometryNode	17 nodes			5 bits					
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
StreamingText	00000	3	2b	3	2b	3	2b	0	0b
Text	00001	4	2b	4	2b	4	2b	1	0b

Circle	00010	1	0b	1	0b	1	0b	1	0b
Curve2D	00011	2	1b	2	1b	2	1b	0	0b
IndexedFaceSet2D	00100	8	3b	6	3b	3	2b	0	0b
IndexedLineSet2D	00101	5	3b	4	2b	2	1b	0	0b
PointSet2D	00110	2	1b	2	1b	2	1b	0	0b
Rectangle	00111	1	0b	1	0b	1	0b	1	0b
Box	01000	1	0b	0	0b	0	0b	1	0b
Cone	01001	4	2b	0	0b	0	0b	2	1b
Cylinder	01010	5	3b	0	0b	0	0b	2	1b
ElevationGrid	01011	13	4b	4	2b	3	2b	1	0b
Extrusion	01100	10	4b	4	2b	0	0b	0	0b
IndexedFaceSet	01101	14	4b	8	3b	4	2b	0	0b
IndexedLineSet	01110	5	3b	4	2b	2	1b	0	0b
PointSet	01111	2	1b	2	1b	2	1b	0	0b
Sphere	10000	1	0b	0	0b	0	0b	1	0b

7.2.6.2.16 SFLayerNode

SFLayerNode	**2 nodes**							**1 bit**	
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
Layer2D	0	5	3b	5	3b	5	3b	0	0b
Layer3D	1	5	3b	9	4b	5	3b	0	0b

7.2.6.2.17 SFLinePropertiesNode

SFLinePropertiesNode	**1 node**							**0 bit**	
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
LineProperties	0	3	2b	3	2b	3	2b	2	1b

7.2.6.2.18 SFMaterialNode

SFMaterialNode	**2 nodes**							**1 bit**	
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
Material2D	0	5	3b	5	3b	5	3b	2	1b
Material	1	6	3b	6	3b	6	3b	6	3b

7.2.6.2.19 SFNormalNode

SFNormalNode	**1 node**							**0 bit**	
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
Normal	0	1	0b	1	0b	1	0b	1	0b

7.2.6.2.20 SFShadowPropertiesNode

SFShadowPropertiesNode	**1 node**							**0 bit**	
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
ShadowProperties	0	2	1b	2	1b	2	1b	1	0b

7.2.6.2.21 SFStreamingNode

SFStreamingNode	**7 nodes**							**3 bits**	
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
AnimationStream	000	5	3b	5	3b	6	3b	0	0b
AudioSource	001	6	3b	4	2b	4	2b	0	0b
AudioClip	010	5	3b	5	3b	7	3b	0	0b
MovieTexture	011	7	3b	5	3b	7	3b	0	0b

Inline2D	100	3	2b	1	0b	1	0b	0	0b
VideoObject2D	101	5	3b	5	3b	7	3b	0	0b
Inline	110	3	2b	1	0b	1	0b	0	0b

7.2.6.2.22 SFTextureCoordinateNode

SFTextureCoordinateNode	**1 node**		**0 bit**					
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
TextureCoordinate	0	1	0b	1	0b	1	0b	0	0b

7.2.6.2.23 SFTextureNode

SFTextureNode	**4 nodes**		**2 bits**					
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
ImageTexture	00	3	2b	1	0b	1	0b	0	0b
MovieTexture	01	7	3b	5	3b	7	3b	0	0b
Composite2DTexture	10	2	1b	2	1b	2	1b	0	0b
Composite3DTexture	11	2	1b	6	3b	2	1b	0	0b

7.2.6.2.24 SFTextureTransformNode

SFTextureTransformNode	**1 node**		**0 bit**					
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
TextureTransform	0	4	2b	4	2b	4	2b	0	0b

7.2.6.2.25 SFTimerNode

SFTimerNode	**1 node**		**0 bit**					
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
TimeSensor	0	5	3b	5	3b	9	4b	0	0b

7.2.6.2.26 SFTopNode

SFTopNode	**4 nodes**		**2 bits**					
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
Group2D	00	3	2b	3	2b	1	0b	0	0b
Group	01	3	2b	3	2b	1	0b	0	0b
Layer2D	10	5	3b	5	3b	5	3b	0	0b
Layer3D	11	5	3b	9	4b	5	3b	0	0b

7.2.6.2.27 SFWorldInfoNode

SFWorldInfoNode	**1 node**		**0 bit**					
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
WorldInfo	0	2	1b	0	0b	0	0b	0	0b

7.2.6.2.28 SFWorldNode

SFWorldNode	**94 nodes**			**7 bits**					
Node name	nodeID	# DEF		# IN		# OUT		# DYN	
AnimationStream	0000000	5	3b	5	3b	6	3b	0	0b
AudioDelay	0000001	4	2b	2	1b	2	1b	0	0b
AudioMix	0000010	5	3b	3	2b	3	2b	1	0b
AudioSource	0000011	6	3b	4	2b	4	2b	0	0b
AudioFX	0000100	6	3b	4	2b	4	2b	0	0b

AudioSwitch	0000101	4	2b	2	1b	2	1b	0	0b
MediaTimeSensor	0000110	2	1b	1	0b	1	0b	0	0b
QuantizationParameter	0000111	35	6b	0	0b	0	0b	0	0b
StreamingText	0001000	3	2b	3	2b	3	2b	0	0b
Valuator	0001001	16	4b	32	5b	16	4b	0	0b
Appearance	0001010	3	2b	3	2b	3	2b	0	0b
AudioClip	0001011	5	3b	5	3b	7	3b	0	0b
Color	0001100	1	0b	1	0b	1	0b	1	0b
ColorInterpolator	0001101	2	1b	3	2b	3	2b	0	0b
FontStyle	0001110	9	4b	0	0b	0	0b	1	0b
ImageTexture	0001111	3	2b	1	0b	1	0b	0	0b
MovieTexture	0010000	7	3b	5	3b	7	3b	0	0b
ScalarInterpolator	0010001	2	1b	3	2b	3	2b	0	0b
Shape	0010010	2	1b	2	1b	2	1b	0	0b
Sound	0010011	10	4b	9	4b	9	4b	5	3b
Switch	0010100	2	1b	2	1b	2	1b	0	0b
Text	0010101	4	2b	4	2b	4	2b	1	0b
TextureCoordinate	0010110	1	0b	1	0b	1	0b	0	0b
TextureTransform	0010111	4	2b	4	2b	4	2b	0	0b
TimeSensor	0011000	5	3b	5	3b	9	4b	0	0b
TouchSensor	0011001	1	0b	1	0b	7	3b	0	0b
WorldInfo	0011010	2	1b	0	0b	0	0b	0	0b
Background2D	0011011	1	0b	2	1b	2	1b	0	0b
Circle	0011100	1	0b	1	0b	1	0b	1	0b
Coordinate2D	0011101	1	0b	1	0b	1	0b	1	0b
Curve2D	0011110	2	1b	2	1b	2	1b	0	0b
DiscSensor	0011111	6	3b	6	3b	9	4b	0	0b
Form	0100000	4	2b	1	0b	1	0b	0	0b
Group2D	0100001	3	2b	3	2b	1	0b	0	0b
Image2D	0100010	1	0b	1	0b	1	0b	0	0b
IndexedFaceSet2D	0100011	8	3b	6	3b	3	2b	0	0b
IndexedLineSet2D	0100100	5	3b	4	2b	2	1b	0	0b
Inline2D	0100101	3	2b	1	0b	1	0b	0	0b
Layout	0100110	12	4b	14	4b	12	4b	0	0b
LineProperties	0100111	3	2b	3	2b	3	2b	2	1b
Material2D	0101000	5	3b	5	3b	5	3b	2	1b
PlaneSensor2D	0101001	5	3b	5	3b	6	3b	0	0b
PointSet2D	0101010	2	1b	2	1b	2	1b	0	0b
Position2DInterpolator	0101011	2	1b	3	2b	3	2b	0	0b
Proximity2DSensor	0101100	3	2b	3	2b	8	3b	0	0b
Rectangle	0101101	1	0b	1	0b	1	0b	1	0b
ShadowProperties	0101110	2	1b	2	1b	2	1b	1	0b
Sound2D	0101111	4	2b	4	2b	4	2b	1	0b
Switch2D	0110000	2	1b	2	1b	2	1b	0	0b
Transform2D	0110001	9	4b	9	4b	7	3b	4	2b
VideoObject2D	0110010	5	3b	5	3b	7	3b	0	0b
ListeningPoint	0110011	4	2b	4	2b	5	3b	2	1b
FBA	0110100	2	1b	2	1b	2	1b	0	0b
Face	0110101	3	2b	3	2b	4	2b	0	0b
FIT	0110110	8	3b	8	3b	8	3b	0	0b
FAP	0110111	70	7b	70	7b	70	7b	0	0b
FDP	0111000	6	3b	6	3b	6	3b	0	0b
FBADefMesh	0111001	4	2b	0	0b	0	0b	0	0b
FBADefTable	0111010	3	2b	1	0b	1	0b	0	0b
FBADefTransform	0111011	3	2b	0	0b	0	0b	0	0b
Background	0111100	9	4b	10	4b	10	4b	4	2b
Billboard	0111101	4	2b	4	2b	2	1b	0	0b
Box	0111110	1	0b	0	0b	0	0b	1	0b
Collision	0111111	5	3b	4	2b	3	2b	0	0b

Cone	1000000	4	2b	0	0b	0	0b	2	1b
Coordinate	1000001	1	0b	1	0b	1	0b	1	0b
CoordinateInterpolator	1000010	2	1b	3	2b	3	2b	0	0b
Cylinder	1000011	5	3b	0	0b	0	0b	2	1b
DirectionalLight	1000100	5	3b	5	3b	5	3b	4	2b
ElevationGrid	1000101	13	4b	4	2b	3	2b	1	0b
Extrusion	1000110	10	4b	4	2b	0	0b	0	0b
Group	1000111	3	2b	3	2b	1	0b	0	0b
IndexedFaceSet	1001000	14	4b	8	3b	4	2b	0	0b
IndexedLineSet	1001001	5	3b	4	2b	2	1b	0	0b
Inline	1001010	3	2b	1	0b	1	0b	0	0b
LOD	1001011	4	2b	1	0b	1	0b	0	0b
Material	1001100	6	3b	6	3b	6	3b	6	3b
Normal	1001101	1	0b	1	0b	1	0b	1	0b
NormalInterpolator	1001110	2	1b	3	2b	3	2b	0	0b
OrientationInterpolator	1001111	2	1b	3	2b	3	2b	0	0b
PointLight	1010000	7	3b	7	3b	7	3b	5	3b
PointSet	1010001	2	1b	2	1b	2	1b	0	0b
PositionInterpolator	1010010	2	1b	3	2b	3	2b	0	0b
ProximitySensor	1010011	3	2b	3	2b	8	3b	0	0b
Sphere	1010100	1	0b	0	0b	0	0b	1	0b
SpotLight	1010101	10	4b	10	4b	10	4b	4	2b
Transform	1010110	8	3b	8	3b	6	3b	5	3b
Viewpoint	1010111	5	3b	5	3b	6	3b	3	2b
Layer2D	1011000	5	3b	5	3b	5	3b	0	0b
Layer3D	1011001	5	3b	9	4b	5	3b	0	0b
Composite2DTexture	1011010	2	1b	2	1b	2	1b	0	0b
Composite3DTexture	1011011	2	1b	6	3b	2	1b	0	0b
CompositeMap	1011100	2	1b	2	1b	2	1b	0	0b
Conditional	1011101	1	0b	3	2b	2	1b	0	0b

λ 7.2.6.3 Node Coding Tables

λ *7.2.6.3.1 Key for Node Coding Tables*

Node Name	**Node Data Type (NDT) list**	**nodeType for each NDT**
Field name	***DEF id IN id OUT id DYN id***	***[min, max] Quantizer id Animation method***

˄*7.2.6.3.2 AnimationStream*

AnimationStream	SFWorldNode	0000000
	SF3DNode	00000
	SF2DNode	00000
	SFStreamingNode	000

Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
loop	000	000	000				
speed	001	001	001		]-?, +?[		
startTime	010	010	010		]-?, +?[		
stopTime	011	011	011		]-?, +?[		
url	100	100	100				
isActive			101				

˄*7.2.6.3.3 AudioDelay*

AudioDelay	SFWorldNode	0000001
	SFAudioNode	000

Field name	*DEF id*	*IN id*	*OUT id*	*DYN id*	*[m, M]*	*Q*	*A*
children	00	0	0				
delay	01	1	1		[0, +?[		
numChan	10				[1, 255]	13 8	
phaseGroup	11				[1, 255]	13 8	

7.2.6.3.4 AudioMix

AudioMix	SFWorldNode SFAudioNode				0000010 001		
Field name	*DEF id*	*IN id*	*OUT id*	*DYN id*	*[m, M]*	*Q*	*A*
children	000	00	00				
numInputs	001	01	01		[1, 255]	13 8	
matrix	010	10	10	0	[0, 1]	0	7
numChan	011				[1, 255]	13 8	
phaseGroup	100				[1, 255]	13 8	

7.2.6.3.5 AudioSource

AudioSource	SFWorldNode SFAudioNode SFStreamingNode				0000011 010 001		
Field name	*DEF id*	*IN id*	*OUT id*	*DYN id*	*[m, M]*	*Q*	*A*
url	000	00	00				
pitch	001	01	01		[0, +?[	13 16	
startTime	010	10	10				
stopTime	011	11	11				
numChan	100				[1, 255]	13 8	
phaseGroup	101				[1, 255]	13 8	

7.2.6.3.6 AudioFX

AudioFX	SFWorldNode SFAudioNode				0000100 011		
Field name	*DEF id*	*IN id*	*OUT id*	*DYN id*	*[m, M]*	*Q*	*A*
children	000	00	00				
orch	001	01	01				
score	010	10	10				
params	011	11	11		]-?, +?[	0	
numChan	100				[1, 255]	13 8	
phaseGroup	101				[1, 255]	13 8	

7.2.6.3.7 AudioSwitch

AudioSwitch	SFWorldNode SFAudioNode				0000101 100		
Field name	*DEF id*	*IN id*	*OUT id*	*DYN id*	*[m, M]*	*Q*	*A*
children	00	0	0				
whichChoice	01	1	1				
numChan	10				[1, 255]	13 8	
phaseGroup	11				[1, 255]	13 8	

7.2.6.3.8 Conditional

Conditional	SFWorldNode SF3DNode SF2DNode			1011101 11110 10111		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
activate		00					
reverseActivate		01					
buffer	0	10	0				
isActive			1				

7.2.6.3.9 MediaTimeSensor

MediaTimeSensor	SFWorldNode SF3DNode SF2DNode			0000110 00001 00001		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
media	0	0	0				
timer	1						

7.2.6.3.10 QuantizationParameter

QuantizationParameter	SFWorldNode SF2DNode SF3DNode			0000111 00010 00010		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
isLocal	000000						
position3DQuant	000001						
position3DMin	000010				]-?, +?[	0	
position3DMax	000011				]-?, +?[	0	
position3DNbBits	000100				[1, 32]	13	5
position2DQuant	000101						
position2DMin	000110				]-?, +?[	0	
position2DMax	000111				]-?, +?[	0	
position2DNbBits	001000				]-?, +?[	13	5
drawOrderQuant	001001						
drawOrderMin	001010				]-?, +?[	0	
drawOrderMax	001011				]-?, +?[	0	
drawOrderNbBits	001100				[1, 32]	13	4
colorQuant	001101						

	1			
colorMin	00111		[0, 1]	0
	0			
colorMax	00111		[0, 1]	0
	1			
colorNbBits	01000		[1, 32]	13 5
	0			
textureCoordinat	01000			
eQuant	1			
textureCoordinat	01001		[0, 1]	0
eMin	0			
textureCoordinat	01001		[0, 1]	0
eMax	1			
textureCoordinat	01010		[1, 32]	13 5
eNbBits	0			
angleQuant	01010			
	1			
angleMin	01011		[0, 2.?]	0
	0			
angleMax	01011		[0, 2.?]	0
	1			
angleNbBits	01100		[1, 32]	13 5
	0			
scaleQuant	01100			
	1			
scaleMin	01101		[0, +?[	0
	0			
scaleMax	01101		[0, +?[	0
	1			
scaleNbBits	01110		[1, 32]	13 5
	0			
keyQuant	01110			
	1			
keyMin	01111		]-?, +?[	0
	0			
keyMax	01111		]-?, +?[	1
	1			
keyNbBits	10000		]-?, +?[	13 5
	0			
normalQuant	10000			
	1			
normalNbBits	10001		[1, 32]	13 5
	0			

7.2.6.3.11 StreamingText

StreamingText	SFWorldNode		0001000		
	SFGeometryNode		00000		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
url	00	00	00				
fontStyle	01	01	01				
ucs_2	10	10	10				

7.2.6.3.12 Valuator

Valuator	SFWorldNode	0001001
	SF3DNode	00011

	SF2DNode				00011		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
inSFBool		00000					
inSFColor		00001					
inMFColor		00010					
inSFFloat		00011					
inMFFloat		00100					
inSFInt32		00101					
inMFInt32		00110					
inSFRotation		00111					
inMFRotation		01000					
inSFString		01001					
inMFString		01010					
inSFTime		01011					
inSFVec2f		01100					
inMFVec2f		01101					
inSFVec3f		01110					
inMFVec3f		01111					
outSFBool	0000	10000	0000				
outSFColor	0001	10001	0001		[0, 1]	4	
outMFColor	0010	10010	0010				
outSFFloat	0011	10011	0011		]-?, +?[		
outMFFloat	0100	10100	0100				
outSFInt32	0101	10101	0101		]-?, +?[		
outMFInt32	0110	10110	0110				
outSFRotation	0111	10111	0111			10	
outMFRotation	1000	11000	1000				
outSFString	1001	11001	1001				
outMFString	1010	11010	1010				
outSFTime	1011	11011	1011		[0, +?[		
outSFVec2f	1100	11100	1100		]-?, +?[	2	
outMFVec2f	1101	11101	1101				
outSFVec3f	1110	11110	1110		]-?, +?[	1	
outMFVec3f	1111	11111	1111				

7.2.6.3.13 Appearance

Appearance	SFWorldNode				0001010		
	SFAppearanceNode				-		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
material	00	00	00				
texture	01	01	01				
textureTransform	10	10	10				

7.2.6.3.14 AudioClip

AudioClip	SFWorldNode				0001011		
	SFAudioNode				101		
	SFStreamingNode				010		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
loop	000	000	000				
pitch	001	001	001		[0, +?[	11	
startTime	010	010	010		]-?, +?[		

stopTime	011	011	011		]-?, +?[
url	100	100	100		
duration_changed			101		
isActive			110		

7.2.6.3.15 Color

Color	SFWorldNode SFColorNode		0001100 -		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
color	0	0	0	0	[0, 1]	4	2

7.2.6.3.16 ColorInterpolator

ColorInterpolator	SFWorldNode SF3DNode SF2DNode		0001101 00100 00100		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_fraction		00			]-?, +?[		
key	0	01	00		[0, 1]	8	
keyValue	1	10	01		[0, 1]	4	
value_changed			10				

7.2.6.3.17 FontStyle

FontStyle	SFWorldNode SFFontStyleNode		0001110 -		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
family	0000						
horizontal	0001						
justify	0010						
language	0011						
leftToRight	0100						
size	0101			0	[0, +?[	11	7
spacing	0110				[0, +?[	11	
style	0111						
topToBottom	1000						

7.2.6.3.18 ImageTexture

ImageTexture	SFWorldNode SFTextureNode		0001111 00		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
url	00	0	0				
repeatS	01						
repeatT	10						

7.2.6.3.19 MovieTexture

MovieTexture	SFWorldNode SFTextureNode	0010000 01

	SFStreamingNode				011		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
loop	000	000	000				
speed	001	001	001		]-?, +?[		
startTime	010	010	010		]-?, +?[		
stopTime	011	011	011		]-?, +?[		
url	100	100	100				
repeatS	101						
repeatT	110						
duration_changed			101				
isActive			110				

7.2.6.3.20 ScalarInterpolator

ScalarInterpolator	SFWorldNode SF3DNode SF2DNode				0010001 00101 00101		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_fraction		00			]-?, +?[		
key	0	01	00		[0, 1]	8	
keyValue	1	10	01		]-?, +?[	0	
value_changed			10				

7.2.6.3.21 Shape

Shape	SFWorldNode SF3DNode SF2DNode				0010010 00110 00110		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
appearance	0	0	0				
geometry	1	1	1				

7.2.6.3.22 Sound

Sound	SFWorldNode SF3DNode				0010011 00111		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
direction	0000	0000	0000		]-?, +?[	9	
intensity	0001	0001	0001		[0, 1]	13 16	
location	0010	0010	0010	000	]-?, +?[	1	0
maxBack	0011	0011	0011	001	[0, +?[	11	7
maxFront	0100	0100	0100	010	[0, +?[	11	7
minBack	0101	0101	0101	011	[0, +?[	11	7
minFront	0110	0110	0110	100	[0, +?[	11	7
priority	0111	0111	0111		[0, 1]	13 16	
source	1000	1000	1000				
spatialize	1001						

7.2.6.3.23 Switch

Switch	SFWorldNode SF3DNode			0010100 01000		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
choice	0	0	0				
whichChoice	1	1	1		[0, 1023]	13 10	

7.2.6.3.24 Text

Text	SFWorldNode SFGeometryNode			0010101 00001		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
string	00	00	00				
length	01	01	01		[0, +?[	11	
fontStyle	10	10	10				
maxExtent	11	11	11	0	[0, +?[	11	7

7.2.6.3.25 TextureCoordinate

TextureCoordinate	SFWorldNode SFTextureCoordinateNode			0010110 -		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
point	0	0	0		]-?, +?[	5	

7.2.6.3.26 TextureTransform

TextureTransform	SFWorldNode SFTextureTransformNode			0010111 -		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
center	00	00	00		]-?, +?[	2	
rotation	01	01	01		[0, 2.?]	6	
scale	10	10	10		]-?, +?[	7	
translation	11	11	11		]-?, +?[	2	

7.2.6.3.27 TimeSensor

TimeSensor	SFWorldNode SFTimerNode SF3DNode SF2DNode			0011000 - 01001 00111		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
cycleInterval	000	000	0000		[0, +?[		
enabled	001	001	0001				
loop	010	010	0010				
startTime	011	011	0011		]-?, +?[		
stopTime	100	100	0100		]-?, +?[		
cycleTime			0101				
fraction_changed			0110				
isActive			0111				

time	1000

7.2.6.3.28 TouchSensor

TouchSensor	SFWorldNode			0011001		
	SF2DNode			01000		
	SF3DNode			01010		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
enabled	0	0	000				
hitNormal_changed			001				
hitPoint_changed			010				
hitTexCoord_changed			011				
isActive			100				
isOver			101				
touchTime			110				

7.2.6.3.29 WorldInfo

WorldInfo	SFWorldNode			0011010		
	SFWorldInfoNode			-		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
info	0						
title	1						

7.2.6.3.30 Background2D

Background2D	SFWorldNode			0011011		
	SF2DNode			01001		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_bind		0					
url	0	1	0				
isBound			1				

7.2.6.3.31 Circle

Circle	SFWorldNode			0011100		
	SFGeometryNode			00010		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
radius	0	0	0	0	[0, +?[	11	7

7.2.6.3.32 Coordinate2D

Coordinate2D	SFWorldNode			0011101		
	SFCoordinate2DNode			-		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
point	0	0	0	0	]-?, +?[	2	1

*7.2.6.3.33 Curve2D

Curve2D	SFWorldNode SFGeometryNode			0011110 00011		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
points	0	0	0				
fineness	1	1	1		]-?, +?[		

*7.2.6.3.34 DiscSensor

DiscSensor	SFWorldNode SF2DNode			0011111 01010		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
autoOffset	000	000	0000				
center	001	001	0001		]-?, +?[	1	
enabled	010	010	0010				
maxAngle	011	011	0011		[-2.?, 2.?]	6	
minAngle	100	100	0100		[-2.?, 2.?]	6	
offset	101	101	0101		]-?, +?[	6	
isActive			0110				
rotation_changed			0111				
trackPoint_changed			1000				

*7.2.6.3.35 Form

Form	SFWorldNode SF2DNode			0100000 01011		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
children	00						
size	01	0	0		[0, +?[	12	
groups	10				[0, 1023]	13 10	
constraint	11				[0, 255]	13 8	

*7.2.6.3.36 Group2D

Group2D	SFWorldNode SFTopNode SF2DNode			0100001 00 01100		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren		00					
removeChildren		01					
children	00	10	0				
bboxCenter	01				]-?, +?[	2	
bboxSize	10				]-?, +?[	12	

*7.2.6.3.37 Image2D

Image2D	SFWorldNode SF2DNode			0100010 01101		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***

url	0	0	0

7.2.6.3.38 IndexedFaceSet2D

IndexedFaceSet2D	SFWorldNode SFGeometryNode		0100011 00100			
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_colorIndex		000					
set_coordIndex		001					
set_texCoordIndex		010					
color	000	011	00				
coord	001	100	01				
texCoord	010	101	10				
colorIndex	011				[-1, +?[	0	
colorPerVertex	100						
convex	101						
coordIndex	110				[-1, +?[	0	
texCoordIndex	111				[-1, +?[	0	

7.2.6.3.39 IndexedLineSet2D

IndexedLineSet2D	SFWorldNode SFGeometryNode		0100100 00101			
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_colorIndex		00					
set_coordIndex		01					
color	000	10	0				
coord	001	11	1				
colorIndex	010				[-1, +?[	0	
colorPerVertex	011						
coordIndex	100				[-1, +?[	0	

7.2.6.3.40 Inline2D

Inline2D	SFWorldNode SF2DNode SFStreamingNode		0100101 01110 100			
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
url	00	0	0				
bboxCenter	01				]-?, +?[	2	
bboxSize	10				]-?, +?[	12	

7.2.6.3.41 Layout

Layout	SFWorldNode SF2DNode		0100110 01111			
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
children	0000	0000	0000				
wrap	0001	0001	0001				
size	0010	0010	0010		[0, +?[	12	
horizontal	0011	0011	0011				

justify	0100	0100	0100			
leftToRight	0101	0101	0101			
topToBottom	0110	0110	0110			
spacing	0111	0111	0111		[0, +?[	0
smoothScroll	1000	1000	1000			
loop	1001	1001	1001			
scrollVertical	1010	1010	1010			
scrollRate	1011	1011	1011		]-?, +?[	0
addChildren		1100				
removeChildren		1101				

7.2.6.3.42 LineProperties

LineProperties	SFWorldNode SFLinePropertiesNode			0100111 -		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
lineColor	00	00	00	0	[0, 1]	4	2
lineStyle	01	01	01		[0, 5]	13 5	
width	10	10	10	1	[0, +?[	12	7

7.2.6.3.43 Material2D

Material2D	SFWorldNode SFMaterialNode			0101000 0		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
diffuseColor	000	000	000	0	[0, 1]	4	2
filled	001	001	001				
lineProps	010	010	010				
shadowsProps	011	011	011				
transparency	100	100	100	1	[0, 1]	4	2

7.2.6.3.44 PlaneSensor2D

PlaneSensor2D	SFWorldNode SF2DNode			0101001 10000		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
autoOffset	000	000	000				
enabled	001	001	001				
maxPosition	010	010	010		]-?, +?[	2	
minPosition	011	011	011		]-?, +?[	2	
offset	100	100	100		]-?, +?[	2	
trackPoint_changed			101				

7.2.6.3.45 PointSet2D

PointSet2D	SFWorldNode SFGeometryNode			0101010 00110		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
color	0	0	0				
coord	1	1	1				

7.2.6.3.46 Position2DInterpolator

Position2DInterpolator	SFWorldNode SF2DNode				0101011 10001		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_fraction		00					
key	0	01	00		[0, 1]	8	
keyValue	1	10	01		]-?, +?[	2	
value_changed			10				

7.2.6.3.47 Proximity2DSensor

Proximity2DSensor	SFWorldNode SF2DNode				0101100 10010		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
center	00	00	000		[-1, +?[	2	
size	01	01	001		[0, +?[	11	
enabled	10	10	010				
isActive			011				
position_changed			100				
orientation_changed			101				
enterTime			110				
exitTime			111				

7.2.6.3.48 Rectangle

Rectangle	SFWorldNode SFGeometryNode				0101101 00111		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
size	0	0	0	0	[0, +?[	12	7

*7.2.6.3.49 ShadowProperties

ShadowProperties	SFWorldNode SFShadowPropertiesNode				0101110 -		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
shadowPos	0	0	0		[-15, 16]	13 5	
shadowColor	1	1	1	0	[0, 1]	4	2

*7.2.6.3.50 Switch2D

Switch2D	SFWorldNode SF2DNode				0110000 10100		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
choice	0	0	0				
whichChoice	1	1	1		[0, 1023]	13 10	

*7.2.6.3.51 Transform2D

Transform2D	SFWorldNode	0110001

	SF2DNode			10101			
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren		0000					
removeChildren		0001					
center	0000	0010	000		]-?, +?[	2	
children	0001	0011	001				
rotationAngle	0010	0100	010	00	[0, 2.?]	6	3
scale	0011	0101	011	01	[0, +?[	7	5
scaleOrientation	0100	0110	100	10	[0, 2.?]	6	3
drawingOrder	0101	0111	101		[0, +?[	3	
translation	0110	1000	110	11	]-?, +?[	2	1
bboxCenter	0111				]-?, +?[	2	
bboxSize	1000				]-?, +?[	12	

7.2.6.3.52 VideoObject2D

VideoObject2D	SFWorldNode			0110010			
	SF2DNode			10110			
	SFStreamingNode			101			
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
loop	000	000	000				
speed	001	001	001		[0, +?[	0	
startTime	010	010	010				
stopTime	011	011	011				
url	100	100	100				
duration_changed			101				
isActive			110				

7.2.6.3.53 ListeningPoint

ListeningPoint	SFWorldNode			0110011			
	SF3DNode			01011			
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_bind		00					
jump	00	01	000				
orientation	01	10	001	0		10	6
position	10	11	010	1	]-?, +?[	1	0
description	11						
bindTime			011				
isBound			100				

7.2.6.3.54 FBA

FBA	SFWorldNode			0110100			
	SF3DNode			01100			
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
face	0	0	0				
body	1	1	1				

7.2.6.3.55 Face

Face	SFWorldNode SFFaceNode			0110101 -		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
fit	00	00	00				
fdp	01	01	01				
fap	10	10	10				
renderedFace			11				

7.2.6.3.56 FIT

FIT	SFWorldNode SFFitNode			0110110 -		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
FAPs	000	000	000		[-1, 68]		
Graph	001	001	001		[0, 68]		
numeratorExp	010	010	010				
denominatorExp	011	011	011				
numeratorTerms	100	100	100		[0, 10]		
denominatorTerms	101	101	101		[0, 10]		
numeratorCoefs	110	110	110		]-?, +?[		
denominatorCoefs	111	111	111		]-?, +?[		

7.2.6.3.57 FAP

FAP	SFWorldNode SFFAPNode			0110111 -		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
visemeSelect	00000 00	00000 00	00000 00		[0, 15]	13 4	
expressionSelect	00000 01	00000 01	00000 01		[0, 5]	13 3	
viseme	00000 10	00000 10	00000 10			0	
expression	00000 11	00000 11	00000 11		[0, 5]	0	
open_jaw	00001 00	00001 00	00001 00		[0, +?[	0	
lower_t_midlip	00001 01	00001 01	00001 01		]-?, +?[	0	
raise_b_midlip	00001 10	00001 10	00001 10		]-?, +?[	0	
stretch_l_corner	00001 11	00001 11	00001 11		]-?, +?[	0	
stretch_r_corner	00010 00	00010 00	00010 00		]-?, +?[	0	
lower_t_lip_lm	00010 01	00010 01	00010 01		]-?, +?[	0	
lower_t_lip_rm	00010 10	00010 10	00010 10		]-?, +?[	0	
lower_b_lip_lm	00010 11	00010 11	00010 11		]-?, +?[	0	

lower_t_lip_rm	00011 00	00011 00	00011 00	]-?, +?[	0
raise_l_cornerlip	00011 01	00011 01	00011 01	]-?, +?[	0
raise_r_cornerlip	00011 10	00011 10	00011 10	]-?, +?[	0
thrust_jaw	00011 11	00011 11	00011 11	[0, +?[	0
shift_jaw	00100 00	00100 00	00100 00	]-?, +?[	0
push_b_lip	00100 01	00100 01	00100 01	]-?, +?[	0
push_t_lip	00100 10	00100 10	00100 10	]-?, +?[	0
depress_chin	00100 11	00100 11	00100 11	[0, +?[	0
close_t_l_eyelid	00101 00	00101 00	00101 00	]-?, +?[	0
close_t_r_eyelid	00101 01	00101 01	00101 01	]-?, +?[	0
close_b_l_eyelid	00101 10	00101 10	00101 10	]-?, +?[	0
close_b_r_eyelid	00101 11	00101 11	00101 11	]-?, +?[	0
yaw_l_eyeball	00110 00	00110 00	00110 00	]-?, +?[	0
yaw_r_eyeball	00110 01	00110 01	00110 01	]-?, +?[	0
pitch_l_eyeball	00110 10	00110 10	00110 10	]-?, +?[	0
pitch_r_eyeball	00110 11	00110 11	00110 11	]-?, +?[	0
thrust_l_eyeball	00111 00	00111 00	00111 00	]-?, +?[	0
thrust_r_eyeball	00111 01	00111 01	00111 01	]-?, +?[	0
dilate_l_pupil	00111 10	00111 10	00111 10	[0, +?[	0
dilate_r_pupil	00111 11	00111 11	00111 11	[0, +?[	0
raise_l_i_eyebrow	01000 00	01000 00	01000 00	]-?, +?[	0
raise_r_i_eyebrow	01000 01	01000 01	01000 01	]-?, +?[	0
raise_l_m_eyebrow	01000 10	01000 10	01000 10	]-?, +?[	0
raise_r_m_eyebrow	01000 11	01000 11	01000 11	]-?, +?[	0
raise_l_o_eyebrow	01001 00	01001 00	01001 00	]-?, +?[	0
raise_r_o_eyebrow	01001 01	01001 01	01001 01	]-?, +?[	0
squeeze_l_eyebrow	01001 10	01001 10	01001 10	]-?, +?[	0
squeeze_r_eyebrow	01001 11	01001 11	01001 11	]-?, +?[	0
puff_l_cheek	01010 00	01010 00	01010 00	]-?, +?[	0
puff_r_cheek	01010	01010	01010	]-?, +?[	0

lift_l_cheek	0101001	0101001	0101001	[0, +?[	0
lift_r_cheek	0101010	0101010	0101010	[0, +?[	0
shift_tongue_tip	0101011	0101011	0101011	]-?, +?[	0
raise_tongue_tip	0101011	0101011	0101011	]-?, +?[	0
thrust_tongue_tip	0101011	0101011	0101011	]-?, +?[	0
raise_tongue	0101011	0101011	0101011	]-?, +?[	0
tongue_roll	0110000	0110000	0110000	[0, +?[	0
head_pitch	0110001	0110001	0110001	]-?, +?[	0
head_yaw	0110010	0110010	0110010	]-?, +?[	0
head_roll	0110011	0110011	0110011	]-?, +?[	0
lower_t_midlip	0110100	0110100	0110100	]-?, +?[	0
raise_b_midlip_o	0110101	0110101	0110101	]-?, +?[	0
stretch_l_cornerlip	0110110	0110110	0110110	]-?, +?[	0
stretch_r_cornerlip_o	0110111	0110111	0110111	]-?, +?[	0
lower_t_lip_lm_o	0111000	0111000	0111000	]-?, +?[	0
lower_t_lip_rm_o	0111001	0111001	0111001	]-?, +?[	0
raise_b_lip_lm_o	0111010	0111010	0111010	]-?, +?[	0
raise_b_lip_rm_o	0111011	0111011	0111011	]-?, +?[	0
raise_l_cornerlip_o	0111100	0111100	0111100	]-?, +?[	0
raise_r_cornerlip_o	0111101	0111101	0111101	]-?, +?[	0
stretch_l_nose	0111110	0111110	0111110	]-?, +?[	0
stretch_r_nose	0111111	0111111	0111111	]-?, +?[	0
raise_nose	1000000	1000000	1000000	]-?, +?[	0
bend_nose	1000001	1000001	1000001	]-?, +?[	0
raise_l_ear	1000010	1000010	1000010	]-?, +?[	0
raise_r_ear	1000011	1000011	1000011	]-?, +?[	0
pull_l_ear	1000100	1000100	1000100	]-?, +?[	0
pull_r_ear	1000101	1000101	1000101	]-?, +?[	0

7.2.6.3.58 FDP

FDP	SFWorldNode SFFDPNode			0111000 -		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
featurePointsCoord	000	000	000				
textureCoord4FeaturePoints	001	001	001				
calibrationMesh	010	010	010				
faceTexture	011	011	011				
FBADefTables	100	100	100				
faceSceneGraph	101	101	101				

7.2.6.3.59 FBADefMesh

FBADefMesh	SFWorldNode SFFBADefNode			0111001 0		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
faceSceneGraphNode	00						
intervalBorders	01					0	
coordIndex	10					0	
displacements	11					0	

7.2.6.3.60 FBADefTable

FBADefTable	SFWorldNode SFFBADefTableNode			0111010 -		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
fapID	00				[1, 68]	13 7	
highLevelSelect	01				[1, 64]	13 6	
tables	10	0	0				

7.2.6.3.61 FBADefTransform

FBADefTransform	SFWorldNode SFFBADefNode			0111011 1		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
faceSceneGraphNode	00						
fieldId	01						
fieldValue	10						

7.2.6.3.62 Background

Background	SFWorldNode SF3DNode			0111100 01101		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_bind		0000					
groundAngle	0000	0001	0000	00	[0, ?/2]	6	3
groundColor	0001	0010	0001	01	[0, 1]	4	2

backURL	0010	0011	0010				
frontURL	0011	0100	0011				
leftURL	0100	0101	0100				
rightURL	0101	0110	0101				
topURL	0110	0111	0110				
skyAngle	0111	1000	0111	10	[0, 2.?]	6	3
skyColor	1000	1001	1000	11	[0, 1]	4	2
isBound			1001				

7.2.6.3.63 Billboard

Billboard	SFWorldNode SF3DNode			0111101 01110			
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren		00					
removeChildren		01					
axisOfRotation	00	10	0			9	
children	01	11	1				
bboxCenter	10				]-?, +?[	1	
bboxSize	11				[0, +?[	11	

7.2.6.3.64 Box

Box	SFWorldNode SFGeometryNode			0111110 01000			
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
size	0			0	[0, +?[	11	7

7.2.6.3.65 Collision

Collision	SFWorldNode SF3DNode			0111111 01111			
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren		00					
removeChildren		01					
children	000	10	00				
collide	001	11	01				
bboxCenter	010				]-?, +?[	1	
bboxSize	011				[0, +?[	11	
proxy	100						
collideTime			10				

7.2.6.3.66 Cone

Cone	SFWorldNode SFGeometryNode			1000000 01001			
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
bottomRadius	00			0	[0, +?[	11	7
height	01			1	[0, +?[	11	7
side	10						
bottom	11						

7.2.6.3.67 Coordinate

Coordinate	SFWorldNode SFCoordinateNode			1000001 -		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
point	0	0	0	0	]-?, +?[	1	0

7.2.6.3.68 CoordinateInterpolator

CoordinateInterpolator	SFWorldNode SF3DNode			1000010 10000		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_fraction		00					
key	0	01	00		[0, 1]	8	
keyValue	1	10	01		]-?, +?[	1	
value_changed			10				

7.2.6.3.69 Cylinder

Cylinder	SFWorldNode SFGeometryNode			1000011 01010		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
bottom	000						
height	001			0	[0, +?[	11	7
radius	010			1	[0, +?[	11	7
side	011						
top	100						

7.2.6.3.70 DirectionalLight

DirectionalLight	SFWorldNode SF3DNode			1000100 10001		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
ambientIntensity	000	000	000	00	[0, 1]	4	2
color	001	001	001	01	[0, 1]	4	2
direction	010	010	010	10		9	6
intensity	011	011	011	11	[0, 1]	4	2
on	100	100	100				

7.2.6.3.71 ElevationGrid

ElevationGrid	SFWorldNode SFGeometryNode			1000101 01011		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_height		00					
color	0000	01	00				
normal	0001	10	01				
texCoord	0010	11	10				
height	0011			0	]-?, +?[	11	11
ccw	0100						
colorPerVertex	0101						
creaseAngle	0110				[0, 2.?]	6	

normalPerVertex	0111					
solid	1000					
xDimension	1001			[0, +?[	13 16	
xSpacing	1010			[0, +?[	0	
zDimension	1011			[0, +?[	13 16	
zSpacing	1100			[0, +?[	0	

7.2.6.3.72 Extrusion

Extrusion	SFWorldNode SFGeometryNode			1000110 01100		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_crossSection		00					
set_orientation		01					
set_scale		10					
set_spine		11					
beginCap	0000						
ccw	0001						
convex	0010						
creaseAngle	0011				[0, 2.?]	6	
crossSection	0100				]-?, +?[		
endCap	0101						
orientation	0110				]-?, +?[	10	
scale	0111				[0, +?[	7	
solid	1000						
spine	1001				]-?, +?[	13 8	

7.2.6.3.73 Group

Group	SFWorldNode SFTopNode SF3DNode			1000111 01 10010		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren		00					
removeChildren		01					
children	00	10	0				
bboxCenter	01				]-?, +?[	1	
bboxSize	10				[0, +?[	11	

7.2.6.3.74 IndexedFaceSet

IndexedFaceSet	SFWorldNode SFGeometryNode			1001000 01101		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_colorIndex		000					
set_coordIndex		001					
set_normalIndex		010					
set_texCoordIndex		011					
color	0000	100	00				
coord	0001	101	01				
normal	0010	110	10				

texCoord	0011	111	11		
ccw	0100				
colorIndex	0101			[-1, +?[	
colorPerVertex	0110				
convex	0111				
coordIndex	1000			[-1, +?[	0
creaseAngle	1001			[0, 2.?]	6
normalIndex	1010			[-1, +?[	0
normalPerVertex	1011				
solid	1100				
texCoordIndex	1101			[-1, +?[	0

7.2.6.3.75 IndexedLineSet

IndexedLineSet	SFWorldNode SFGeometryNode			1001001 01110		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_colorIndex		00					
set_coordIndex		01					
color	000	10	0				
coord	001	11	1				
colorIndex	010				[-1, +?[	0	
colorPerVertex	011						
coordIndex	100				[-1, +?[	0	

7.2.6.3.76 Inline

Inline	SFWorldNode SF3DNode SFStreamingNode			1001010 10011 110		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
url	00	0	0				
bboxCenter	01				]-?, +?[	1	
bboxSize	10				[0, +?[	11	

7.2.6.3.77 LOD

LOD	SFWorldNode SF3DNode			1001011 10100		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
level	00	0	0				
center	01				]-?, +?[	1	
range	10				[0, +?[	11	
fpsRange	11				[0, 127]	13 32	

7.2.6.3.78 Material

Material	SFWorldNode SFMaterialNode			1001100 1		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
ambientIntensity	000	000	000	000	[0, 1]	4	2

diffuseColor	001	001	001	001	[0, 1]	4	2
emissiveColor	010	010	010	010	[0, 1]	4	2
shininess	011	011	011	011	[0, 1]	4	2
specularColor	100	100	100	100	[0, 1]	4	2
transparency	101	101	101	101	[0, 1]	4	2

7.2.6.3.79 Normal

Normal	SFWorldNode SFNormalNode			1001101 -		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
vector	0	0	0	0		4	9

7.2.6.3.80 NormalInterpolator

NormalInterpolator	SFWorldNode SF3DNode			1001110 10101		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_fraction		00					
key	0	01	00		[0, 1]	8	
keyValue	1	10	01		]-?, +?[	9	
value_changed			10				

7.2.6.3.81 OrientationInterpolator

OrientationInterpolator	SFWorldNode SF3DNode			1001111 10110		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_fraction		00					
key	0	01	00		[0, 1]	8	
keyValue	1	10	01		]-?, +?[	10	
value_changed			10				

7.2.6.3.82 PointLight

PointLight	SFWorldNode SF3DNode			1010000 10111		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
ambientIntensity	000	000	000	000	[0, 1]	4	2
attenuation	001	001	001		[0, +?[	13 16	
color	010	010	010	001	[0, 1]	4	2
intensity	011	011	011	010	[0, 1]	4	2
location	100	100	100	011	]-?, +?[	1	0
on	101	101	101				
radius	110	110	110	100	[0, +?[	11	7

7.2.6.3.83 PointSet

PointSet	SFWorldNode SFGeometryNode			1010001 01111		
Field name	***DEF***	***IN id***	***OUT***	***DYN***	***[m, M]***	***Q***	***A***

	id		id	id			
color	0	0	0				
coord	1	1	1				

7.2.6.3.84 PositionInterpolator

PositionInterpolator	SFWorldNode SF3DNode			1010010 11000		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_fraction		00					
key	0	01	00		[0, 1]	8	
keyValue	1	10	01		]-?, +?[	1	
value_changed			10				

7.2.6.3.85 ProximitySensor

ProximitySensor	SFWorldNode SF3DNode			1010011 11001		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
center	00	00	000		]-?, +?[	1	
size	01	01	001		[0, +?[	11	
enabled	10	10	010				
isActive			011				
position_changed			100				
orientation_changed			101				
enterTime			110				
exitTime			111				

7.2.6.3.86 Sphere

Sphere	SFWorldNode SFGeometryNode			1010100 10000		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
radius	0			0	[0, +?[	11	7

7.2.6.3.87 SpotLight

SpotLight	SFWorldNode SF3DNode			1010101 11010		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
ambientIntensity	0000	0000	0000	00	[0, 1]	4	2
attenuation	0001	0001	0001	01	[0, +?[	11	2
beamWidth	0010	0010	0010		[0, ?/2]	6	
color	0011	0011	0011	10	[0, 1]	4	2
cutOffAngle	0100	0100	0100		[0, ?/2]	6	
direction	0101	0101	0101		]-?, +?[	9	
intensity	0110	0110	0110		[0, 1]	4	
location	0111	0111	0111	11	]-?, +?[	1	0
on	1000	1000	1000				
radius	1001	1001	1001		[0, +?[	11	

7.2.6.3.88 Transform

Transform	SFWorldNode SF3DNode			1010110 11011		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
addChildren		000					
removeChildren		001					
center	000	010	000	000	]-?, +?[	1	0
children	001	011	001				
rotation	010	100	010	001		10	6
scale	011	101	011	010	[0, +?[	7	5
scaleOrientation	100	110	100	011		10	6
translation	101	111	101	100	]-?, +?[	1	0
bboxCenter	110				]-?, +?[	1	
bboxSize	111				[0, +?[	11	

7.2.6.3.89 Viewpoint

Viewpoint	SFWorldNode SF3DNode			1010111 11100		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
set_bind		000					
fieldOfView	000	001	000	00	[0, 3.14159]	6	3
jump	001	010	001				
orientation	010	011	010	01		10	6
position	011	100	011	10	]-?, +?[	1	0
description	100						
bindTime			100				
isBound			101				

7.2.6.3.90 Layer2D

Layer2D	SFWorldNode SFTopNode SFLayerNode			1011000 10 0		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
children	000	000	000				
childrenLayer	001	001	001				
size	010	010	010		]-?, +?[	2	
translation	011	011	011		]-?, +?[	2	
depth	100	100	100		]-?, +?[	3	

7.2.6.3.91 Layer3D

Layer3D	SFWorldNode SFTopNode SFLayerNode			1011001 11 1		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
children	000	0000	000				
childrenLayer	001	0001	001				
translation	010	0010	010		]-?, +?[	2	
depth	011	0011	011		]-?, +?[	3	

size	100	0100	100		]-?, +?[	2
background		0101				
fog		0110				
navigationInfo		0111				
viewpoint		1000				

7.2.6.3.92 Composite2DTexture

Composite2DTexture	SFWorldNode SFTextureNode			1011010 10		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
children	0	0	0			
size	1	1	1		]-?, +?[	2	

7.2.6.3.93 Composite3DTexture

Composite3DTexture	SFWorldNode SFTextureNode			1011011 11		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
children	0	000	0			
size	1	001	1		]-?, +?[	2	
background		010					
fog		011					
navigationInfo		100					
Viewpoint		101					

7.2.6.3.94 CompositeMap

CompositeMap	SFWorldNode SF3DNode			1011100 11101		
Field name	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
children2D	0	0	0			
sceneSize	1	1	1		]-?, +?[	2	

7.3 Identification and Association of Elementary Streams

7.3.1 Introduction

Elementary Streams are one major conceptual element of the MPEG-4 architecture. The Scene Description as well as any media data stream is conveyed within one or more Elementary Streams. In order to access the content of Elementary Streams, they must be properly identified and associated to each other.

The semantic *association* of different Elementary Streams is accomplished by the Scene Description that declares the spatio-temporal relation of various media objects. Those media objects that necessitate streaming media data then point to Elementary Streams by means of a numeric identifier, an *objectDescriptorID*.

Identification of Elementary Streams includes information about the source of the conveyed media data, in form of an URL or a numeric identifier, as well as the encoding format, the configuration for the Access Unit Layer packetization of the Elementary Stream (see Subclause) and intellectual property information. Optionally more information can be associated to a media object, most notably Object Content Information.

This identification of Elementary Streams is provided by means of Object Descriptors that are labeled by the said objectDescriptorIDs. This allows one to resolve the association conveyed by the scene description to the actual Elementary Streams. The syntax and semantics of Object Descriptors, the method to convey them, as well as their usage are specified in this subclause as detailed below:

- Subclause specifies how Object Descriptors are conveyed between peer MPEG-4 terminals in the form of Object Descriptor Streams.
- Subclause specifies the syntax and semantics of the Object Descriptor and its sub-descriptors.
- Subclause specifies how Object Descriptors are used in an MPEG-4 session. This includes the scoping of objectDescriptorIDs in MPEG-4 sessions and the usage of URLs. A walk through a session set up is provided as well.

7.3.2 Object Descriptor Elementary Stream

Object Descriptors are transported in a dedicated Elementary Stream with streamType = ObjectDescriptorStream. An Object Descriptor Stream shall always be associated to another stream with streamType = SceneDescriptionStream within the same Object Descriptor that declares the scene description stream. The method to associate the two streams is specified in Subclause .

The Object Descriptor stream serves to convey and update Object Descriptors. Complete Object Descriptors and their primary element, the ES_descriptors, may be conveyed or updated anytime during the course of an MPEG-4 session, given the semantic restrictions detailed below. The update mechanism is specified to be able to advertise new Elementary Streams for a Media Object as they become available or to remove the reference to streams that are no longer available. These updates are time stamped to indicate the instant in time they take effect.

Note: Updates need not be co-located in time with the addition or removal of media objects in the scene description that refer to such an Object Descriptor.

This subclause specifies the structure of the Object Descriptor Elementary Stream as well as the syntax and semantics of the Object Descriptor Updates.

λ 7.3.2.1 Structure of the Object Descriptor Elementary Stream

Object Descriptors are always conveyed by Object Descriptor Updates (OD-Update) as specified in the subsequent subclauses. Each OD-Update groups information for one or more Object Descriptors or ES_descriptors.

OD-Updates are packaged in Access Units. An Access Unit groups one or more OD-Updates that shall become valid at a specific instant in time. More than one subsequent Access Unit may refer to the same instant in time.

Access Units in Object Descriptor Elementary Streams shall be labeled and time stamped by suitable means. In the context of this Committee Draft of International Standard, this shall be done by means of the related flags and the decoding time stamp, respectively, in the AL_PDU Header (see Subclause).

λ 7.3.2.2 OD-Update Syntax and Semantics

The syntax to convey, update and remove ObjectDescriptor and ES_descriptor items is specified in this subclause.

˚7.3.2.2.1 *ObjectDescriptorUpdate*

The ObjectDescriptorUpdate class is used to transmit a list of ObjectDescriptors. If the content of an existing ObjectDescriptor is changed by an ObjectDescriptorUpdate, this implies the invalidation of the Elementary Stream(s) and the decoders that had been described by the old ObjectDescriptor and the immediate set up of new decoder(s) and attachment to the Elementary Stream(s) as described in the updated ObjectDescriptor.

l7.3.2.2.1.1 Syntax

```
aligned(8) class ObjectDescriptorUpdate : uint(8) ObjectDescriptorUpdateTag {
      int j;
      uint(8) objectCount;
      for(j=0;j<objectCount; j++) {
            ObjectDescriptor OD[j];
      }
}
```

l7.3.2.2.1.2 Semantics

objectCount - indicates the number of ObjectDescriptors to follow

OD[j] - an ObjectDescriptor as defined in Subclause .

˚7.3.2.2.2 *ObjectDescriptorRemove*

The ObjectDescriptorRemove class is used to invalidate an ObjectDescriptor. This implies that the reference to all Elementary Streams that have been listed in this ObjectDescriptor is removed.

l7.3.2.2.2.1 Syntax

```
aligned(8) class ObjectDescriptorRemove : uint(8) ObjectDescriptorRemoveTag {
      int j;
      uint(6) OD_Count;
      for(j=0;j<OD_Count; j++) {
            uint(10) objectDescriptorId[j];
      }
}
```

l7.3.2.2.2.2 Semantics

OD_Count - indicates the number of ObjectDescriptors that are removed.

objectDescriptorId - indicates the ID of an ObjectDescriptor that is to be removed.

7.3.2.2.3 ES_DescriptorUpdate

The ES_DescriptorUpdate class is used to add references to new Elementary Streams in an ObjectDescriptor. If the content of an existing ES_Descriptor is changed by an ES_DescriptorUpdate, this implies the invalidation of the Elementary Stream and the decoder that had been described by the old ES_Descriptor and the immediate set up of a new decoder and attachment to the Elementary Stream as described in the updated ES_Descriptor.

l7.3.2.2.3.1 Syntax

```
aligned(8) class ES_DescriptorUpdate : uint(8) ES_DescriptorUpdateTag {
      int j;
      uint(10) objectDescriptorId;
      uint(5) streamCount;
      uint(1) reserved;
      for(j=0;j<streamCount; j++) {
            ES_Descriptor ESD[j];
      }
}
```

l7.3.2.2.3.2 Semantics

objectDescriptorID - identifies the ObjectDescriptor for which ES_Descriptors are updated.

streamCount - indicates how many ES_Descriptors will follow.

ESD[j] - an ES_descriptor as defined in Subclause .

7.3.2.2.4 ES_DescriptorRemove

The ES_descriptorRemove class is used to remove the reference to an Elementary Stream from an ObjectDescriptor.

l7.3.2.2.4.1 Syntax

```
aligned(8) class ES_DescriptorRemove : uint(8) ES_DescriptorRemoveTag
{
      int j;
      uint(10) objectDescriptorId;
      uint(5) streamCount;
      uint(1) reserved;
      for(j=0;j<streamCount; j++) {
            uint(5) ES_Number[j];
            uint(3) reserved;
      }
}
```

l7.3.2.2.4.2 Semantics

objectDescriptorID - identifies the ObjectDescriptor from which ES_Descriptors are removed.

streamCount - indicates the number of ES_Descriptors to be removed.

ES_number[j] - indicates the label of an ES_Descriptor to be removed from objectDescriptorID.

λ 7.3.2.3 Descriptor tags

Tags are defined to identify the descriptor update and remove messages defined in the previous subclause. They are shown in Table 7-3.

Table 7-3: List of Descriptor Tags

0	reserved
1	ObjectDescriptorUpdateTag
2	ObjectDescriptorRemoveTag
3	ES_DescriptorUpdateTag
4	ES_DescriptorRemoveTag
0x05-0xFF	reserved for ISO use

l7.3.3 Object Descriptor Syntax and Semantics

The Object Descriptor structure complements the information contained in the scene description for such media objects in the scene hierarchy that are associated to streaming content in the form of Elementary Streams. In other words, Elementary Streams are only accessible in a session if they are described by an Object Descriptor.

In this context it may be noted that the scene description itself is also conveyed as an Elementary Stream and therefore has an Object Descriptor that describes this stream.

Each media object that refers to streaming data is associated to at most one Object Descriptor. The Object Descriptor constitutes a grouping mechanism that includes one or more ES_Descriptors, each providing details on one Elementary Stream that is associated to this media object. This grouping mechanism allows to specify an ordered list of interdependent streams as they are needed e. g. for a scaleable coded content representation. Furthermore, multiple alternative streams that convey the same content, e. g., in multiple qualities, can be grouped together. Finally, streams with additional information, such as Object Content Information, may be associated to the media object.

The ES_Descriptor provides an unambiguous identifier of the Elementary Stream and contains those parameters that are needed at the time the decoding of an Elementary Stream is initiated. These parameters include the decoder configuration, the configuration of the AL-PDU Header, Intellectual Property Information (IPI) and Quality of Service (QoS) requirements .

The syntax and semantics of the ObjectDescriptor class and of all its components are specified in the subsequent subclauses.

λ 7.3.3.1 ObjectDescriptor

The ObjectDescriptor consists of three different parts:

The first part is an objectDescriptorId which uniquely identifies a media object within a session. The objectDescriptorId is used for example in the scene description to associate media objects in the scene hierarchy with their respective ObjectDescriptor carrying the details of one or more Elementary Streams that are associated to those media objects.

The second part consists of a list of ES_Descriptors, each providing parameters for a single Elementary Stream.

The third part is an optional extension mechanism, whose semantic supports the inclusion of future extensions in a backward compatible way, as well as the transport of private data.

7.3.3.1.1 Syntax

```
class ObjectDescriptor () {
      uint(10) ObjectDescriptorID;
      uint(5) streamCount;
```

```
      uint(1) extensionFlag;
      for(j=0; j<streamCount; j++) {
            ES_Descriptor ();
      }
      if (extensionFlag) {
            uint(8) descriptorLength;
            for (j=0;j<descriptorLength;) {
                  extensionDescriptor ();
                  j = j + lengthof(extensionDescriptor)
            }
      }
}
```

7.3.3.1.2 Semantics

objectDescriptorId **–** This syntax element identifies the ObjectDescriptor. The value objectDescriptorId=0 is forbidden and the value objectDescriptorId=1023 is reserved. objectDescriptorIds are unique within an MPEG-4 session.

streamCount **–** This syntax element indicates the number of ES_Descriptor structures included in the ObjectDescriptor. A streamCount of zero indicates an empty ObjectDescriptor for which no Elementary Streams are available yet. The streamCount of 31 is reserved for future extensions.

extensionFlag – This syntax element if set to 1 indicates that the ObjectDescriptor includes a sequence of additional descriptors.

descriptorLength **–** This syntax element indicates the length in byte of the subsequent list of extensionDescriptors.

λ 7.3.3.2 ES_descriptor

The ES_Descriptor conveys all information related to a particular Elementary Stream and consists of three major parts:

The first part consists of the ES_number which is a unique reference to the Elementary Stream within this ObjectDescriptor, a mechanism to group Elementary Streams within this ObjectDescriptor and an optional URL string. Grouping of Elementary Streams and usage of URLs are specified in Subclause .

The second part consists of the DecoderConfigDescriptor, ALConfigDescriptor, IPI_Descriptor and QoS_Descriptor structures which convey the exact characteristics and requirements of the Elementary Stream.

The third part is an optional extension mechanism, whose semantic supports the inclusion of future extensions in a backward compatible way, as well as the transport of private data.

An ES_descriptor may be used as a pointer to a remote ObjectDescriptor by means of an URL, as specified in detail in Subclause . In that case the second and third part of the ES_descriptor are not present, with the exception of DecoderConfigDescriptor.

*7.3.3.2.1 Syntax

```
class ES_Descriptor () {
      uint(5) ES_Number;
      uint(1) streamDependence;
      uint(1) URLflag;
      uint(1) extensionFlag;
      if (URLflag) {
            uint(8) URLlength;
            uint(URLlength*8) URLstring;
            uint(1) remoteODflag;
      }
```

```
      if (streamDependence) {
            uint(5) dependsOn_ES_number;
      }
      aligned(8)  DecoderConfigDescriptor decConfigDescr;
      if (decConfigDescr.streamType!=initialObjectDescriptor) {
            ALConfigDescriptor alConfigDescr;
            IPI_Descriptor ipiDescr;
            QoS_Descriptor qosDescr;
            if (extensionFlag==1) {
                  uint(8) descriptorLength;
                  for (i=0,j=0;j<descriptorLength;i++) {
                        extensionDescriptor extDescr[i];
                        j = j + lengthof(extDescr[i]);
                  }
            }
      }
}
```

*7.3.3.2.2 Semantics

ES_Number – This syntax element provides a unique label for each Elementary Stream associated to this ObjectDescriptor. Values of zero and 31 for ES_number are reserved and shall not be assigned.

The combination of the objectDescriptorID and the ES_number forms the ES_Id which uniquely identifies each Elementary Stream within one MPEG-4 session. The ES_Id is a 16 bit value that is obtained by the following formula:

ES_Id = objectDescriptorID << 5 & ES_number

streamDependence - If set to one indicates that a dependsOn_ES_number will follow.

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

extensionFlag - if set to 1 indicates that the ES_Descriptor includes a list of extensionDescriptors.

URLlength - specifies the length of URLstring in byte.

URLstring - contains a URL.

remoteODflag - if set to zero indicates that the data retrieved from the URL consists of an AL-packetized Elementary Stream. The parameters of this Elementary Stream are fully specified in this ES_descriptor.

If set to one remoteODflag indicates that the first data retrieved from the URL is a new ObjectDescriptor.

For a streamType of this Elementary Stream equal to initialObjectDescriptor the semantics of this new Object Descriptors are specified in .

Else this new Object Descriptor shall contain only one ES_descriptor that replaces the content of the current ES_Descriptor. The streamType of this new ES_descriptor shall have the same value as in the original ES_descriptor. If this is not the case, the remote content shall be considered unusable. The parameters of the AL-packetized Elementary Stream that is subsequently retrieved from the URL are specified in this new ES_descriptor.

dependsOn_ES_number - gives the ES_number of another Elementary Stream that is associated to this ObjectDescriptor on which this Elementary Stream depends. The semantics of this syntax element is linked with that of streamType as specified in Subclause .

decConfigDescr - is a DecoderConfigDescriptor as specified in Subclause .

alConfigDescr - is a ALConfigDescriptor as specified in Subclause .

ipiDescr - is a IPI_Descriptor as specified in Subclause .

qosDescr - is a QoS_Descriptor as specified in Subclause .

descriptorLength - is the length in byte of the list of extensionDescriptors included in the ES_Descriptor.

extDescr[i] - is an extensionDescriptor as specified in Subclause .

λ 7.3.3.3 DecoderConfigDescriptor

The DecoderConfigDescriptor provides information about the decoder type and the required decoder resources needed for the associated Elementary Stream. This is needed at the receiver to determine whether it is able to decode the Elementary Stream.

The streamType identifies the category of the stream while an optional number of specificInfoBytes contain stream specific information for the set up of the decoder in a stream specific format opaque to this layer.

7.3.3.3.1 Syntax

```
class DecoderConfigDescriptor () {
      uint specificInfoLength; // not read from bitstream here
      uint(8) profileAndLevelIndication;
      uint(6) streamType;
      uint(1) upStream;
      uint(1) specificInfoFlag;
      uint(24) bufferSizeDB;
      uint(32) maxBitrate;
      uint(32) avgBitrate;
      if (specificInfoFlag) {
            uint(8) length8;
            if (length8==0) {
                  uint(32) length32;
                  specificInfoLength = length32;
            } else {
                  specificInfoLength = length8;
            }
            for (i=0; i<specificInfoLength; i++) {
                  uint(8) specificInfoByte[i];
            }
      }
}
```

*7.3.3.3.2 Semantics

profileAndLevelIndication - an indication of the profile and level that needs to be supported by the decoder for this Elementary Stream as per this table.

Table 7-4: profileAndLevelIndication Values

Value	profileAndLevelIndication Description
0x00	reserved for ISO use
0x01	Visual 14496-2 simple profile
0x02	Visual 14496-2 core profile
0x03	Visual 14496-2 simple facial anim profile
0x04	Visual 14496-2 scalable image texture profile
0x05	Audio 14496-3 main natural
0x06	Audio 14496-3 simple scalable
0x07	Audio 14496-3 simple parametric
0x08	Audio 14496-3 main synthetic
0x09	Audio 14496-3 algorithmic synthesis

0x0A	Audio 14496-3 wavetable synthesis
0x0B	Audio 14496-3 general MIDI
0x0C	Audio 14496-3 TTS
0x0D	Visual 13818-2 SP@ML
0x0E	Visual 13818-2 MP@LL
0x0F	Visual 13818-2 MP@ML
0x10	Visual 13818-2 MP@H-1440
0x11	Visual 13818-2 MP@HL
0x12	Visual 13818-2 SNR@LL
0x13	Visual 13818-2 SNR@ML
0x14	Visual 13818-2 Spatial@H-1440
0x15	Visual 13818-2 HP@ML
0x16	Visual 13818-2 HP@H-1440
0x17	Visual 13818-2 HP@HL
0x18	Visual 13818-2 422@ML
0x19	Audio 13818-7
0x1A	Audio 13818-3
0x1B	Visual 11172-2
0x1C	Audio 11172-3
0x1D - 0x7F	MPEG reserved
0x80 - 0xFF	user private

streamType - conveys the type of this Elementary Stream as per this table.

Table 7-5: streamType Values

streamType value	**stream type description**
0x00	reserved for ISO use
0x01	initialObjectDescriptor (see Subclause)
0x02	ObjectDescriptorStream (see Subclause)
0x03	ClockReferenceStream (see Subclause)
0x04	SceneDescriptionStream
0x05	VisualStream
0x06	AudioStream
0x07	MPEG7Stream
0x08-0x09	MPEG reserved
0x0A	ObjectContentInfoStream (see Subclause)
0x0B - 0x1F	MPEG reserved
0x20 - 0x3F	user private

upStream - indicates that this stream is used for upstream information.

specificInfoFlag - if set to one indicates presence of optional specificInfoBytes.

bufferSizeDB - is the size of the decoding buffer for this Elementary Stream in bytes.

maxBitrate - is the maximum instantaneous bitrate of this Elementary Stream in any time window of one second duration.

avgBitrate - is the average bitrate of this Elementary Stream in any time window of one minute duration. A value of zero indicates that this Elementary Stream does not have a constant average bitrate.

specificInfoLength - count of the optional specificInfoBytes.

specificInfoByte[i] - one byte of stream specific information

λ 7.3.3.4 ALConfigDescriptor

This descriptor configures the Access Unit Layer parser for this Elementary Stream. The specification of this descriptor is defered to Subclause of the specification of the Access Unit Layer for better readability.

λ 7.3.3.5 IPI_Descriptor

The IPI_Descriptor includes a mechanism to identify content. This is done by means of one or more IP_IdentificationDataSet included in this IPI_Descriptor.

Alternatively a pointer to the Elementary Stream that includes the data sets that are valid for this stream is provided. This pointer mechanism allows to transmit an IP_IdentificationDataSet only in one ES_Descriptor while making reference to it from all other ES_Descriptors that share the same IP_IdentificationDataSet.

*7.3.3.5.1 Syntax

```
class IPI_Descriptor() {
      uint(1) IPI_Pointer;
      if (IPI_Pointer) {
            uint(16) IPI_ES_Id;
      } else {
            uint(8)     IPI_Length;
            for (i=0, j=0; i<IPI_Length;j++ ) {
                  IP_InformationDataSet IP_IDS[j];
                  i = i + lengthof(IP_IDS[j]);
            }
      }
}
```

*7.3.3.5.2 Semantics

IPI_Pointer - if set to one indicates that an IPI_ES_Id will follow.

IPI_ES_Id - is the ES_Id of the Elementary Stream that contains the IP Information valid for this Elementary Stream. This ES_Id is unique within the MPEG-4 session.

IPI_Length - is the length of the subsequent list of IP_InformationDataSet in byte.

IP_IDS[j] - is an IP_InformationDataSet as specified in Subclause .

*7.3.3.5.3 IP Identification Data Set

The ♯ Intellectual Property Identification Data Set ♯ is used to identify content. All types of Elementary Streams carrying content can be identified using this meachanism. The content types include audio, visual and scene description data.

l7.3.3.5.3.1 Syntax

```
class IP_InformationDataSet () {
      uint(2)           compatibility;
      uint(1)           type_of_content_present;
      if (type_of_content_present)
            uint(8)     type_of_content;
      uint(1)           content_identifier_present;
      if (content_identifier_present == 1) {
            uint(8)     type_of_content_identifier;
            uint(8)     lenght_of_content_identifier;
            uint(lenght_of_content_identifier*8)
                        content_identifier;
      }
```

```
      uint(8)           number_of_supplementary_content_identifiers;
      if (number_of_supplementary_content_identifiers>0) {
            uint(24)    language_code;
            for (i=0; i < number_of_supplementary_content_identifiers; I++) {
                  uint(8)     length_of_supplementary_content_identifier_title;
                  uint(length_of_supplementary_content_identifier_title*8)
                        supplementary_content_identifier_title;
                  uint(8)     length_of_supplementary_content_identifier_value;
                  uint(length_of_supplementary_content_identifier_value*8)
                        supplementary_content_identifier_value;
            }
      }
}
```

l7.3.3.5.3.2 Semantics

`compatibility` - has to be set to 00. If an MPEG-4 version 1 player reads a value different from 00, the attached AVO need not be played.

`type_of_content_present` - flag to indicate if a definition of the type of content is available.

`type_of_content` - defining a type of content with a value of the following table

Table 7-6: type_of_content 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
8-254	Reserved for future use
255	Others

`content_identifier_present` - flag to indicate presence of creation ID.

`type_of_content_identifier` - defining a type of content identifier with a value of the following table.

Table 7-7: type_of_content_identifier 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)
9	SPIFF	Still Picture ID
10	DOI	Digital Object Identifier
11-255	Reserved for future use	

length_of_content_identifier - since the length of each of these identifiers can vary a length indicator is needed to give the length in byte.

content_identifier - international code identifying the content according to the preceding type_of_content_identifier.

number_of_supplementary_content_identifiers - since not all works are having a numbered identification scheme (yet), non-standard schemes can be used (which can be alphanumerical or binary). The number_of_supplementary_content_identifiers indicates, how many of these data ♯supplementary♯ data fields are following

language code - This 24 bits field contains the ISO 639 three character language code of the language of the following text fields. Remember that for languages that only use Latin characters, just one byte per character is needed in Unicode. Otherwise two byte are needed.

supplementary_content_identifier_title and supplementary_content_identifier_value - Each of these two entries give a title-and-value pair such as (♯Title♯,♯Hey Jude♯) whenever a numeric content definition is not available. However, the supplementary_content_identifier_value can contain ♯binary♯ information as well. The length of the title (in byte) is being indicated by length_of_supplementary_content_identifier_title (0 byte to 255 byte) and the length of the supplementary_content_identifier_value is indicated by the length_of_supplementary_content_identifier_value (0 byte to255).

λ 7.3.3.6 QoS_Descriptor

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

7.3.3.6.1 Syntax

```
class QoS_Descriptor () {
      uint(5) streamPriority;
      uint(3) predefined;
      if (predefined==0) {
            uint(8) QoS_QualifierCount;
            for (i=0; i<QoS_QualifierCount; i++) {
                  uint(8) QoS_QualifierLength;
                  uint(8*QoS_QualifierLength) QoS_Qualifier[i];
            }
      }
}
```

7.3.3.6.2 Semantics

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.

predefined - a value unequal zero indicates a predefined QoS profile according to the table below.

Table 7-8: Predefined QoS_Descriptor Values

predefined			
0x00 custom	Custom		
0x01 - 0xff reserved			

λ 7.3.3.7 extensionDescriptor

Additional descriptors may be defined by ISO using the following syntax.

7.3.3.7.1 Syntax

```
class extensionDescriptor () {
      uint(8) descriptorTag;
      uint(8) descriptorDataLength;
      for (j=0;j<descriptorDataLength;j++) {
            uint(8) descriptorDataByte[j];
      }
}
```

7.3.3.7.2 Semantics

descriptorTag - is one of the registered tags from this table.

Table 7-9: descriptorTag Values

0x00	reserved
0x01-0x7F	reserved for ISO use
0x80-0xFF	user private

descriptorDataLength - is the number of descriptorDataBytes.

descriptorDataByte[j] - is a data byte.

l7.3.4 Usage of Object Descriptors

l7.3.4.1 Association of Object Descriptors to Media Objects

Object Descriptors are associated to specific Media Objects within a Scene Description by means of the objectDescriptorID. Each Media Object has a specific type (Audio, Visual, sub-sceneDescription, etc.) and shall therefore only be related to an Object Descriptor that advertises Elementary Streams that are compatible to the type of the Media Object.

The behavior of the MPEG-4 terminal is undefined if an Object Descriptor advertises Elementary Streams with stream types that are incompatible to the associated Media Object.

λ 7.3.4.2 Rules for Grouping Elementary Streams within one ObjectDescriptor

A grouping mechanism is established by allowing a list of Elementary Streams to be present within one Object Descriptor. The syntax elements (see Subclause) streamDependence together with dependsOn_ES_number, as well as streamType constitute a tool to qualify this grouping further.

Since Object Descriptors are refered to from specific Media Objects within a Scene Description (Video and Audio objects as well as inlined sub-scenes), the grouping of Elementary Streams (ES) within one ObjectDescriptor (OD) has to follow a number of semantical rules, as detailed below.

- An OD shall only group ESs with compatible streamType. This means that the ESs within one OD shall all have either a visualStream, audioStream or SceneDescriptionStream type. The following exceptions exist:
  - An OD may contain zero or one additional ES of streamType = ObjectContentInfoStream. This ObjectContentInfoStream is valid for the content conveyed through the other ESs advertised in this OD.

- An OD may contain any number of additional ESs of streamType = ClockReferenceStream. A ClockReferenceStream is valid for those ES within the MPEG-4 session that refer to the ES_Id of this ClockReferenceStream in their ES_descriptor.
- An OD that contains ESs of streamType = SceneDescriptionStream may contain any number of additional ESs with streamType = ObjectDescriptorStream.

- An ES may depend on another ES in the same OD, indicated by a dependsOn_ES_number. This dependance is governed by the following rules:
  - For dependant ES of streamType equal to either audioStream, visualStream or SceneDescriptionStream the dependent ES shall have the same streamType as the independent ES. This implies that the dependent stream contains enhancement information to the one it depends on. The semantic meaning of enhancement depends on the decoder that is associated to these streams and is opaque at this layer.
  - An ES that flows upstream, as indicated by DecoderConfigDescriptor.upStream = 1 shall always depend upon another ES of the same *streamType* that has the upStream flag set to zero. This implies that this upstream is associated to the downstream it depends on.
  - An ES with streamType = SceneDescriptionStream may depend on a stream with streamType = ObjectDescriptorStream. This implies that the ObjectDescriptorStream contains the Object Descriptors that are associated to this SceneDescriptionStream.
    - This further implies that if this OD contains another ES with streamType = SceneDescriptionStream that depends on the first SceneDescriptionStream the Object Descriptors in the ObjectDescriptorStream are valid for this additional SceneDescriptionStream as well.

- Multiple ES which do not depend on other ESs and have the same streamType convey alternative representations of the same content. ESs are ordered according to preference. The ES with the lower ES_number is prefered over an ES with a higher ES_number.
- An OD which contains one ES with streamType = initialObjectDescriptor shall not contain any other ES. The sole purpose of this OD is to provide a pointer to remote content. All descriptive information about this remote content shall be retrieved from the remote Object Descriptor that is being pointed to by this OD.

In case of stream dependency, the availability of the dependent stream is undefined if an ES_Descriptor for the stream it depends upon is not yet available.

Note: The receiving terminal is expected to evaluate the ES_descriptors of all available Elementary Streams 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. This subset may be determined by some profile indication.

λ 7.3.4.3 Usage of URLs in Object Descriptors

URLs may be conveyed in ES_descriptors to locate an Elementary Stream. This is an alternative method to the content location by means of the numerical ES_Id (the concatentation of objectDescriptorID and ES_number). The usage of ES_Id constrains content references to the current MPEG-4 session (see Subclause), while the URL allows to reference content from remote locations.

Subclause specifies the necessary steps to retrieve the Elementary Stream identified either by a URL or an ES_Id. This subclause specifies the rules for the usage of URLs in ES_descriptors.

A URL may be present in an ES_descriptor with any streamType.

For all streamType*s* except initialObjectDescriptor, the URL shall point to remote content consisting of one single Elementary Stream. Depending on the value of remoteODflag, the parameters of this Elementary Stream are either known a priori by means of the local Object Descriptor or only after a connection to the remote content has been made and a remote

Object Descriptor has been retrieved. This remote Object Descriptor shall only contain one single ES_descriptor that shows the same streamType as the local ES_descriptor.

For streamType equal to initialObjectDescriptor, the URL shall point to remote content consisting of one Object Descriptor only. This Object Descriptor shall contain all necessary information to set up a new MPEG-4 session as specified in Subclause .

λ 7.3.4.4 Object Descriptors and the MPEG-4 Session

Unique labels for an Object Descriptor and an Elementary Stream declared within an OD are given by objectDescriptorID and ES_Id, respectively. In order to facilitate the merging of content produced at different sources, these labels need to have a suitable scope. This scope is defined by the MPEG-4 session.

The term MPEG-4 session as well as the initial data to be retrieved at session set up are specified in this subclause. This allows to subsequently specify how the scope for objectDescriptorID and ES_Id is determined.

˙7.3.4.4.1 MPEG-4 session

An MPEG-4 session is defined as a peer-to-peer relation between one MPEG-4 content source and the receiving MPEG-4 terminal. An MPEG-4 session consists of an arbitrary number of Elementary Streams that are exchanged between the peer MPEG-4 terminals.

No assumptions are made at this point how this session is created and how its components are identified or managed.

By definition, the start up of the connection between two peer MPEG-4 terminals creates a new MPEG-4 session. Furthermore a new MPEG-4 session is created each time a connection is made to a URL conveyed in an Object Descriptor with a streamType of initialObjectDescriptor.

As an exception, a connection made to a URL that just refers to a single Elementary Stream does not create a new MPEG-4 session.

An Inline node in the Scene Description incorporates one or more new streams of type SceneDescriptionStream in the scene graph. The relation between an Inline node and the MPEG-4 session is as follows:

- An Inline node that points to an Object Descriptor which identifies the new Scene Description streams by means of ES_ID does not open a new MPEG-4 session.
- An Inline node that uses a URL or points to an Object Descriptor that uses a URL to point to a stream of type initialObjectDescriptor implies the set up of a new MPEG-4 session.

l 7.3.4.4.2 The initial Object Descriptor

At the set up of an MPEG-4 session it is necessary to communicate which content streams are to be conveyed as part of this session. This task is accomplished by the first Object Descriptor that is retrieved at set up of the session.

This initial Object Descriptor shall be delivered to the MPEG-4 terminal in the format specified in Subclause , without any packetization or protocol overhead through a reliable channel. This means that it is assumed that the initial Object Descriptor is always delivered without error.

The initial Object Descriptor shall contain ESs of type SceneDescriptionStream and optionally ObjectDescriptorStream as required by the content to be conveyed in this session and according to the constraints specified in Subclause .

Note: As specified in the previous subclause, retrieval of an individual ES through a URL does not open a new MPEG-4 session. Therefore the Object Descriptor that optionally is retrieved in that case is not constrained in the above way.

7.3.4.4.3 Scope of objectDescriptorID and ES_ID labels

Each time a new MPEG-4 session is opened a new name scope for objectDescriptorID and, hence, ES_ID labels is created. This allows to hierarchically structure the name space for these labels. The Object Descriptors needed within this new MPEG-4 session, if any, are conveyed in the ObjectDescriptorStreams declared in the initial Object Descriptor of that session.

λ 7.3.4.5 Session set up

This subclause gives the pre-conditions and a walk through the procedure by which the MPEG-4 receiver acquires the first bitstream at the start of a new MPEG-4 session in accordance with the specifications of the previous subclauses. In Subclause the slightly different walk through the procedure for aquiring a single Elementary Stream from a remote location is summarized.

7.3.4.5.1 Pre-conditions

- A mechanism exists to open a session that takes a URL as input and provides some returned data as output. This may be embodied by the DA_ServiceAdd primitive of the DMIF Application Interface (DAI).
- A mechanism exists to open a channel that takes user data as input and provides some returned data as output. This may be embodied by the DA_ChannelAdd primitive of the DMIF Application Interface.
- A parser for Object Descriptors has been instantiated.

7.3.4.5.2 Session set up procedure

1. A connection to a URL is made, using a service set up call, e.g. DA_ServiceAdd.
2. The service set up call shall return data consisting of a single Object Descriptor.
3. This initial Object Descriptor contains ES_Descriptors as needed and as constrained by the specification in Subclause .
4. The ES_ID for the streams that are to be opened are determined.
5. Requests for delivery of the selected ESs are made, using a channel set up call, e.g. DA_ChannelAdd with the ES_ID as the uuData parameter.
6. The channel set up call shall return handles to the streams that correspond to the list of ES_IDs
7. The stream handles are passed to the appropriately configured AU Layer parsers for each ES_ID.
8. A confirmation that the terminal is ready to receive data is delivered to the sender, e.g., via DA_ChannelReady.
9. Delivery of streams starts
10. Scene Description and ObjectDescriptor stream are read
11. Further streams are opened as needed with the same procedure, starting at step 4.

l7.3.4.5.2.1 Example

The set up example in the drawing conveys an initial Object Descriptor that points to one SceneDescriptionStream, an optional ObjectDescriptorStream and additional optional SceneDescriptionStreams or ObjectDescriptorStreams.

Figure 7-18: Session setup example

7.3.4.5.3 Set up for retrieval of a single Elementary Stream from a remote location

Individual Elementary Streams may be retrieved from a remote location using either URLs in suitable Media Object declarations (e.g. VideoObject2D node, see Subclause) or URLs within an ES_descriptor. Reference to such a stream does not open a new MPEG-4 session, as specified in Subclause . However, the set up procedure is still required to open a connection to the remote location with the following steps:

1. A connection to a URL is made, using a service set up call, e.g. DA_ServiceAdd.
2. The service set up call shall optionally return data consisting of a single Object Descriptor, depending on the setting of the remoteODflag in the local ES_descriptor
3. This optional Object Descriptor contains a single ES_Descriptor specifying the configuration of the stream.
4. If not present, the local ES_descriptor specifies the configuration of the stream.
5. Request for delivery of the stream is made, using a channel set up call, e.g. DA_ChannelAdd with no parameter.
6. The channel set up call shall return a handle to the stream
7. The stream handle is passed to the appropriately configured AU Layer parser.
8. A confirmation that the terminal is ready to receive data is delivered to the sender, e.g., via DA_ChannelReady.
9. Delivery of stream starts

7.4 Synchronization of Elementary Streams

7.4.1 Introduction

This subclause of the specification defines the tools to maintain temporal synchronization between Elementary Streams. The conceptual elements, most notably time stamps and clock reference information, that are required for this purpose have already been introduced in Subclause . The syntax and semantics to convey these elements end-to-end is embodied in the Access Unit Layer, specified in Subclause . This syntax is configurable to adapt to the needs of different types of Elementary Streams. The required configuration information is specified in Subclause .

On the Access Unit Layer an Elementary Stream is mapped into a sequence of packets, called an AL-packetized Stream (APS). For this packetization information has to be exchanged between the entity that generates an Elementary Stream and the Access Unit Layer. This relation is best described by a conceptual interface between both layers, termed the Elementary Stream Interface (ESI). The ESI is a Reference Point that need not be accessible in an implementation. It is described in Subclause .

AL-packetized Streams are made available to a delivery mechanism outside the scope of this specification. This delivery mechanism is only described in terms of a conceptual Stream Multiplex Interface (SMI) that specifies the information that need to be exchanged between the Access Unit Layer and the delivery mechanism. The SMI is a Reference Point that need not be accessible in an implementation. The SMI may be embodied by the DMIF Application Interface specified in CD 14496-6. The required properties of the SMI are described in Subclause .

Note: The delivery mechanism described by the SMI serves to abstract any transmission as well as storage media. The basic data transport property that this delivery mechanism shall provide is the framing of the data packets generated by the AU Layer. The FlexMux tool (see) is an example for such a tool that may be used in the delivery protocol stack as required.

The items specified in this subclause are depicted in Figure 7-19 below.

Figure 7-19 Systems Layers

l7.4.2 Access Unit Layer

The Access Unit Layer (AU Layer or AL) specifies a syntax for the fragmentation of Elementary Streams into Access Units or parts thereof. These fragments are called AL-PDUs (AU Layer Protocol Data Unit). The sequence of AL-PDUs resulting from one Elementary Stream is called AL-packetized Stream (APS). Access Units are the only semantic entities at this layer and their content is opaque. They are used as the basic unit for synchronization.

An AL-PDU consists of an AL-PDU Header and an AL-PDU Payload. The AL-PDU Header supplies 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 Subclause on Systems Decoder Model. The AL-PDU Header is configurable as specified in Subclause . The AL-PDU Header itself is specified in Subclause .

Note: An AL-PDU does not contain an indication of its length. Therefore AL-PDUs must be framed by a suitable lower layer protocol using, e.g., the FlexMux Tool, specified in Subclause . Consequently an AL-packetized Stream is not a self-contained data stream that can be stored or decoded without such framing.

λ 7.4.2.1 AL-PDU Specification

Syntax and semantics of an AL_PDU are specified here. A pre-condition for an AL_PDU to be parsed is that the ALConfigDescriptor for the Elementary Stream to which the AL_PDU belongs is known since the ALConfigDescriptor conveys the configuration of the syntax of the AL-PDU Header.

7.4.2.1.1 Syntax

```
class AL_PDU (ALConfigDescriptor AL) {
      aligned (8) AL_PDU_Header alPduHeader(AL);
      aligned (8) AL_PDU_Payload alPduPayload;
}
```

7.4.2.1.2 Semantics

alPduHeader - an AL_PDU_Header element as specified in Subclause .

alPduPayload - an AL_PDU_Payload that contains an opaque payload.

λ 7.4.2.2 AL-PDU Header Configuration

The AL-PDU Header may be configured according to the needs of each individual Elementary Stream. The presence, resolution and accuracy of time stamps, clock references and other parameters, as detailed below, can be selected. With this flexibility, for example, a low bitrate Elementary Stream can choose to spend less overhead on AL-PDU Headers than a higher bitrate stream.

For each Elementary Stream the configuration is conveyed in an ALConfigDescriptor as part of the associated ES_descriptor within an Object Descriptor.

The configurable parameters in the AL-PDU Header can be divided in two classes: Those that apply to each AL-PDU (e.g. OCR, sequenceNumber) and those that are strictly related to Access Units (e.g. time stamps, accessUnitLength, instantBitrate, degradationPriority).

*7.4.2.2.1 Syntax

```
class ALConfigDescriptor {
      uint (8) predefined;
      if (predefined==0) {
            bit(1) useAccessUnitStartFlag;
            bit(1) useAccessUnitEndFlag;
            bit(1) useRandomAccessPointFlag;
            bit(1) usePaddingFlag;
            bit(1) useTimeStampsFlag;
            uint(32) timeStampResolution;
            uint(32) OCRResolution;
            uint(6) timeStampLength;
            uint(6) OCRLength;
            uint(1) useWallClockTimeStampFlag;
            if (!useTimeStampsFlag) {
                  uint(16) accessUnitRate;
                  uint(16) compositionUnitRate;
                  uint(timeStampLength) startDecodingTimeStamp;
                  uint(timeStampLength) startCompositionTimeStamp;
                  float(64) wallClockTimeStamp;
            }
            uint(5) AU_Length;
            uint(8) instantBitrateLength;
            uint(4) degradationPriorityLength;
            uint(4) seqNumLength;
      }
      bit(1) OCRstreamFlag;
      if (OCRstreamFlag) {
            uint(16) OCR_ES_Id;
      }
}
```

*7.4.2.2.2 Semantics

predefined - allows to default the values from a set of predefined parameter sets as detailed below.

Table 7-10: Overview of predefined ALConfigDescriptor values

predefined field value	Description
0x00	custom
0x01	null AL-PDU Header
0x02 - 0xFF	Reserved

Table 7-11: Detailed predefined ALConfigDescriptor values

predefined field value	0x01
useAccessUnitStartFlag	0
useAccessUnitEndFlag	0
useRandomAccessPointFlag	0
usePaddingFlag	0
useTimeStampsFlag	0
timeStampResolution	-
OCRResolution	-
timeStampLength	-
OCRlength	-
useWallClockTimeStampFlag	0
if (!useTimeStampsFlag)	
{	
accessUnitRate	-
compositionUnitRate	-
startDecodingTimeStamp	-
startCompositionTimeStamp	-
wallClockTimeStamp	-
}	
AUlength	-
instantBitrateLength	-
degradationPriorityLength	-
seqNumLength	-

useAccessUnitStartFlag - indicates that the accessUnitStartFlag is present in each AL-PDU Header of this Elementary Stream.

useAccessUnitEndFlag - indicates that the accessUnitEndFlag is present in each AL-PDU Header of this Elementary Stream.

If neither useAccessUnitStartFlag nor useAccessUnitEndFlag are set this implies that each AL-PDU corresponds to a single Access Unit.

It is illegal to set useAccessUnitStartFlag=0 and useAccessUnitEndFlag=1.

useRandomAccessPointFlag - indicates that the RandomAccessPointFlag is present in each AL-PDU Header of this Elementary Stream.

usePaddingFlag - indicates that the paddingFlag is present in each AL-PDU Header of this Elementary Stream.

useTimeStampsFlag - indicates that time stamps are used for synchronization of this Elementary Stream. They are conveyed in the AL-PDU Headers. Else accessUnitRate, compositionUnitRate, startDecodingTimeStamp and startCompositionTimeStamp conveyed in this AL-PDU Header Configuration shall be used for synchronization.

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

OCRresolution - is the resolution of the Object Time Base in cycles per second.

timeStampLength - is the length of the time stamp fields in AL-PDU Headers.

OCRlength - is the length of the objectClockReference field in AL-PDU Headers. A length of zero indicates that no objectClockReferences are present in this Elementary Stream. If OCRstreamFlag is set, OCRLength shall be zero.

useWallClockTimeStampFlag - indicates that wallClockTimeStamps are present in this Elementary Stream. If useTimeStampsFlag equals to zero, only one such time stamp is present in this ALConfigDescriptor.

accessUnitRate - is the rate at which Access Units are decoded in Hertz.

compositionUnitRate - is the rate at which the resulting Composition Units are presented in Hertz.

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.

wallClockTimeStamp - is a wall clock time stamp in SFTime format (see Subclause) that indicates the current wall clock time corresponding to the time indicated by startCompositionTimeStamp.

AU_Length - is the length of the accessUnitLength fields in AL-PDU Headers for this Elementary Stream.

instantBitrateLength - is the length of the instantBitrate field in AL-PDU Headers for this Elementary Stream.

degradationPriorityLength - is the length of the degradationPriority field in AL-PDU Headers for this Elementary Stream.

seqNumLength - is the length of the sequenceNumber field in AL-PDU Headers for this Elementary Stream.

OCRstreamFlag - indicates that an OCR_ES_ID syntax element will follow.

OCR_ES_ID - indicates the Elementary Stream from which the time base for this Elementary Stream is derived . This OCR_ES_Id is unique within the MPEG-4 session.

If the AL-PDU Header is configured such that it does not have a length of an integer number of bytes, zero bit stuffing shall be applied for byte alignment.

λ 7.4.2.3 AL-PDU Header Specification

The AL-PDU Header, as configured per the ALConfigDescriptor has the following structure.

7.4.2.3.1 Syntax

```
class AL_PDU_Header (ALConfigDescriptor AL) {
      default uint      accessUnitStartFlag = 1;
      default uint      randomAccessPointFlag = 0;
      default uint      accessUnitEndFlag = 1; // should read: (subsequent
                              // AL-PDU has accessUnitStartFlag==1) ? 1 : 0
      default uint      OCRflag = 0;
      default uint      paddingFlag = 0;
      default uint      paddingBits = 0;
      default uint      decodingTimeStampFlag = 0;
      default uint      compositionTimeStampFlag = 0;
      default uint      wallClockTimeStampFlag = 0;
      default uint      instantBitrateFlag = 0;
      default uint      accessUnitLength = 0;
      if (AL.useAccessUnitStartFlag) {
```

```
            uint(1)           accessUnitStartFlag;
      }
      if (AL.useRandomAccessPointFlag) {
            uint(1)           randomAccessPointFlag;
      }
      if (AL.useAccessUnitStartFlag && AL.useAccessUnitEndFlag) {
            uint(1)           accessUnitEndFlag;
      }
      if (AL.OCRLength>0) {
            uint(1)           OCRflag;
      }
      if (AL.usePadding) {
            uint(1)           paddingFlag;
      }
      if (paddingFlag) {
            uint(3)           paddingBits;
      }
      if (!paddingFlag || paddingBits!=0) {
            if (AL.seqNumLength>0) {
                  uint(AL.seqNumLength)   sequenceNumber;
            }
            if (OCRflag) {
                  uint(AL.OCRlength)      objectClockReference;
            }
            if (AL.useAccessUnitStartFlag && accessUnitStartFlag) {
                  if (AL.useTimeStampsFlag) {
                        uint(1)     decodingTimeStampFlag;
                        uint(1)     compositionTimeStampFlag;
                        if (AL.useWallClockTimeStampFlag)
                              uint(1) wallClockTimeStampFlag;
                  }
                  if (AL.instantBitrateLength>0) {
                        uint(1)     instantBitrateFlag;
                  }
                  if (decodingTimeStampFlag) {
                        uint(AL.timeStampLength)      decodingTimeStamp;
                  }
                  if (compositionTimeStampFlag) {
                        uint(AL.timeStampLength)      compositionTimeStamp;
                  }
                  if (wallClockTimeStampFlag) {
                        float(64)                     wallClockTimeStamp;
                  }
                  if (AL.AUlength > 0) {
                        uint(AL.AUlength)             accessUnitLength;
                  }
                  if (instantBitrateFlag) {
                        uint(AL.instantBitrateLength) instantBitrate;
                  }
                  if (AL.degradationPriorityLength>0) {
                        uint(AL.degradationPriorityLength) degradationPriority;
                  }
            }
      }
}
```

*7.4.2.3.2 Semantics

accessUnitStartFlag - when set to one indicates that an Access Unit starts in this AL-PDU. If this syntax element is omitted from the AL-PDU Header configuration its default value is one, i. e., each AL-PDU starts a new Access Unit.

accessUnitEndFlag - when set to one indicates that an Access Unit ends in this AL-PDU. If this syntax element is omitted from the AL-PDU Header configuration its default value is only known after reception of the subsequent AL-PDU with the following rule:

accessUnitEndFlag = (subsequent-AL-PDU has accessUnitStartFlag==1) ? 1 : 0.

If neither AccessUnitStartFlag nor AccessUnitEndFlag are configured into the AL-PDU Header this implies that each AL-PDU corresponds to a single Access Unit, hence both accessUnitStartFlag = accessUnitEndFlag = 1.

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 AL-PDU Header configuration its default value is zero, i. e., no random access points are indicated.

OCRflag - when set to one indicates that an objectClockReference will follow. The default value for OCRflag is zero.

paddingFlag - indicates the presence of padding in this AL-PDU. The default value for paddingFlag is zero.

paddingBits - indicate the mode of padding to be used in this AL-PDU. The default value for paddingBits is zero.

If paddingFlag is set and paddingBits is zero, this indicates that the subsequent payload of this AL-PDU 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 AL-PDU is followed by paddingBits of zero stuffing bits for byte alignment of the payload.

sequenceNumber - if present, shall be continuously incremented for each AL-PDU as a modulo counter. A discontinuity at the decoder corresponds to one or more missing AL-PDU. In that case an error shall be signaled to the Access Unit Layer user. If this syntax element is omitted from the AL-PDU Header configuration, a continuity checking by the AU Layer cannot be performed for this Elementary Stream.

Duplication of AL-PDUs: Elementary Streams that have a sequenceNumber field in their AL-PDU headers may use duplication of AL-PDUs for error resilience. This is restricted to only one duplication. The sequenceNumber of both AL-PDUs shall have the same value and each byte of the original AL-PDU shall be duplicated, with the exception of an objectClockReference field, if present, which shall encode a valid value for the duplicated AL-PDU.

objectClockReference - contains an Object Clock Reference time stamp as specified in Subclause . objectClockReference is only present in the AL-PDU Header if OCRflag is set.

Note: It is possible to convey just an OCR value and no payload within an AL-PDU.

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:

decodingTimeStampFlag - indicates that a Decoding Timestamp is present for this Access Unit.

compositionTimeStampFlag - indicates that a Composition Timestamp is present for this Access Unit.

wallClockTimeStampFlag - indicates that a wallClockTimeStamp is present for this Access Unit.

accessUnitLengthFlag - indicates that the length of this Access Unit is signaled.

instantBitrateFlag - indicates that an instantBitrate is signaled.

decodingTimeStamp - is a Decoding Timestamp as configured in ALConfigDescriptor.

compositionTimeStamp - is a Composition Timestamp as configured in ALConfigDescriptor.

wallClockTimeStamp - is a wall clock Timestamp in SFTime format.

accessUnitLength - is the length of the Access Unit in byte. If this syntax element is not present or has the value zero, the length of the Access Unit is unknown.

instantBitrate - is the instantaneous bitrate of this Elementary Stream until the next instantBitrate field is found.

degradationPriority - indicates the importance of the payload of this Access Unit. The streamPriority defines the base priority of an ES. degradationPriority defines a decrease in priority for this Access Unit relative to the base priority. The priority for this Access Unit is given by

AccessUnitPriority = streamPriority - degradationPriority

degradationPriority remains at this value until its next occurrence. This indication is used for graceful degradation by the decoder of this Elementary Stream. The relative amount of complexity degradation among Access Units of differenet Elementary Streams becomes more as AccessUnitPriority decreases.

λ 7.4.2.4 Clock Reference Stream

An Elementary Stream of streamType = ClockReferenceStream may be declared by means of the Object Descriptor syntax. It is used for the sole purpose to convey Object Clock Reference time stamps. Multiple Elementary Streams in an MPEG-4 session may make reference to such a ClockReferenceStream by means of the OCR_ES_ID syntax element in the ALConfigDescriptor to avoid redundant transmission of Clock Reference information.

On the Access Unit Layer a ClockReferenceStream is realized by configuring the AL-PDU Header syntax for this AL-packetized Stream such that only OCR values of the required OCRresolution and OCRlength are present in the AL-PDU Header.

There shall not be any AL-PDU Payload present in an AL-packetized Stream of streamType = ClockReferenceStream.

The following ALConfigDescriptor elements have recommended values for a Clock Reference Stream:

```
useAccessUnitStartFlag=0;
useAccessUnitEndFlag=0;
useRandomAccessPointFlag=0;
usePaddingFlag=0;
useTimeStampsFlag=0;
timeStampResolution=0;
timeStampLength=0;
useWallClockTimeStampFlag=0;
```

l7.4.3 Elementary Stream Interface

The Elementary Stream Interface (ESI) specifies which data need to be exchanged between the entity that generates an Elementary Stream and the Access Unit Layer. It is described here to clarify that in case that both layers are implemented as separate entities the communication between the two layers occurs not only in terms of a compressed media data stream but as well in terms of additional information to convey time codes, length of Access Units, etc.

An implementation of MPEG-4 does not have to implement the Elementary Stream Interface. Instead it is possible to integrate parsing of the AL-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 Stream Multiplex Interface (see Subclause) rather than a data stream.

The interface to receive Elementary Stream data from the AU Layer has a number of parameters that reflect the side information that has been retrieved while parsing the incoming AL-packetized stream:

ESI.receiveData (*ESdata, dataLength, decodingTimeStamp, compositionTimeStamp, accessUnitStartFlag, randomAccessFlag, accessUnitEndFlag, degradationPriority, errorStatus*)

ESdata - a number of *dataLength* data bytes for this Elementary Stream

dataLength - the length in bytes of *ESdata*

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

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 AU Layer requires the following parameters that will subsequently be encoded on the AU Layer:

ESI.sendData (*ESdata, dataLength, decodingTimeStamp, compositionTimeStamp, accessUnitStartFlag, randomAccessFlag, accessUnitEndFlag, degradationPriority*)

ESdata - a number of *dataLength* data bytes for this Elementary Stream

dataLength - the length in bytes of *ESdata*

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

degradationPriority - indicates the degradation priority for this Access Unit

l7.4.4 Stream Multiplex Interface

The Stream Multiplex Interface (SMI) specifies which data needs to be exchanged between the Access Unit Layer and the delivery mechanism below. It is described here to clarify that in case that both layers are implemented as separate entities the communication between the two layers occurs not only in terms of an AL-packetized Stream but as well in terms of additional information to convey the length of each AL-PDU.

An implementation of MPEG-4 does not have to expose the Stream Multiplex Interface. However, each MPEG-4 terminal shall have the functionality described by the SMI to be able to receive the AL-PDUs that constitute an AL-packetized Stream. Specifically the delivery mechanism below the AU Layer shall supply a method to frame or otherwise encode the length of the AL-PDUs transported through it.

A superset of the required SMI functionality is embodied by the DMIF Application Interface specified in CD 14496-6. The DAI has DA_data primitives to receive and send data which conveys a number of data bytes. This interface has to be constrained in the following way: Each invocation of a DA_data.Request or a DA_Data.Indication shall transfer one AL-PDU between the AU Layer and the delivery mechanism below.

7.5 Multiplexing of Elementary Streams

7.5.1 Introduction

Elementary Stream data encapsulated in AL-packetized Streams are sent/received through the Stream Multiplex Interface, as specified in . Multiplexing procedures and the architecture of the delivery protocol layers themselves are not in the scope of this specification. However, care has been taken to define the Access Unit Layer syntax and semantics such that AL-packetized Streams can be easily embedded in various transport protocol stacks. The abstract name “TransMux” is used to refer generically to any such protocol stack. Some examples for the embedding of AL-packetized Streams in various TransMux protocol stacks are given in informative Annex C.

The analysis of existing TransMux protocol stacks has shown that in case of stacks with fixed length packets (e.g. MPEG-2 Transport Stream) or with rather high multiplexing overhead (Internet protocol stack, i.e. (RTP/)UDP/IP) it may be advantageous to have a generic, low complexity, multiplexing tool that allows to interleave data from multiple AL-packetized Streams with low overhead, enabling low delay, low bitrate applications. Such a multiplex tool that may optionally be used by applications is specified in this subclause.

l7.5.2 FlexMux Tool

The FlexMux tool is a flexible multiplex specification to accommodate interleaving of AL-packetized Streams with largely varying instantaneous bit rate. The basic data entity of the FlexMux is a FlexMux-PDU that has a variable length. One or more AL-PDUs are embedded in a FlexMux-PDU as specified in detail in the remainder of this subclause. The FlexMux tool provides identification of AL-PDUs originating from different Elementary Streams by means of FlexMux Channel numbers. Each AL-packetized Stream is mapped into one FlexMux Channel. FlexMux-PDUs with data from different AL-packetized Streams can therefore be arbitrarily interleaved. The sequence of FlexMux-PDUs that are interleaved into one stream are called FlexMux Stream.

A FlexMux Stream retrieved from a storage or transmission media can be parsed as a single data stream without the need for any side information. However, the FlexMux requires framing of FlexMux-PDUs by the underlying layer for random access or error recovery. There is no requirement to frame each individual FlexMux-PDU. 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 exist. They are called Simple Mode and MuxCode Mode. A FlexMux Stream may contain an arbitrary mixture of FlexMux-PDUs using either Simple Mode or MuxCode Mode. The syntax and semantics of both modes are specified hereafter.

λ 7.5.2.1 Simple Mode

In the simple mode one AL-PDU is encapsulated in one FlexMux-PDU and tagged by an index which is equal to the FlexMux Channel number as indicated in Figure 7-20. This mode does not require any configuration or maintenance of state by the receiving terminal.

Figure 7-20 : Structure of FlexMux-PDU in simple mode

λ 7.5.2.2 MuxCode mode

In the MuxCode mode one or more AL-PDUs are encapsulated in one FlexMux-PDU as indicated in Figure 7-21. This mode needs configuration and maintenance of state by the receiving terminal. The configuration describes how FlexMux-PDUs are shared between multiple AL-PDUs. In that mode the index value needs dereferencing to know to which FlexMux Channels the payload of a FlexMux-PDU belongs.

Figure 7-21: Structure of FlexMux-PDU in MuxCode mode

λ 7.5.2.3 FlexMux-PDU specification

The two modes of the FlexMux, Simple Mode and MuxCode Mode are distinguished by the value of index as specified by the following syntax and semantics.

7.5.2.3.1 Syntax

```
class FlexMux_PDU () {
      uint(8) index;
      uint(8) length;
      if (index>239) {
            uint(4) version;
            uint(4) reserved;
            multiple_AL_PDU payload;
      } else {
            AL_PDU payload;
      }
}
```

7.5.2.3.2 Semantics

index - if index is smaller than 240 then

FlexMux Channel = index

of the payload of this FlexMux-PDU. This range of values corresponds to the Simple Mode.

If index ranges from 240 to 255 this indicates that 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 Subclause .

Note *Although the number of FlexMux Channels is limited to 256, through the use of multiple TransMux Channels virtually any number of Elementary Streams may be transmitted.*

length is the length of the FlexMux-PDU payload in bytes. This is equal to the length of the single encapsulated AL-PDU in Simple Mode and to the total length of the multiple encapsulated AL-PDUs in MuxCode Mode.

version indicates the current version of the MuxCodeTableEntry referenced by MuxCode. Version is used for error resilience. If this version does not match the version of the referenced MuxCodeTableEntry that has been received most recently, the FlexMux-PDU cannot be parsed. The implementation is free to either wait until the required version of MuxCodeTableEntry becomes available or to discard the FlexMux-PDU.

payload is either a single AL-PDU (Simpe Mode) or a number of one or more AL-PDUs (MuxCode Mode)

*7.5.2.3.3 Configuration for MuxCode Mode

The configuration for MuxCode Mode is signaled by MuxCodeTableEntry messages. The syntax and semantics of MuxCodeTableEntry is specified here. The transport of the MuxCodeTableEntry shall be defined during the design of the transport protocol stack that makes use of the FlexMux tool. Committee Draft of International Standard 14496-6 proposes one 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 tight real-time performance is required for this control channel since version numbers allow to detect FlexMux-PDUs that cannot currently be decoded and, hence, trigger suitable action in the receiving terminal.

l7.5.2.3.3.1 Syntax

```
aligned(8) class MuxCodeTableEntry () {
      uint(8) length;
      uint(4) MuxCode;
      uint(4) version;
      uint(8) substructureCount;
      for (i=0; i<substructureCount; i++) {
            uint(5) slotCount;
            uint(3) repetitionCount;
            for (k=0; k<slotCount;  k++){
                  uint(8) flexMuxChannel[i][k];
                  uint(8) numberOfBytes[i][k];
            }
      }
}
```

l7.5.2.3.3.2 Semantics

length is the length in bytes of the remainder of the MuxCodeTableEntry following the length element.

MuxCode is the number by 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 is the number of substructures of this MuxCodeTableEntry

slotCount is 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] is the FlexMux Channel to which the data in this slot belongs.

numberOfBytes[i][k] is the number of data bytes in this slot associated to flexMuxChannel[i][k]. This number of bytes corresponds to one AL-PDU.

λ 7.5.2.4 Usage of MuxCode Mode

The MuxCodeTableEntry describes how a FlexMux-PDU is partitioned into slots that carry data from different FlexMux Channels. This is to be used as a template for parsing FlexMux-PDUs. If a FlexMux-PDU is longer than the template, parsing shall resume from the beginning of the template. If a FlexMux-PDU is shorter than the template, the remainder of the template is ignored.

*7.5.2.4.1 *Example*

SubstructureCount = 3
slotCount[i] = 2, 3, 2 (for the corresponding substructure)
RepetitionCount[i] = 3, 2, 1 (for the corresponding substructure)
with Bytes[i][k] of data for FlexMuxChannels FMC[i][k] in slot [i][k] would give a splitting of the FlexMux-PDU 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

yielding a FlexMux-PDU with the following content:

Figure 7-22 Example for a FlexMux-PDU in MuxCode mode

7.6 Syntactic Description Language

7.6.1 Introduction

This section describes the mechanism with which bitstream syntax is documented in this Working Draft of International Standard. This mechanism is based on a syntactic description language, documented here in the form of formal rules. It directly extends the C-like syntax used in International Standards ISO/IEC 11172 and 13818 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.” It then proceeds to extend the typing system by providing facilities for defining bitstream-level quantities, and how they should be parsed.

We first describe elementary constructs, then composite syntactic constructs, arithmetic and logical expressions, and finally address syntactic control flow and built-in functions. Syntactic flow control is needed to take into account context-sensitive data. Several examples are used to clarify the structure.

l7.6.2 Elementary Data Types

SDL uses the following elementary syntactic elements:

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 decoder.
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).
5.

These are described in more detail below. Note that all quantities are represented with the most significant byte first, and also with the most significant bit first.

λ 7.6.2.1 Constant-Length Direct Representation Bit Fields

Constant-length direct representation bit fields are represented as:

Rule E.1: Elementary Data Types

[aligned] *type*[(*length*)] *element_name* [= *value*]; // C++-style comments allowed

---

The *type* can be any of the following: ‘*int*’ for signed integer, ‘*unsigned int*’ for unsigned integer, ‘*double*’ for floating point, and ‘*bit*’ for raw binary data. ‘*length*’ indicates the length of the element in bits, as it is stored in the bitstream. Note that ‘double’ can only use 32 or 64 bit lengths. The *value* attribute is only present 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* are always present, except if the data is non-parsable, i.e., it is not included in the bitstream. The attribute ‘aligned’ means 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), can be used to signify alignment on other than byte boundary. Allowed values are 8, 16, 32, 64, and 128. An entity such as temporal reference would be represented as:

unsigned int(5) temporal_reference;

where ‘unsigned int(5)’ indicates that the element should be interpreted as a 5-bit unsigned integer. By default, data is represented with the most significant bit first, and the most significant byte first.

Note that constants are defined using the ‘const’ attribute:

```
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 is used, similar to the ‘0x’ prefix for hexadecimal numbers, and a period (‘.’) can be optionally placed every four digits for readability. Hence 0x0F is equivalent to 0b0000.11111.

In several instances it is desirable to examine the immediately following bits in the bitstream, without actually removing the bits. To support this behavior, a ‘*’ character can be placed after the parse size parentheses to modify the parse size semantics.

Rule E.2: Look-ahead parsing

[aligned] *type* (*length*)* *element_name*;

For example, we can check the value of next 32 bits in the bitstream as an unsigned integer without advancing the current position in the bitstream using the following representation:

```
aligned unsinged int (32)* next_code;
```

λ 7.6.2.2 Variable Length Direct Representation Bit Fields

This case is covered by Rule E.1, by allowing the ‘*length*’ field to be a variable included in the bitstream, a non-parsable variable, or an expression involving such variables. For example:

```
unsigned int(3) precision;
int(precision) DC;
```

λ 7.6.2.3 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 one can extract the final desired value. This indirection can 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) is always ‘bit’. The *output_type* entry is either a predefined type or a defined class (classes are defined in Subclause). 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 are specified as aggregates surrounded by curly braces, similar to C or C++ structures.

As an example, we have:

```
class YUVblocks {// classes are fully defined later on
   uint Yblocks;
   uint Ublocks;
   uint 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 has to be identical to the type returned from the map. Example:

YUVblocks(blocks_per_component) chroma_format;

Using the above declaration, we can access a particular value of the map using the construct: chroma_format.Ublocks.

λ 7.6.2.4 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 is 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 is 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.

This is illustrated in the following example (the class type ‘val’ is used, 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 asigned to the value of the first element (val.foo), while the following 32 bits will 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 must match the type used in the class declaration associated with the map’s return type.

l7.6.3 Composite Data Types

l7.6.3.1 Classes

Classes are the mechanism with which definition of composite types or objects is performed. Their definition is as follows.

Rule C.1: Classes

[aligned] 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 definitions of elementary bitstream components as we saw in Subclause , or control flow that is discussed later on.

The optional ‘extends *parent_class*’ specifies that the class is “derived” from another class. Derivation means that all information present in the base class is also present in the derived class, and that all such information *precedes* in the bitstream any additional bitstream syntax declarations that are specified in the new class.

The *object_id* is optional, 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.

A derived class can appear at any point where its base class is specified in the bitstream. In order to be able to determine if the base class or one of its derived classes is present in the bitstream, the *object_id* (or range) is used. This identifier is given a particular value (or range of values) at the base class; all derived classes then have to specify their own unique values (or ranges of values) as well. If a class declaration does not provide an *object_id* then class derivation is still allowed, but any derived classes cannot substitute their base class in the bitstream. This mechanism expresses the concept of “polymorphism” [5] in the context of bitstream syntax.

Examples:

```
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 bitstream components is important: it is the same order in which the elements appear in the bitstream. In the above examples, foo::b immediately precedes foo::c in the bitstream.

We can also encapsulate objects within other objects. In this case, the *element* mentioned at the beginning of this section is an object itself.

λ 7.6.3.2 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. Because SDL follows a declarative approach, references to other objects 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] 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 name and variable name pairs separated by commas. Any element of the bitstream, or value derived from the bitstream with a vlc, 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;
}

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
}
```

λ 7.6.3.3 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)’), *name* is the name of the array, and *length* is its length. For example, we can have:

```
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 .

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 can be given, but they must all agree on the type specification. This notation is also valid for multidimensional arrays. For example:

```
int(4) a[[3]][[5]];
```

indicates the element a(5, 3) of the array, 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 that ‘a[5]’ means that the array has five elements, whereas ‘a[[5]]’ implies that there are at least six.

l7.6.4 Arithmetic and Logical Expressions

All standard arithmetic and logical operators of C++ are allowed, including their precedence rules.

l7.6.5 Non-Parsable Variables

In order to accommodate complex syntactic constructs in which context information cannot be directly obtained from the bitstream but is the 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 can 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
```

l7.6.6 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.

Rule FC.1: Flow Control Using If-Then-Else

```
if (condition) {
    ...
} [ else if  (condition) {
    ...
}] [else {
    ...
}]
```

The following example illustrates the procedure.

```
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’. Another example is:

```
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 should 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) {
        …
}
```

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).

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).

To reduce the need for temporary variables (counters etc.), a ‘repeat’ statement is also defined.

Rule FC.6: Flow Control Using Repeat

```
repeat (parameter) {
        …
}
```

The block within the braces will be repeated as many times as indicated by *parameter*. Note that *parameter* is only examined once, at the beginning of the iterations.

l7.6.7 Bult-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.

l7.6.8 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). In particular, only variables declared in class scope are considered class member variables, and are thus available in objects of that particular type.

7.7 Object Content Information

7.7.1 Introduction

This subclause describes the syntax and semantics of Object Content Information (OCI) as well as the method by which this data is conveyed within an MPEG-4 session.

l7.7.2 Object Content Information (OCI) Data Stream

Object Content Information shall be conveyed in an Elementary Stream. Each media object that contains streaming data, i.e. that has an associated Object Descriptor, may have a separate OCI stream attached to it.

The Object Descriptor for such a media object shall contain at most one ES_descriptor pointing to an OCI stream, identified by the value 0x0A of the streamType field of this ES_descriptor. This solution allows the OCI stream to be optional.

Note: Since the Scene Description is itself conveyed in an Elementary Stream described by an Object Descriptor, it is especially possible to associate OCI to the scene as such.

OCI data is partitioned in Access Units as any media stream. Each OCI Access Unit corresponds to one OCI_Event, as described in the next subclause, and has an associated decoding time stamp (DTS) that identifies the point in time at which the OCI Access Unit becomes valid.

A pre-defined ALHeader configuration may be established for specific profiles in order to fit the OCI requirements in terms of synchronization.

l7.7.3 Object Content Information (OCI) Syntax and Semantics

This subclause specifies the syntax and semantics of MPEG-4 Object Content Information stream.

λ 7.7.3.1 OCI Decoder Configuration

The OCI Stream decoder needs to be configured to ensure proper decoding of the subsequent OCI data. The configuration data is specified in this subclause.

In the context of this Committee Draft of Internation Standard this configuration data shall be conveyed in the ES_descriptor declaring the OCI stream. It shall be put in the container for stream specific information (decoderConfigDescriptor.specificInfoByte[i]) as specified in Subclause .

7.7.3.1.1 Syntax

```
class OCI_Stream_Configuration {
      uint(8) version_label;
}
```

7.7.3.1.2 Semantics

version_label **-** This 8 bits field indicates the version of OCI specification used on the corresponding OCI data stream. For MPEG-4 version 1 only the value 0x01 is allowed; all the other values are reserved.

λ 7.7.3.2 OCI_Events

The MPEG-4 Object Content Information stream is based on the concept of **Events**. Each OCI_Event is conveyed as an OCI Access Unit that has an associated Decoding Time Stamp identifying the point in time at which this OCI_Event becomes valid.

7.7.3.2.1 Syntax

```
aligned(8) Class OCI_Event : bit(16) event_id {
      uint(16) length;
      uint(32) starting_time;
      uint(32) duration;
      uint(8) number_of_descriptors;
      for(i=0;i< number_of_descriptors;i++){
            OCI_Descriptor oci_descriptor;
      }
}
```

7.7.3.2.2 Semantics

event_id **-** This 16 bits field contains the identification number of the described event.

length - This 16 bits field gives the total length in bytes of the following descriptors.

starting_time - This 32 bits field contains the starting time referred to the starting time 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.

Example: 02: 36:45:89 is coded as “0x023645” concatenated and “01011001”

duration - This 32 bits field contains the starting time referred to the starting time 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.

Example: 02: 36:45:89 is coded as “0x023645” concatenated and “01011001”

number_of_descriptors **-** This 8 bits field gives the number of descriptors for the corresponding event.

λ 7.7.3.3 Descriptors

λ *7.7.3.3.1 OCI_Descriptor Class*

λ *7.7.3.3.1.1 Syntax*

```
Class OCI_Descriptor {
      uint label(8);
      uint length(8);
      switch (label) {
            case 0x01 : Content_Classification_Descriptor
                  content_classification_descriptor ;
            case 0x02 : Key_Wording_Descriptor key_wording_descriptor ;
            case 0x03 : Rating_Descriptor rating_descriptor ;
            case 0x04 : Language_Descriptor language_descriptor ;
            case 0x05 : Short_Textual_Descriptor short_textual_descriptor ;
            case 0x06 : Expanded_Textual_Descriptor
                  expanded_textual_descriptor ;
            case 0x07 : Name_Content_Creators_Descriptor
                  name_creators_descriptor ;
            case 0x08 : Date_Content_Creation_Descriptor
```

```
                  date_creation_descriptor ;
            case 0x09 : Name_OCI_Creators_Descriptor
                  name_OCI_creators_descriptor ;
            case 0x10 : Date_OCI_Creation_Descriptor
                  date_OCI_creation_descriptor ;
      }
}
```

l7.7.3.3.1.2 Semantics

label **-** This 8 bits field identifies each descriptor. The values 0x00 and 0x11 to 0xFF are reserved.

length - This 8 bits field specifies the total number of bytes of the data portion of the descriptor following the byte defining the value of this field.

7.7.3.3.2 Content classification descriptor

7.7.3.3.2.1 Syntax

```
Class Content_Classification_Descriptor{
      uint(8) number_of_classifications;
      for (i=0;i< number_of_classifications;i++){
            uint(16) classification_entity;
            uint(8) classification_table;
            for( i=0; i<N; i++){
                  uint(8) content_classifier_byte;
            }
      }
}
```

l7.7.3.3.2.2 Semantics

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

number_of_classifications **-** This 8 bits field indicates the number of content classification data sets to be provided for the corresponding event.

classification_entity **-** A 16 bits field indicating which is the content classification entity. The values of this field are to be attributed by a registration authority to be identified.

classification_table **-** This 8 bits field 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.

content_classifier_byte - This 8 bits field contains 1 byte of information using a non-default classification table.

7.7.3.3.3 Key wording descriptor

7.7.3.3.3.1 Syntax

```
Class Key_Wording_Descriptor {
      uint(24) language_code;
      uint(8) number_key_words;
      for ( i=0;i< number_key_words ;i++){
            uint(8) key_word_length;
            if (language_code == latin) then {
```

```
                  for (j=0;j< key_word_length;j++){
                        bit(8) key_word_char;
                  }
            } else {
                  for (j=0;j< key_word_length/2;j++){
                        bit(16) key_word_char ;
                  }
            }
      }
}
```

l7.7.3.3.3.2 Semantics

The key-wording descriptor allows the OCI creator/provider to indicate a set of key-words, characterizing the content. The choice of the key-words is completely free but each time the key-wording descriptor appears, all the key-words given are for the language indicated in language_code. This means that the key-wording class descriptor will appear, for a certain event, a number of times equal to the number of languages for which key-words are to be provided. This solution results from a compromise between efficiency and parsing complexity.

The content classification descriptor and the key-wording descriptor together provide a complete and useful solution to the problem of content characterization/description both for composited objects (scenes) as well as elementary objects.

language_code **-** This 24 bits field contains the ISO 639 [3] three character language code of the language of the following text fields. Remember that for languages that only use Latin characters, just one byte per character is needed in Unicode [5].

number_key-words **-** A 8 bits field indicating the number of key-words to be provided.

key-word_length **-** A 8 bits field specifying the length in bytes of each key-word.

key-word_char **-** 8 or 16 bits field; a string of 'key-word_char' fields specifies the key-word. Text information is coded using the Unicode character sets and methods [5].

7.7.3.3.4 Rating descriptor

7.7.3.3.4.1 Syntax

```
Class Rating_Descriptor{
      uint(8) number_of_ratings;
      for (i=0;i< number_of_ratings;i++){
            uint(16) rating_entity;
            uint(16) rating_criteria;
            for( i=0; i<N; i++){
                  uint(8) rating_byte;
            }
      }
}
```

l7.7.3.3.4.2 Semantics

This descriptor gives one or more ratings, originated from corresponding rating entities, valid for a specified country. The rating_entity field indicates the organization which is rating the content. The possible values have to be registered with a registration authority to be identified.

number_of_ratings **-** This 8 bits field indicates the number of ratings of content provided for the corresponding event.

rating_entity **-** A 16 bits field indicating which is the rating entity. The values of this field are to be attributed by a registration authority to be identified.

rating_criteria **-** This 16 bits field indicates which rating criteria is being used for the corresponding rating entity. The value 0x00 is reserved.

rating_byte - This 8 bits field contains 1 byte of rating information.

7.7.3.3.5 Language descriptor

7.7.3.3.5.1 Syntax

```
Class Language_Descriptor {
      uint(24) language_code;
}
```

l7.7.3.3.5.2 Semantics

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

language_code **-** This 24 bits field contains the ISO 639 [3] three character language code of the language of the following text fields. Remember that for languages that only use Latin characters, just one byte per character is needed in Unicode [5].

7.7.3.3.6 Short textual descriptor

7.7.3.3.6.1 Syntax

```
Class Short_Textual_Descriptor {
      uint(24) language_code;
      uint(8) name_length;
      if (language_code == latin) then {
            for (i=0;i<name_length;i++){
                  bit(8) name_char;
            }
            uint(8) text_length;
            for (i=0;i<text_length;i++){
                  bit(8) text_char;
            }
      } else {
            for (i=0;i<name_length/2;i++){
                  bit(16) name_char;
            }
            uint(8) text_length;
            for (i=0;i<text_length/2;i++){
                  bit(16) text_char;
            }
      }
}
```

l7.7.3.3.6.2 Semantics

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

language_code **-** This 24 bits field contains the ISO 639 [3] three character language code of the language of the following text fields. Remember that for languages that only use Latin characters, just one byte per character is needed in Unicode [5].

name_length **-** This 8 bits field specifies the length in bytes of the event name.

name_char **-** 8 or 16 bits field; a string of 'name_char' fields specifies the event name. Text information is coded using the Unicode character sets and methods [3].

text_length **-** This 8 bits field specifies the length in bytes of the following text describing the event.

text_char **-** 8 or 16 bits field; a string of 'text_char' fields specifies the text description for the event. Text information is coded using the Unicode character sets and methods [3].

7.7.3.3.7 Expanded textual descriptor

7.7.3.3.7.1 Syntax

```
Class Expanded_Textual_Descriptor{
      uint(24) language_code;
      uint(8) length_of_items
      for (i=0;i< length_of_items;i++){
            uint(8) item_description_length;
            if (language_code == latin) then {
                  for (i=0;i< item_description_length;i++){
                        bit(8) item_description_char;
                  }
            } else {
                  for (i=0;i< item_description_length /2;i++){
                        bit(16) item_description_char;
                  }
            }
            uint(8) item_length;
            if (language_code == latin) then {
                  for (i=0;i< item_length;i++){
                        bit(8) item_char;
                  }
            } else{
                  for (i=0;i< item_length /2;i++){
                        bit(16) item_char;
                  }
            }
      }
      uint(8) text_length;
      length_multiplier=0
      while( text_length == 255 ) {
            length_multiplier++;
            uint(8) text_length;
      }
      if (language_code == latin) then {
            for (i=0;i< length_multiplier*255+text_length;i++){
                  bit(8) text_char;
            }
      } else {
            for (i=0;i< (length_multiplier*255+text_length) /2;i++){
                  bit(16) text_char;
            }
      }
}
```

l7.7.3.3.7.2 Semantics

The expanded textual descriptor provides a detailed text description of an event, which may be used in addition to the short event descriptor. Beside non itemised expanded text also text information structured into two columns, one giving an item description field and the other the item text, may be provided. A typical 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.

language_code **-** This 24 bits field contains the ISO 639 [3] three character language code of the language of the following text fields. Remember that for languages that only use Latin characters, just one byte per character is needed in Unicode.

length_of_items **-** This 8 bits field specifies the length in bytes of the following items (itemised text).

item_description_length **-** This 8 bits field specifies the length in bytes of the item description.

item_description_char **-** 8 or 16 bits field; a string of 'item_description_char' fields specifies the item description. Text information is coded using the Unicode character sets and methods described in [5].

item_length **-** This 8 bits field specifies the length in bytes of the item text.

item_char **-** 8 or 16 bits field; a string of 'item_char' fields specifies the item text. Text information is coded using the Unicode character sets and methods described in [3].

text_length **-** This 8 bits field specifies the length in bytes of the non itemised expanded text. The value 255 works as an escape code, and it is followed by another 8 bits field which contains the length in bytes above 255. For lengths greater than 511 a third field is used, and so on.

text_char **-** 8 or 16 bits field; a string of 'text_char' fields specifies the non itemised expanded text. Text information is coded using the Unicode character sets and methods described in [5].

7.7.3.3.8 Name of content creators descriptor

7.7.3.3.8.1 Syntax

```
Class Name_Content_Creators_Descriptor {
      uint(8) number_content_creators;
      for (i=0;i< number_content_creators ;i++){
            uint(24) language_code;
            uint(8) content_creator_length;
            if (language_code == latin) then {
                  for (j=0;j< content_creator_length;j++){
                        bit(8) content_creator_char;
                  }
            } else {
                  for (j=0;j< content_creator_length/2;j++){
                        bit(16) content_creator_char;
                  }
            }
      }
}
```

l7.7.3.3.8.2 Semantics

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

number_content_creators **-** A 8 bits field indicating the number of content creators to be provided.

language_code **-** This 24 bits field contains the ISO 639 [3] three character language code of the language of the following text fields. Remember that for languages that only use Latin characters, just one byte per character is needed in Unicode.

content_creator_length **-** A 8 bits field specifying the length in bytes of each content creator name.

content_creator_char - 8 or 16 bits field; a string of 'content_creator_char' fields specifies the content creator name. Text information is coded using the Unicode character sets and methods [3].

7.7.3.3.9 Date of content creation descriptor

7.7.3.3.9.1 Syntax

```
Class Date_Content_Creation_Descriptor {
      uint(40) date_content_creation;
}
```

l7.7.3.3.9.2 Semantics

This descriptor identifies the date of the content creation.

date_content_creation **-** This 40 bits field 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 A). This field is coded as 16 bits giving the 16 LSBs of MJD followed by 24 bits coded as 6 digits in 4 bits Binary Coded Decimal (BCD). If the start time is undefined (e.g. for an event in a NVOD reference service) all bits of the field are set to "1".

7.7.3.3.10 Name of OCI creators descriptor

7.7.3.3.10.1 Syntax

```
Class Name_OCI_Creators_Descriptor {
      uint(8) number_OCI_creators;
      for ( i=0;i< number_OCI_creators ;i++){
            uint(24) language_code;
            uint(8) OCI_creator_length;
            if (language_code == latin) then {
                  for (j=0;j< OCI_creator_length;j++){
                        bit(8) OCI_creator_char;
                  }
             } else {
                  for (j=0;j< OCI_creator_length/2;j++){
                        bit(16) OCI_creator_char;
                  }
            }
      }
}
```

l7.7.3.3.10.2 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.

number_OCI_creators **-** A 8 bits field indicating the number of OCI creators.

language_code **-** This 24 bits field contains the ISO 639 [3] three character language code of the language of the following text fields. Remember that for languages that only use Latin characters, just one byte per character is needed in Unicode.

OCI_creator_length **-** A 8 bits field specifying the length in bytes of each OCI creator name.

OCI_creator_char - 8 or 16 bits field; a string of 'OCI_creator_char' fields specifies the OCI creator name. Text information is coded using the Unicode character sets and methods [5].

7.7.3.3.11 Date of OCI creation descriptor

7.7.3.3.11.1 Syntax

```
Class Date_OCI_Creation_Descriptor {
      uint(40) date_OCI_creation;
}
```

l7.7.3.3.11.2 Semantics

This descriptor identifies the date of the OCI description creation.

date_OCI_creation **-** This 40 bits field contains the OCI creation date for the OCI data corresponding to the event in question, in Universal Time, Co-ordinated (UTC) and Modified Julian Date (MJD) (see annex A). This field is coded as 16 bits giving the 16 LSBs of MJD followed by 24 bits coded as 6 digits in 4 bits Binary Coded Decimal (BCD). If the start time is undefined (e.g. for an event in a NVOD reference service) all bits of the field are set to "1".

l7.7.4

lAnnex: Conversion between time and date conventions

The types of conversion which may be required are summarized in the diagram below.

Figure 7-23: 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
x:	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' x 365,25)] / 30,6001 }

D = MJD - 14 956 - int (Y' x 365,25) - int (M' x 30,6001)
If M' = 14 or M' = 15, then K = 1; else K = 0
Y = Y' + K
M = M' - 1 - K x 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 + int [(Y - L) x 365,25] + int [(M + 1 + L x 12) x 30,6001]

c) To find WD from WJD

WD = [(MJD + 2) mod 7] + 1

d) To find MJD from WY, WN, WD

WJD = 15 012 + WD + 7 x { WN + int [(WY x 1 461 / 28) + 0,41] }

e) To find WY, WN from MJD

W = int [(MJD / 7) - 2 144,64]
WY = int [(W x 28 / 1 461) - 0,0079]
WN = W - int [(WY x 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.

7.8 Profiles

7.8.1 Scene Description Profiles.

This Subclause defines profiles of usage of Scene Description as specified in Subclause . Scene Description profiles are intended to allow the implementation of systems conforming to this specification, which implement a precisely specified subset of the specification.

Nodes are organized into Shared, 2D, 3D, Mixed. The profiles are defined in terms of these sets of nodes, when possible.

λ 7.8.1.1 2D profile

Applications supporting only 2D graphics capabilities have to conform to the “2D profile” as specified in this Subclause. This is intended for systems that implement low complexity graphics with 2D transformations and alpha blending.

The “2D profile” is defined by the nodes in Subclause (Shared) plus nodes in Subclause (2D) and nodes (Layer2D) and (Composite2DTexture).

λ 7.8.1.2 3D profile

Applications supporting the complete set of 3D capabilities have to conform to the “3D profile” as specified in this Subclause. This profile addresses systems that implement a full 3D graphics system, which requires much higher implementation complexity than the “2D profile”.

The “3D profile” is defined by the nodes in Subclause (Shared) plus nodes in Subclause (3D) and nodes (Layer3D), (Composite3DTexture) and (CompositeMap).

Note that the “3D profile” is a 3D only profile; the 2D nodes which are not Shared nodes are not included in the 3D profile.

λ 7.8.1.3 VRML profile

Applications claiming conformance with the “VRML profile” of this Committee Draft have to implement all and only the nodes specified by this International Standard, that are common to the specification of Draft International Standard 14472-1 (VRML) [2].

Nodes that are common to MPEG-4 Systems and VRML are clearly identified in the description of nodes of Subclause .

λ 7.8.1.4 Complete profile

Applications supporting the complete set of capabilities specified by this Part 1 of Committee Draft conform to the “Complete Profile”.

λ 7.8.1.5 Audio profile

Applications supporting all and only the audio related nodes as defined by this Subclause conform to the “Audio Profile”.

The “Audio profile” is defined by nodes 3.2.3 to 3.2.7 (Audio nodes), plus node 3.4.17 (Sound node).

7.9 Elementary Streams for Upstream Control Information

Media Objects may require upstream control information to allow for interactivity.

The content creator needs to advertise the availability of an upstream control stream by means of an ES_descriptor for this stream with the upstream flag set to one. This ES_descriptor shall be part of the same Object Descriptor that declares the downstream Elementary Stream for this Media Object. See Subclause for the specification of the Object Descriptor syntax.

An Elementary Stream flowing from receiver to transmitter is treated the same way as any downstream Elementary Stream. The ES data will be conveyed through the Elementary Stream Interface to the Access Unit Layer where the Access Unit data is packaged in AL-PDUs. The parameters for the AU Layer shall be selected as requested in the ES_descriptor that has advertised the availablility of the upstream control stream.

The AL-packetized Stream (APS) with the upstream control data is subsequently passed through the Stream Multiplex Interface to a delivery mechanism similar to the downstream APS's. The interface to this delivery mechanism may be embodied by the DMIF Application Interface as specified in Part 6 of this Committee Draft of International Standard.

Note: The content of upstream control streams is specified in the same part of this specification that defines the content of the downstream data for this Media Object. E.g., control streams for video compression algorithms are defined in 14496-2.

Annex A: Bibliography

Annex B: Time Base Reconstruction (Informative)

B.1 Time base reconstruction

In an MPEG-4 receiving terminal, the various timestamps are the means to synchronize events related to decoding, composition and the associated buffer management. However, these timestamps can only be used if the time base(s) of the transmitting terminal(s) are correctly reconstructed at the receiver. A normative method to do so is not specified. How this may conceptually be done is described below.

lB.1.1 Adjusting the receivers OTB

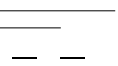
Each media object may be encoded by a media encoder with a different object time base (OTB). For each stream that conveys OCR, it is possible for the receiver to adjust a local OTB to the encoders' OTB. This is done by well-known PLL techniques. The notion of time for each object can therefore be recovered at the receiver side.

lB.1.2 Mapping Time Stamps to the STB

All OTBs in a MPEG-4 session may run at a different speed than the STB. 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. If time stamps are not mapped to the STB they are not useful to control the receiving terminals operation. Especially, over- or underflow of buffers may occur.

Note however, that the receiver terminals' System Time Base need not be locked to any of the Object Time Bases in an MPEG-4 session.

The composition time t_{SPT} in terms of STB of a Composition Unit can be calculated from the composition time stamp value t_{OPT} in terms of the relevant OTB, being transmitted by a linear transformation :



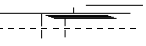
with:

composition time of a Composition Unit measured in units of

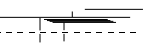
current time in the receiving terminals System Time Base

composition time of a Composition Unit measured in units of

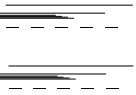
current time in the media object encoder's time base, conveyed by an OCR



value of receiving terminals STB when the first OCR time stamp of the media object is encountered.



value of the first OCR time stamp of the media object



The quotient is the scaling factor between the two time bases. In cases, where the clock speed and resolution of the media encoder and of the receiving terminal are nominally identical, this quotient is very near to 1. For avoiding long term rounding errors, the quotient 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.

A similar formula can be derived for decoding times by replacing composition 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 also to be mapped with the same mechanism.

With this mechanism it is possible to synchronize to several OTB so that a correct decoding and composition of composition units from several media objects becomes possible.

lB.1.3 Adjusting the STB to an OTB

In the special case that all media objects in a session are encoded by encoders using the same OTB, it is possible to lock the STB to this OTB by well known PLL techniques. In that case the mapping described in the previous subclause is not necessary.

lB.1.4 System Operation without Object Time Base

If a time base for an encoded media object is neither conveyed by OCR stamps nor derived from another Elementary Stream, time stamps can only be interpreted by the decoder in a flow controlled application (e.g. reading from file). Otherwise the decoder may only operate under the assumption that each Access Unit is to be decoded and presented as soon as it is received. In that case the Systems Decoder Model does not hold and cannot be used by the transmitting terminal to model the receivers' behavior.

In the case that a universal clock is available that can be shared between peer terminals, this may be used as a common time base and it may be possible to still use the Systems Decoder Model without explicit OCR transmission. This is currently not further specified in this document.

B.2 Temporal aliasing and audio resampling

The MPEG-4 receiver system is not required to synchronize decoding of AUs and composition of CUs. In other words, the STB does not have to be identical to the OTB of any media object. The number of decoded and actually presented (displayed) units per second may therefore differ. Hence, temporal aliasing may occur, resulting from composition units being presented multiple times or being skipped.

If audio signals are encoded on a system with an OTB different from the STB of the decoder, even nominally identical sampling rates of the audio samples will not match, 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 audiovisual scene: a walkthrough

The different steps to reconstruct a synchronized scene are as follows:

1. The time base for each object is recovered either from the OCR conveyed with this object or from another object present in the scene graph.
2. Object time stamps are mapped to the STB according to a suitable algorithm (e.g. the one detailed above),
3. Received Access Units are put in the Decoding Buffer
4. Each Access Unit is instantaneously decoded by the media object decoder at its implicit or explicit DTS and the resulting one or more Composition Units are put in the Composition Memory,

5. The compositor may access each CU between its CTS and the CTS of the subsequent CU.

Annex C: Embedding of MPEG-4 Streams in TransMux Instances (Informative)

The specification of this Committee Draft of International Standard terminates at the Stream Multiplex Interface, which is an interface at which packets of Elementary Stream data are conveyed. This assumes that a multitude of transport protocol stacks exists that is able to finally convey the packetized Elementary Stream data end-to-end.

The generic term TransMux is used to abstract all these potential protocol stacks, as shown in the following figure.

This informative annex presents a number of examples on how content complying to this Committee Draft of International Standard can be embedded in some instances of such a TransMux.

C.1 ISO/IEC 14496 content embedded in ISO/IEC 13818-1 Transport Stream

C.1.1 Introduction

This informative annex describes an encapsulation method for a stream complying to this Committee Draft of International Standard by an ISO/IEC 13818-1 Transport Stream [1]. This informative annex provides how to transmit the Object Descriptor, Stream Map Table, FM-PDU and AL-PDU that contains the BIFS, Audio and Visual Elementary Streams, in the 13818-1 Transport Stream. This informative annex also provides the method on how to specify the 14496 indication in the 13818-1 Program Specific Information.

lC.1.2 IS 14496 Stream Indication in Program Map Table

In a 13818-1 Transport Stream, the Elementary Streams are transmitted in the TS packets. The Packet ID (PID) is used to identify which data is transmitted in the payload of the TS packet. Some specific PID values are defined in Table 2-3 of ISO/IEC 13818-1. The value ‘0’ of the PID is used for the Program Association Table (PAT) specified in Subclause 2.4.4.3 of

ISO/IEC 13818-1. The PAT specifies the PID of the TS packet that conveys the Program Map Table (PMT) specified in Subclause 2.4.4.8 of ISO/IEC 13818-1. The PMT contains the information such as the stream_type and Elementary_PID of Elementary Streams within a program. The PMT consists of the TS_program_map_section specified in Table 2-28 of ISO/IEC 13818-1. The stream_type specified in Table 2-29 of ISO/IEC 13818-1 identifies the data type of the stream. The Elementary_PID identifies the PID of the TS packet that conveys the Elementary Stream in the program.

In this Committee Draft of International Standard, an Elementary Stream has an associated Object Descriptor. The Object Descriptor contains the information of the Elementary Stream such as the identifier ES_ID, streamType, etc. The Stream Map Table connects an ES_ID with a FlexMux channel. The Object Descriptor and Stream Map Table are transmitted in the payload of the TS packet. The PID of the TS packet that conveys the Object Descriptor and Stream Map Table shall be specified in the PMT. Therefore, it is proposed to assign an specific value of the stream_type for the stream that contains Object Descriptor and Stream Map Table. Proposed stream_type assignment is shown in Table Table C-13. The TS packet payload of the stream_type value 0x10 is the OD_SMT_section described in Subclause .

Table C-12 : Transport Stream Program Map Section of ISO/IEC 13818-1

Syntax	No. of bits	Mnemonic
TS_program_map_section () {		
table_id	8	Uimsbf
section_syntax_indicator	1	Bslbf
'0'	1	bslbf
Reserved	2	bslbf
section_length	12	uimsbf
program_number	16	uimsbf
Reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
Reserved	3	bslbf
PCR_PID	13	uimsbf
Reserved	4	bslbf
program_info_length	12	uimsbf
for (I=0; i<N; i++) {		
descriptor ()		
}		
for (I=0; i<N1; I++) {		
stream_type	8	uimsbf
Reserved	3	bslbf
Elemenatry_PID	13	uimsbf
Reserved	4	bslbf
ES_info_length	12	uimsbf
for (i=0; i<N2; i++) {		
Descriptor ()		
}		
}		
CRC_32	32	Rpchof
}		

Table C-13 : ISO/IEC 13818-1 Stream Type Assignment

Value	Description
0x00 - 0x0F	Defined in table 2-29 of ISO/IEC 13818-1

0x10	ISO/IEC 14496-1 MPEG-4
0x11 - 0x7F	ISO/IEC 13818-1 reserved
0x80 - 0xFF	User private

lC.1.3 Object Descriptor and Stream Map Table Encapsulation

The Object Descriptor and Stream Map Table should be retransmitted periodically in the Transport Stream. The OD_SMT_section shown in Table C-14 contains one Stream Map Table and one or more Object Descriptors of the Elementary Streams that multiplexed in a same PID. The OD_SMT_section has the same syntax and semantics as the private section specified in Subclause 2.4.4.10 of ISO/IEC 13818-1, excluding the section_syntax_indicator and table_id_extension. The section_syntax_indicator is fixed to '1' to use the table_id_extension, version_number, current_next_indicator, section_number, last_section_number and CRC_32. The lower 13 bits of the table_id_extension is the PID that conveys Elementary Streams described in the Object Descriptors in this OD_SMT_section. The upper 3 bits of the table_id_extension are reserved. Neither AL-PDU nor FM-PDU format applies to the Object Descriptor and Stream Map Table.

Table C-14 : OD SMT Section

Syntax	No. of bits	Mnemonic
OD_SMT_section () {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
private_indicator	1	bslbf
Reserved	2	bslbf
private_section_length	12	uimsbf
table_id_extension (reserved, PID)	16(3, 13)	uimsbf
Reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
Stream Map Table ()		
for (I=0; I<N; i++) {		
Object Descriptor ()		
}		
CRC_32	32	rpchof
}		

The Stream Map Table in the OD_SMT_section is shown in Table C-15. When the value of streamCount equals to zero, it specifies that the Stream Map Table contains one ES_ID and FlexMux does not apply to this Elementary Stream. When the value of the streamCount is larger than zero, it specifies that the Stream Map Table contains one or more ES_IDs, and the FlexMux applies to these Elementary Streams and their FlexMux channels are presented.

The ESinfoLength specifies the number of bytes of the descriptors associated Elementary Stream. The descriptor that immediately follows the ESinfoLength field may contain the association_tag field defined by DMIF when DMIF is used.

Table C-15 : Stream Map Table

Syntax	No. of bits	Mnemonic
Stream_Map_Table () {		
StreamCount	8	uimsbf

if (streamCount == 0) {		
ES_ID	16	uimsbf
ESinfoLength	8	uimsbf
for (i=0; i<ESinfoLength; i++) {		
descriptor ()		
}		
}		
else {		
for (i=0; i<streamCount; i++) {		
ES_ID	16	uimsbf
FMC	8	uimsbf
ESinfoLength	8	uimsbf
for (i=0; I<ESinfoLength; i++) {		
descriptor ()		
}		
}		
}		
}		

lC.1.4 Scene Description Stream Encapsulation

The BIFS stream may be retransmitted periodically in the Transport Stream. Therefore, it is appropriate to use the private section specified in section 2.4.4.10 of ISO/IEC 13818-1 to transmit the BIFS stream. The private section for the BIFS stream is shown in Table C-16. The section_syntax_indicator is fixed to '1' to use the table_id_extension, version_number, current_next_indicator, section_number, last_section_number and CRC_32. The AL-PDU format may apply to the BIFS stream. The AL-PDU header is configured appropriately in the ALConfigDescriptor that resides in the Object Descriptor. The AL-PDU that contains the BIFS stream is copied to the private_data_byte. One private section contains a complete AL-PDU. The PID of the TS packet that conveys this private section stream is derived from the lower 13 bits of the table_id_extension in the OD_SMT_section.

Table C-16 : Private section for the BIFS stream

Syntax	No. of bits	Mnemonic
private_section () {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
private_indicator	1	bslbf
Reserved	2	bslbf
private_section_length	12	uimsbf
table_id_extension	16	uimsbf
Reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
for (i=0; I<private_section_length - 9; i++)		
{		
private_data_byte	8	bslbf
}		
CRC_32	32	rpchof
}		

lC.1.5 Audio Visual Stream Encapsulation

The AL-PDU or FM-PDU format is used for Audio Visual Elementary Streams complying to this Committee Draft of International Standard. The AL-PDU or FM-PDU shall be aligned to the TS packet payload because neither PDU format provides sync-word to determine the first byte of the PDU. The AL-PDU or FM-PDU shall start from the first byte of the TS packet payload. The payload_unit_start_indicator specified in Subclause 2.4.3.2 of ISO/IEC 13818-1 is used to identify whether the AL or FM-PDU starts. The AL-PDU or FM-PDU starts from the TS packet when the payload_unit_start_indicator is set to '1'. While the AL-PDU or FM-PDU does not start from the TS packet when the payload_unit_start_indicator is set to '0'. A TS packet shall not contain more than one AL-PDUs when FM-PDU format is not used.

The PID of the TS packet that conveys Audio or Visual Elementary Stream complying to this Committee Draft of International Standard is derived from the lower 13 bits of the table_id_extension in the OD_SMT_section. The FlexMux channel is derived from the Stream Map Table in the OD_SMT_section when the FlexMux is used.

lC.1.6 Framing of AL-PDU and FM-PDU into TS packets

There exists the case that the AL-PDU or FM-PDU does not fit the size of TS packet, because the size of the TS packet is fixed to 188 bytes. Two different methods can be used for stuffing. The first method uses the adaptation field specified in Subclause 2.4.3.4 of ISO/IEC 13818-1. The second method uses the paddingFlag and paddingBits in the AL-PDU header.

λ C.1.6.1 Use of MPEG-2 TS Adaptation Field

The adaptation field specified in Subclause 2.4.3.4 of ISO/IEC 13818-1 is used when the size of AL-PDU or FM-PDU does not fit the size of the TS packet payload. The size of the adaptation field is identified by the adaptation_field_length field. The adaptation_field_length is the 8 bits field and specifies the length of adaptation field immediately following the adaptation_field_length. The size of the adaptation field can be calculated as follows :
in case of FM-PDU :

the_size_of_adaptation_field = 188 - 4 - MIN [the_size_of_FM_PDU, 184]

in case of AL-PDU :

the_size_of_adaptation_field = 188 - 4 - MIN [the_size_of_AL_PDU, 184]

λ C.1.6.2 Use of MPEG-4 PaddingFlag and PaddingBits

The paddingFlag and paddingBits are the fields that are used in the AL-PDU header to do the fitting of FM-PDUs into TS packets. These fields construct the FM-PDU that contains padding byte only. This type of FM-PDU can be used to adjust the size of TS packet to 188 bytes. An example is shown in Figure C-24. Note that this method cannot be used when the FlexMux does not applies to the Elementary Stream or when the size of the FM-PDU equals to 183 bytes. In that case, the method described in Subclause can be applied.

The packets are built as follows :

- TS_packet = TS_packet_header (4 bytes) + TS_packet_payload
- TS_packet_payload = FM_PDU
- FM_PDU = FM_PDU_header + FM_PDU_payload
- FM_PDU_payload = AL_PDU
- AL_PDU = AL_PDU_header + AL_PDU_payload
- AL_PDU_payload = 1 SegmentOfAU

- AU = Σ (SegmentOfAU)

Figure C-24 : An example of stuffing for the MPEG-2 TS packet

C.2 MPEG-4 content embedded in MPEG-2 DSM-CC Data Carousel

C.2.1 Scope

The scope of this experiment is to evaluate the use of the module multiplexing capabilities of the MPEG-2 DSM-CC (ISO/IEC 13818-6) Data Carousel as a flexible multiplexer when the TransMux channel is an ISO/IEC 13818-1 Transport Stream. The DSM-CC data carousel is designed so as to be compatible with the DMIF/Application Interfaces specified by this Committee Draft of International Standard.

lC.2.2 Introduction

The purpose of this informative annex is to describe a method based on the ISO/IEC 13818-1 DSM-CC Data Carousel to provide a FlexMux-like functionality in an ISO/IEC 13818-1 Transport Stream. The DSM-CC Data Carousel is designed to support the transmission of AL-PDUs in carousel modules. Each module is considered to be a Flexible Multiplexer channel. Advantages of the method are:

- Compatibility with existing ISO/IEC 13818 Hardware. The new services provided using this Committee Draft of International Standard are transported in DSM-CC sections which do not interfere with the current delivery of ISO/IEC 13818-1 Transport of Video and Audio streams. Consequently, current receiver hardware will not be affected by the new design.
- Use of current decoder hardware filtering capabilities to route Elementary Streams complying to this Committee Draft of International Standard to separate object buffers.
- Easy management of FlexMux channels. New channels can be added or removed easily by simply updating and changing the version of the carousel directory message.
- Support of several channels within one PID. As a result, maintaining the Stream Map Table becomes much easier as it does not depend on hard coded PID values.

• C.2.3 DSM-CC Data Carousel

The DSM-CC Data Carousel framework is specified in ISO/IEC 13818-6. It is built on the DSM-CC download protocol. In particular, data and control messages are periodically re-transmitted following a pre-defined periodicity.
The download server sends periodic download control messages which allow a client application to discover the data modules being transmitted and determine which, if any, of these modules are appropriate for the client.

The client typically retrieves a subset of the modules described in the control messages. The data modules are transmitted in blocks by means of download data messages. Acquisition of a module does not necessarily have to start at the first block in the module as download data messages feature a block numbering field which allows a client application to assemble the blocks in order.

When the Data Carousel is encapsulated in an ISO/IEC 13818-1 Transport Stream, special encapsulation rules apply to facilitate acquisition of the modules. The DSM-CC Broadcast Data Carousel framework allows many data modules to be transported within a single PID. Therefore small and large data modules can be bundled together within the same PID.

lC.2.4 General Concept

The DSM-CC Data Carousel protocol is extended to support unbounded modules. This is achieved by resetting the module size to **0**. Each module is then regarded as a FlexMux-like

channel as modules are transported in chunks by means of DownloadDataBlock() messages. It is proposed here to make each FlexMux AL-PDU coincide with the payload of a DownloadDataBlock() message. The FlexMux channel number is the identifier of the module and the length of each AL-PDU is the length of the DownloadDataBlock() message payload. The data channels are identified by DSM-CC sections for which **table_id** is equal to **0x3B** (DSM-CC download data messages). The signaling channel conveys the download control messages which are transported in DSM-CC sections with **table_id** equal to **0x3C** (DSM-CC download control messages).

The proposed design is fully compatible with Part 1 (Systems) and 5 (DMIF) of this Committee Draft of International Standard and in particular, is based on the concept of TransMuxAssociationTag which abstracts the location of an ISO/IEC 13818-1 Transport Stream and its associated FlexMux channels in the underlying broadcast network.

The location of the DSM-CC Data Carousel is announced in the Program Map Table by the inclusion of an association_tag_descriptor() structure defined by DSM-CC which binds the TransMux instance to a particular PID. The Stream Map Table is transmitted in the form of a directory service in DownloadInfoIndication() control messages. This table associates each Elementary Stream identifiers (**ES_Id**) with a module identifier (**moduleId**) and a **channelHandle** which is exposed at the DAI (DMIF/Application Interface).

The BIFS and Object Descriptors are transmitted in particular modules in the data carousel. The mechanism used for identifying the module conveying the First Object Descriptor is user private. The remaining modules are used to transmit the various Elementary Streams which provide the data necessary to compose the MPEG-4 scene.

The Figure below provides an overview of the system. The dotted lines represents the interface between the application employing this Committee Draft of International Standard and the underlying DMIF session. In the Figure below, the TransMuxAssociationTag abstracts the location of the ISO/IEC 13818-1 Transport Stream which is used to transport the DSM-CC Data Carousel. Each channel in the designated MPEG-2 Transport Stream is identified by **moduleId**. The module identifier for each module is published in the directory messages (DSM-CC downloadInfoIndication() messages) on the signaling channel and the various chunks of data modules are transported via DSM-CC downloadDataBlock() messages. Each of these messages is viewed as the payload of a FlexMux AL-PDU. The application requests to open a channel by means of a DA_ChannelAdd.Req() interface. The **ES_id** field is an input argument which identifies the desired Elementary Stream in the Object Descriptor. The underlying DMIF session replies by providing an handle to the multiplexed channel carrying the Elementary Stream. This handle is the **channelHandle**. Subsequently, the DMIF session informs the application of the arrival of new data by means of a Data.Indication() message. The client acknowledges receipt of the data through a Data.Response() interface.

lC.2.5 Design of Broadcast Applications

The purpose of this section is to review the most important transmission elements required to support applications employing this Committee Draft of International Standard in a broadcast environment, and to show how they can be mapped to the ISO/IEC 13818-1 DSM-CC Data Carousel.

λ C.2.5.1 Program Map Table

The Program Map Table (see Table 2-28 in Subclause 2.4.4.8 of ISO/IEC 13818-1) is acquired from PSI sections with **table_id** value **0x02.**

Table C-17: Transport Stream Program Map Section

Syntax	No. of bits	Mnemonic
TS_program_map_section(){		
table_id	**8**	**uimsbf**
section_syntax_indicator	**1**	**bslbf**
'0'	**1**	**bslbf**
Reserved	**2**	**bslbf**
section_length	**12**	**uimsbf**
program_number	**16**	**uimsbf**
Reserved	**2**	**bslbf**
version_number	**5**	**uimsbf**
current_next_indicator	**1**	**bslbf**
section_number	**8**	**uimsbf**
last_section_number	**8**	**uimsbf**
Reserved	**3**	**bslbf**
PCR_PID	**13**	**uimsbf**
Reserved	**4**	**bslbf**

program_info_length	**12**	**uimsbf**
for(I=0I<N;I++){		
Descriptor()		
}		
for(I=0;I<N1;I++){		
stream_type	**8**	**uimsbf**
Reserved	**3**	**bslbf**
Elementary_PID	**13**	**uimsbf**
Reserved	**4**	**bslbf**
ES_info_length	**12**	**uimsbf**
for(j=0;j<N2;j++){		
Descriptor()		
}		
}		
CRC_32	**32**	**rpchof**
}		

The data carousel resides in the stream identified by **stream_type** value **0x0B**. Table 9-4 in Subclause 9.2.3 of the ISO/IEC 13818-6 DSM-CC standard specifies that **stream_type** value **0x0B** indicates that DSM-CC section is present and conveys a DSM-CC User-to-Network message. The only permitted User-to-Network messages shall be the Download protocol messages defined in Subclause 7.3 of ISO/IEC 13818-6.

With the Broadcast Data Carousel are associated two descriptors in the descriptor loop following the **ES_info_length** field. The carousel identifier descriptor features a carousel identifier field **carousel_id** which can is used to identify the carousel at the service level. The **descriptor_tag** value for this descriptor is **0x13** (19). See Table 11-2 in Subclause 11.5.1 of ISO/IEC 13818-6 for a complete definition of this descriptor. The second descriptor is the association_tag_descriptor (see Table 11-3 in Subclause 11.5.2 of ISO/IEC 13818-6) which binds the **associationTag** with the TransMux channel which is in this case is the ISO/IEC 13818-1 Transport Stream channel identified by the field **elementary_PID**. More than one association_tag_descriptor can be included.

The value **PCR_PID** defines the PID of the Clock Reference common to all the multiplexed channels.

Table C-18: Association Tag Descriptor

Syntax	No. of bits	Mnemonic
association_tag_descriptor(){		
descriptor_tag	**8**	**uimsbf**
descriptor_length	**8**	**uimsbf**
association_tag	**16**	**uimsbf**
use	**16**	**uimsbf**
selector_byte_length	**8**	**uimsbf**
for(n=0; n<selector_byte_length		
{		
selector_byte	**8**	**uimsbf**
}		
for(I=0;I<N;I++){		
private_data_byte	**8**	**uimsbf**
}		
}		

The **descriptor_tag** value is **20** (**0x14**). Here the field **association_tag** conveys a copy of the **TransMuxAssociationTag** value.

λ C.2.5.2 FlexMux Descriptor

Multiplexed channels are aggregated into a TransMux channel by means of the MPEG2CarouselDescriptor specified by Part 5 (DMIF) of this Committee Draft of International Standard. This descriptor binds several multiplexed channels under a unique **associationTag** which here points to a TransMux channel. There a few possible solutions for transmitting this descriptor to the receiver. For example, it can be sent via a downloadServerInitiate() message or it can be transmitted at the bottom of a downloadInfoIndication() message. Both these messages are sent on the application signaling channel.

λ C.2.5.3 Application Signaling Channel and Data Channels

A PID with number **elementary_PID** and for which **stream_type** is equal to **0x0B** carries a DSM-CC Data Carousel. In this case the ISO/IEC 13818-1 stream is made of DSMCC_section() structures which convey either download control or download data messages. In applications utilizing this Committee Draft of International Standard, download control messages make up the application signaling channel and download data messages form a bundle of multiplexed channels. The definition of the structure DSMCC_section can be found in Table 9-2 of Subclause 9.2.2. in ISO/IEC 13818-6 specifications and is repeated here for convenience.

Table C-19: DSM-CC Section

Syntax	No. of bits	Mnemonic
DSMCC_section(){		
table_id	**8**	**uimsbf**
section_syntax_indicator	**1**	**bslbf**
private_indicator	**1**	**bslbf**
Reserved	**2**	**bslbf**
dsmcc_section_length	**12**	**uimsbf**
table_id_extension	**16**	**uimsbf**
Reserved	**2**	**bslbf**
version_number	**5**	**uimsbf**
current_next_indicator	**1**	**bslbf**
section_number	**8**	**uimsbf**
last_section_number	**8**	**uimsbf**
if(table_id == 0x3A){		
LLCSNAP()		
}		
else if(table_id == 0x3B){		
userNetworkMessage()		
}		
else if(table_id == 0x3C){		
downloadDataMessage()		
}		
else if(table_id == 0x3D){		
DSMCC_descriptor_list()		
}		
else if(table_id == 0x3E){		

for(I=0;I<dsmcc_section_length-9;I++){		
private_data_byte	**8**	**bslbf**
}		
}		
if(section_syntax_indicator == '0'){		
Checksum	**32**	**uimsbf**
}		
else {		
CRC_32	**32**	**rpchof**
}		

Table C-20: DSM-CC table_id Assignment

table_id	DSMCC Section Type
0x00 - 0x37	ITU-T Rec. H.222.0 | ISO/IEC 13818-1 defined
0x38 - 0x39	ISO/IEC 13818-6 reserved
0x3A	DSM-CC sections containing multiprotocol encapsulated data
0x3B	DSM-CC sections containing U-N Messages, except Download Data Messages.
0x3C	DSM-CC sections containing Download Data Messages
0x3D	DSM-CC sections containing Stream Descriptors
0x3E	DSM-CC sections containing private data
0x3F	ISO/IEC 13818-6 reserved
0x40 - 0xFE	User private
0xFF	forbidden

The application signaling channel can be recognized by the fact that DSMCC_sections have a **table_id** value equal to **0x3B**. DSMCC_section() structures with **table_id** equal to **0x3C** belong to the multiplexed channel bundle.

λ C.2.5.4 Stream Map Table

The Stream Map Table links Elementary Stream Identifiers (**ES_id**) used by BIFS and the application to the **channelHandle** value that DMIF uses to refer to the stream.

In the case of the DSM-CC Data Carousel, the Stream Map Table is conveyed by means of the downloadInfoIndication() message in the application signaling channel.

In the PID identified by **elementary_PID** and for which **stream_type** is equal to **0x0B**, the DSM-CC sections with **table_id** value equal to **0x3B** convey DownloadInfoIndication() messages which provide a directory service announcing the data modules available in the carousel._The DownloadInfoIndication() message is classified as a download control message (see Subclause 7.3.2 of ISO/IEC 13818-6). Therefore, this message is preceded by a dsmccMessageHeader() message header (see Table 2-1 in Clause 2 of ISO/IEC-13818-6)

Table C-21: DSM-CC Message Header

Syntax	Num. Of Bits
DsmccMessageHeader(){	

ProtocolDiscriminator	**8**
DsmccType	**8**
MessageId	**16**
TransactionId	**32**
Reserved	**8**
AdaptationLength	**8**
MessageLength	**16**
if(adaptationLength > 0){	
DsmccAdaptationHeader()	
}	
}	

- protocolDiscriminator is set to **0x11** as specified in Clause 2 of ISO/IEC 13818-6
- **dsmccType** is set to **0x03** to indicate that a download message follows (see Table 2-2 in chapter 2 of ISO/IEC 13818-6).
- **messageId** is set to **0x1002** as specified in Table 7-4 in Subclause 7.3 of ISO/IEC 13818-6
- the two most significant bits of **transaction_id** are set to **0x01** to indicate that this field is assigned by the server (see Table 2-3 in Clause 2 of ISO/IEC 13818-6).
-

- **Table C-22: Adaptation Header**

Syntax	**Num of bits**
dsmccAdaptationHeader(){	
AdaptationType	**8**
for(I=0;I<adaptationLength-1;I++){	
AdaptationDataByte	**8**
}	
}	

Table C-23: DSM-CC Adaptation Types

Adaptation Type	Description
0x00	ISO/IEC 13818-6 reserved
0x01	DSM-CC Conditional Access adaptation format.
0x02	DSM-CC User ID adaptation format
0x03-0x7F	ISO/IEC 13818-6 Reserved.
0x80-0xFF	User defined adaptation type

Table C-24: DownloadInfoIndication Message

Syntax	Num. Of Bits
DownloadInfoIndication(){	
DsmccMessageHeader()	
DownloadId	**32**
BlockSize	**16**
WindowSize	**8**
AckPeriod	**8**
TCDownloadWindow	**32**
TCDownloadScenario	**32**
CompatibilityDescriptor()	
NumberOfModules	**16**

for(I=0;I<numberOfModules;I++){	
ModuleId	**16**
ModuleSize	**32**
ModuleVersion	**8**
ModuleInfoLength	**8**
for(j=0;j<moduleInfoLength;j++){	
moduleInfoByte	**8**
}	
}	
PrivateDataLength	**16**
for(I=0;I<privateDataLength;I++){	
PrivateDataByte	**8**
}	
}	

- **downloadId** conveys a copy of **carousel_id**
- **windowSize** is set to **0x00** (as specified in Subclause 7.3.2 of ISO/IEC 13818-6)
- **ackPeriod** is set to **0x00** (as specified in Subclause 7.3.2 of ISO/IEC 13818-6)
- **tcDownloadWindow** is set to **0x00** (as specified in Subclause 7.3.2 of ISO/IEC 13818-6)

The field **moduleId** featured in the downloadInfoIndication() message above is the data channel number. The first 2 bytes of the **moduleInfoByte** field convey the **ES_id** field which is used by BIFS to make reference to the Elementary Stream. The next 2 bytes of the **moduleInfoByte** field convey a copy of **channelHandle** which is exposed at the DAI interface. The mechanism for identifying Object Descriptor and BIFS streams is private.

λ C.2.5.5 TransMux Channel

DSM-CC sections with **table_id** value equal to **0x3C** convey downloadDataBlock() messages which include each a chunk of a data module. The data pipe ensuring the transport of these DSMCC_section() structures can therefore been seen as the TransMux channel. This channel is identified by the **associationTag** in the PMT. The protection layer provided by the TransMux is error detection (**checksum** is present). Consequently, the field **section_syntax_indicator** in these DSMCC sections shall always be set to **1**.

λ C.2.5.6 FlexMux Channel

Any module can be acquired by filtering DSM-CC sections having **table_id** equal to **0x3C** and **table_id_extension** equal to the module identifier, **moduleId**. These values correspond to DSM-CC sections conveying DownloadDataBlock() messages. These messages convey data blocks which are chunks of a target module identified by **moduleId**. A FlexMux channel is the set of DSM-CC sections conveying the DownloadDataBlock() messages corresponding to one **moduleId**. The FlexMux channel number is the module identifier, **moduleId**. The DownloadDataBlock() message is classified as a download data message. See Table 7-7 in Subclause 7.3 of ISO/IEC 13818-6. Therefore, it is preceded by a dsmccDownloadDataHeader() message header (see Table 7-3 in Subclause 7.2.2.1 of ISO/IEC 13818-6).

The FlexMux PDU can be considered to start in each DSMCC_section() from the field **dsmcc_section_length** field included up to last byte before the 4 last bytes used for error protection not included (**CRC_32** or **checksum** field). The equivalent of the **index** and **length** field in the regular MPEG-4 FlexMux are the DSMCC_section fields **table_id_extension** and **dsmcc_section_length**, respectively**.** The field **table_id** provides a distinction between the signaling channel and the data channel. The remaining DSMCC_section fields can be viewed as part of the protection layer provided by the

TransMux. According to the encapsulation rules specified in Clause 9 of ISO/IEC 13818-6, the field **table_id_extension** conveys a copy of **moduleId**, or in different terms, the FlexMux channel number. The MPEG-4 AL-PDU Header is placed in the dsmccDownloadDataHeader() of the DownloadDataHeader() and the AL-PDU payload is the payload of DownloadDataBlock() message.

Table C-25: DSM-CC Download Data Header

Syntax	Num. Of Bits
DsmccDownloadDataHeader(){	
ProtocolDiscriminator	**8**
DsmccType	**8**
MessageId	**16**
downloadId	**32**
reserved	**8**
adaptationLength	**8**
messageLength	**16**
if(adaptationLength > 0){	
dsmccAdaptationHeader()	
}	
}	

where

- **protocolDiscriminator** value is set to **0x11** as specified in Clause 2 of ISO/IEC 13818-6
- **dsmccType** is set to **0x03** as specified in Table 2-2 in Clause 2 of ISO/IEC 13818-6
- **messageId** is set to **0x1003** as specified in Table 7-4 in Subclause 7.3 of ISO/IEC 13818-6 if the message is a downloadDataBlock() message.
- **download_id** conveys a copy of **carousel_id**

The dsmccAdaptationHeader() is defined in Table 2-4 in Subclause 2.1 of ISO/IEC 13818-6.It is proposed to reserve the **adaptationType** value **0x03** to signal the presence of the AL-PDU Header in the adaptation field. The following table shows the new **adaptationType** values.

Table C-26: DSM-CC Adaptation Types

Adaptation Type	Description
0x00	ISO/IEC 13818-6 reserved
0x01	DSM-CC Conditional Access adaptation format.
0x02	DSM-CC User ID adaptation format
0x03	*MPEG-4 adaptation format*
0x04-0x7F	ISO/IEC 13818-6 Reserved.
0x80-0xFF	User defined adaptation type

The FlexMux AL-PDU payload is the payload of the downloadDataBlock() message which includes a chunk of the data module.

Table C-27: DSM-CC DownloadDataBlock() Message

Syntax	Num of bits
DownloadDataBlock(){	
DsmccDownloadDataHeader()	
ModuleId	**16**

ModuleVersion	**8**
Reserved	**8**
BlockNumber	**16**
for(I=0;I<N;I++){	
BlockDataByte	**8**
}	
}	

λ C.2.5.7 Payload

A data module can be reconstructed by concatenating the received data blocks according to the **blockNumber** field received in each FlexMux AL-PDU payload. In streaming mode, the change in module content is captured by the **moduleVersion** value which is incremented by one each time the module is loaded with new data. The field **moduleVersion** is reset once it has reached its allowed maximum value. A one-to-one mapping between **moduleVersion** (8 bits) in the DownloadDataBlock() message and the **version_number** field in DSMCC_section (5 bits) shall be provided by always setting the 3 most significant bits of **moduleVersion** to **0**. Consequently, the allowed maximum value for **moduleVersion** is **0x1F** (**31**). Likewise, the 8 most significant bits of the **blockNumber** field (16 bits) in a DownloadDataBlock() message shall not be used to provide a one-to-one mapping with the **section_number** field (8 bits) in DSMCC_section().

C.3 MPEG-4 content embedded in a Single FlexMux Stream

This subclause gives an example of a minimal configuration of a system utilizing this Committee Draft of International Standard that is applicable if an application constrains a session to a single peer-to-peer interactive connection or to the access to a single stored data stream. This configuration is not intended for random access nor is it resilient to errors in the transmitted or stored data stream. Despite these limitation, this configuration has some applications, e.g., storage of data complying to this Committee Draft of International Standard on disk for interchange, non-real-time transmission and even real-time transmission and playback, as long as the missing random access is tolerable. For this minimal configuration, the tools defined within this specification already constitute a near complete Systems layer.

This minimal configuration consists of a FlexMux Stream. Any number of up to 256 Elementary Streams can be multiplexed into a Single FlexMux Stream (SFS), that offers 256 transport channels, termed FlexMux Channels.

In addition to the raw FlexMux Stream that constitutes a fully compliant FlexMux Stream according to this specification, it is necessary to specify conventions how to know what is the content of this Single FlexMux Stream. Such conventions are specified in this subclause.

lC.3.1 Initial Object Descriptor

MPEG-4 content in a Single FlexMux Stream shall always start with the Initial Object Descriptor for the content carried in this SFS. The initial Object Descriptor is neither packaged in an OD_Update command nor in an AL-PDU but conveyed in its raw format.

The initial Object Descriptor follows the constraints specified in if the SFS contains more than one Elementary Stream. If the SFS contains only one Elementary Stream the Initial Object Descriptor shall contain only one ES_descriptor describing the properties of this Elementary Stream.

lC.3.2 Stream Map Table

The initial Object Descriptor mechanism allows to store Elementary Streams with arbitrary ES_IDs in the Single FlexMux Stream. In order to associate ES_IDs to the FlexMux Channels that carry the corresponding Elementary Streams, a Stream Map Table (SMT) is required.

The Stream Map Table shall immediately follow the initial Object Descriptor in the SFS. The syntax and semantics of the SMT is specified here.

λ C.3.2.1 Syntax

```
class StreamMapTable {
      uint(8) streamCount;
      for (i=0; i<streamCount; i++) {
            uint(16) ES_Id;
            uint(8) FlexMuxChannel;
      }
}
```

λ C.3.2.2 Semantics

streamCount is the number of streams for which the stream association is conveyed in this table.

ES_Id is the concatenation of ObjectDescriptorID and ES_number as defined in Subclause .

FlexMuxChannel is the FlexMux Channel number within the current FlexMux Stream.

lC.3.3 Single FlexMux Stream Payload

The remainder of the Single FlexMux Stream, following the initial Object Descriptor and the Stream Map Table shall consist of FlexMux-PDUs that encapsulate the AL-PDUs of one or more AL-packetized Streams.

The configuration of the AL-PDU Headers of the individual AL-packetized Streams is known from their related ES_descriptors that are conveyed either in the initial Object Descriptor or in additional Object Descriptors that are conveyed in an ObjectDescriptorStream that has been established by means of the initial Object Descriptor.

Annex D: View Dependent Object Scalability (Normative)

D.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 gridded 3D mesh such as terrain). Corresponding tools from the Visual and Systems parts of this specification are used in conjunction with downstream and upstream channels of a decoding terminal. The combined capabilities provide the means for an encoder to react to a stream of viewpoint information received from a terminal. The encoder transmits a series of coded textures optimized for the viewing conditions which can be applied in 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 encoder. 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, along 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 decoding 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 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 encoding database server to the decoder. 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 decoder 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.

D.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 receiver.

Subclause is concerned with the bitstream syntax for a View Dependent Object which initializes the session at the upstream data decoder. Subclause is concerned with the View Dependent Object Layer and contains the viewpoint information that is to be communicated back to the coder.

lD.2.1 View Dependent Object

ViewDependentObject() {	No. of bits	Mnemonic
view_dep_object_start_code	32	bslbf
field_of_view	16	uimsbf

marker_bit	1	bslbf
xsize_of_rendering_window	16	uimsbf
marker_bit	1	bslbf
ysize_of_rendering_window	16	uimlbf
marker_bit	1	bslbf
do {		
ViewDependentObjectLayer()		
} while(nextbits_bytealigned()== view_dep_object_layer_start_code)		
next_start_code()		
}		

lD.2.2 View Dependent Object Layer

ViewDependentObjectLayer() {	**No. of bits**	**Mnemonic**
view_dep_object_layer_start_code	32	bslbf
xpos1	16	uimsbf
marker_bit	1	bslbf
xpos2	16	uimsbf
marker_bit	1	bslbf
ypos1	16	uimsbf
marker_bit	1	bslbf
ypos2	16	uimsbf
marker_bit	1	bslbf
zpos1	16	uimsbf
marker_bit	1	bslbf
zpos2	16	uimsbf
marker_bit	1	bslbf
xaim1	16	uimsbf
marker_bit	1	bslbf
xaim2	16	uimsbf
marker_bit	1	bslbf
yaim1	16	uimsbf
marker_bit	1	bslbf
yaim2	16	uimsbf
marker_bit	1	bslbf
zaim1	16	uimsbf
marker_bit	1	bslbf
zaim2	16	uimsbf
}		

lD.3 Bitstream Semantics

This subclause details the semantic meaning of the various fields in the bitstream as specified previously in the Subclause .

lD.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 illegal 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 unsigned integer that specifies the vertical size of the rendering window.

lD.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 bit 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 16 bit word of the 32 bit integer xpos.

zpos1 – This is a 16 bit codeword which forms the lower 16 bit of the 32 bit integer xpos. The integer zpos can be computed as follows: zpos = zpos1 + (zpos2 << 16).

zpos2 – This is a 16 bit codeword which forms the upper 16 bit of the 32 bit integer xpos.

xaim1 – This is a 16 bit codeword which forms the lower 16 bit 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 bit of the 32 bit integer xaim.

yaim1 – This is a 16 bit codeword which forms the lower 16 bit 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 bit of the 32 bit integer yaim.

zaim1 – This is a 16 bit codeword which forms the lower 16 bit 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 bit of the 32 bit integer zaim.

D.4 Decoding Process of a View-Dependent Object

D.4.1 Introduction

This subclause explains the process for decoding the texture data using the VDS parameters. In order to determine which of the DCT coefficients are to be updated, a “mask”, which is a simple binary image, shall be computed. The first step is to determine the viewing parameters obtained from the texture-mesh composition procedure that drives 3D rendering in the user's decoding terminal. These parameters are used to construct the DCT mask corresponding to the first viewpoint of the session (VD mask). This mask is then updated with differential masks, built with the new viewing parameters that allow the texture image to be streamed. The bitstream syntax for view parameters and incremental transmission of DCT coefficients is given elsewhere in the Visual and Systems parts of this standard.

lD.4.2 General Decoding Scheme

The following subclauses outline the overall process for the decoder and encoder to accomplish the VDS functionalities.

λ D.4.2.1 View-dependent parameters computation

The VDS parameters (α and β angles, distance d for each cell) shall be computed using the geometrical parameters (Mesh, Viewpoint, Aimpoint, Rendering window). These parameters shall be computed for each cell of the grid mesh.

λ D.4.2.2 VD mask computation

For each 8x8 block of texture elements within a 3D mesh cell, the locations of the visible DCT coefficients inside the DCT block shall be computed using α and β angles, and the distance d defined for each cell relative to the viewpoint. The result shall be put in a binary mask image.

λ D.4.2.3 Differential mask computation

With the knowledge of which DCT coefficients have already been received (Binary mask buffered image) and which DCT coefficients are necessary for the current viewing conditions (Binary VD mask image), the new DCT coefficients shall be determined (Binary Differential mask image) as described in Subclause of this specification.

λ D.4.2.4 DCT coefficients decoding

The Video Intra bitstream, in the downstream channel, shall be decoded by the receiver terminal to obtain the DCT coefficients (DCT image). The decoding procedure is described in Subclause of this specification.

λ D.4.2.5 Texture update

The current DCT buffer in the receiver terminal shall be updated according to the Differential mask, using the received DCT image. The new received DCT coefficients shall be added to the buffered DCT image.

λ D.4.2.6 IDCT

The Inverse DCT of the updated DCT image shall computed, as specified in Subclause of this specification, to obtain the final texture.

λ D.4.2.7 Rendering

The texture is mapped onto the 3D mesh and the rendering of the scene is done, taking into account the mesh and the viewing conditions. This part of the procedure is outside the scope of this specification.

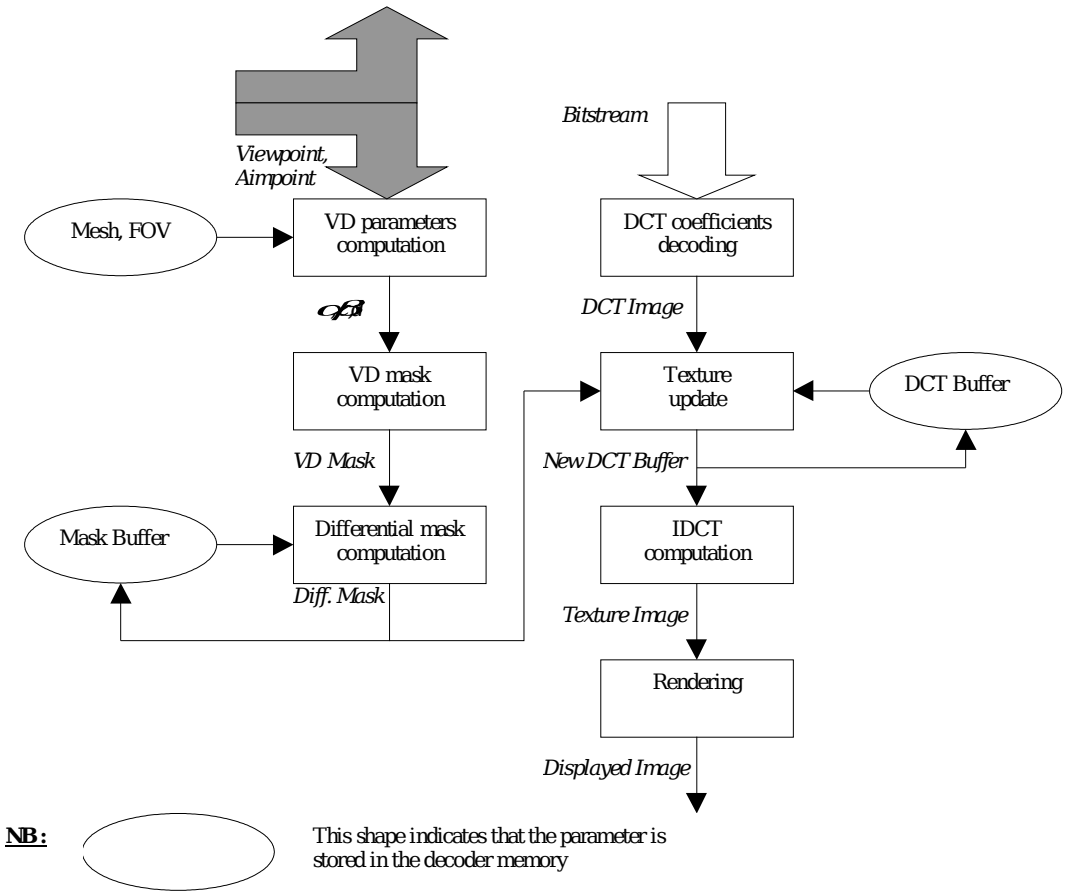
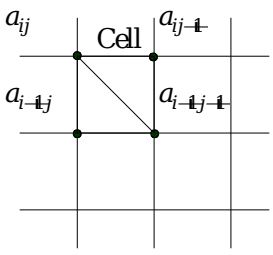


Figure D-25: General Decoding Scheme of a View-Dependent Object

lD.4.3 Computation of the View-Dependent Scalability parameters

The VDS parameters shall be computed for each cell of the grid mesh. The mesh may either be a quadrilateral or a triangular mesh. The number of cells in each dimension shall be equal to the texture size divided by 8.



λ D.4.3.1 Distance criterion:

$$Rd = \frac{1}{u}$$

u is the distance between viewpoint and Cell center: $u = \|v - c\|$ with $\vec{c} = \frac{1}{4}(\vec{a}_{ij} + \vec{a}_{i+1j} + \vec{a}_{ij+1} + \vec{a}_{i+1j+1})$ and v is the viewpoint vector.

λ D.4.3.2 Rendering criterion:

$$R_r = \frac{p}{q}$$

p is the distance between viewpoint and projection plane normalized to window width. p may be computed using:

$$Ra = \cos(\alpha)$$
$$Rb = \cos(\beta)$$

where FOV is the Field of View specified in radians, and q = <TextureWidth>/<WindowWidth> where the texture width is the width of the full texture (1024 for instance) and the WindowWidth is the width of the rendering window.

λ D.4.3.3 Orientation criteria:

$$Ra = \cos(\alpha)$$
$$Rb = \cos(\beta)$$

The angle between the aiming direction and the normal of the current cell center shall be projected into two planes. These two planes are spans of normal vector $\vec{n}$ of the cell and the cell edges in x and y directions, respectively. Then the angles (α, β) between projected vectors and the normal $\vec{n}$ shall be calculated, respectively.

g^y

The angle $?$ is specified as the projection of the angle between $\vec{n}$, the normal of the quad cell, and $\vec{u}$, the aiming direction, onto the plane Π_x that passes through $\vec{g}_x$ and is parallel to $\vec{n}$. Similarly, the angle $?$ is specified as the projection of the same angle onto the plane Π_y that passes through $\vec{g}_y$ and its parallel to $\vec{n}$.

Quad Cell

g^x

$\vec{n}$

This is illustrated in Figure D-26 $\vec{u}$

ViewPoint

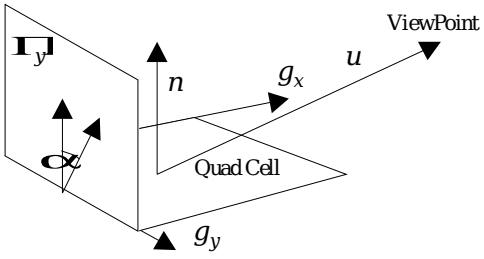


Figure D-26: Definition of α and β angles

λ D.4.3.4 Cropping criterion:

Cells that are out of the field of view shall not be transmitted/received: that is, at least one of the 4 vertices which define the cell should all be inside the horizontal and vertical Field Of View (FOV).

The horizontal FOV shall be deduced from the vertical FOV using the screen geometry. The vertical FOV is equal to the FOV. Then the following shall be calculated

where *w* and *h* are the width and height, respectively, of the rendered image.

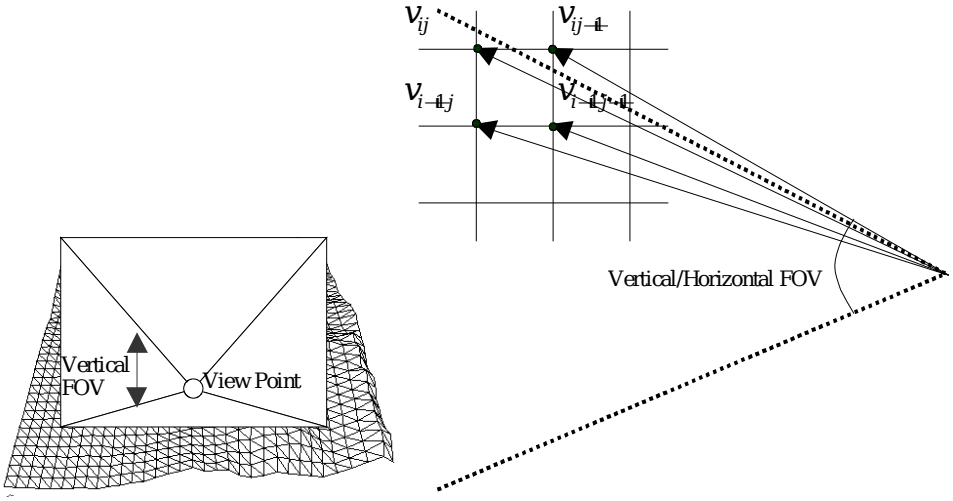


Figure D-27: Definition of Out of Field of View cells

lD.4.4 VD mask computation

The VD mask is a binary image of the same size as the texture image. Each value in the mask shall indicate if the corresponding DCT coefficient is needed (1) or not (0), given the VDS parameters.

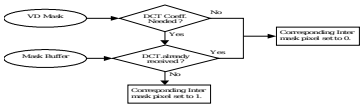
For each cell, the following rules shall be applied to fill the corresponding 8x8 block of the VD mask:

0**Use of cropping criterion:** If all the vertices of the cell are out of the field of view, the corresponding 8x8 block of the mask image shall be set to 0.

1

2**Use of rendering, distance, tilting and rotation criteria:** For each 8x8 block of the mask (corresponding to a quad cell), the 4 criteria mentioned above shall be computed. Two values of the rotation and tilting criteria shall be obtained for a quad cell, but only the higher value of each criterion shall be kept.

Two thresholds, T_X and T_Y, shall be calculated as the product of the three VDS parameters *Rr, Rd, Rb,* and *Rr, Rd, Ra*, respectively, and the value 8. The results shall be bounded to 8. This procedure may be indicated symbolically as follows



The flag (i,j) of the 8x8 block corresponding to the current cell shall be set to 1 if $i < T_X$ and $j < T_Y$. The flag shall be set to 0 in all other cases, as illustrated in the figure below.

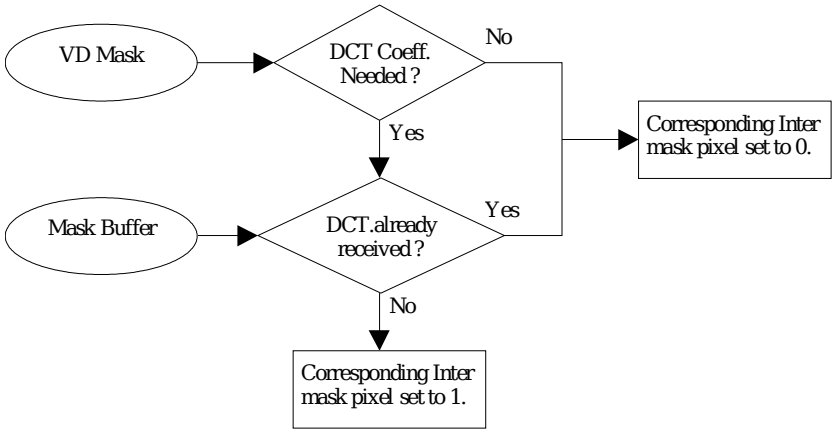


Figure D-28: VD mask of an 8x8 block using VD parameters

lD.4.5 Differential mask computation

Once the first image has been received using the previously described filter, less data is necessary to update the texture data for the following frames. (assuming there is a correlation between viewpoint positions). Since this computation is exactly the same for each flag of each cell, it shall be performed directly on the full mask images and not on a cell by cell basis.

If the coefficient has not been already transmitted (buffer mask set to 0) and is needed according to VDS visibility criteria (VD mask set to 1), then the corresponding pixel of the differential mask shall be set to 1. This implies that the texture shall be updated, according to the procedure described in the Subclause of this specification.

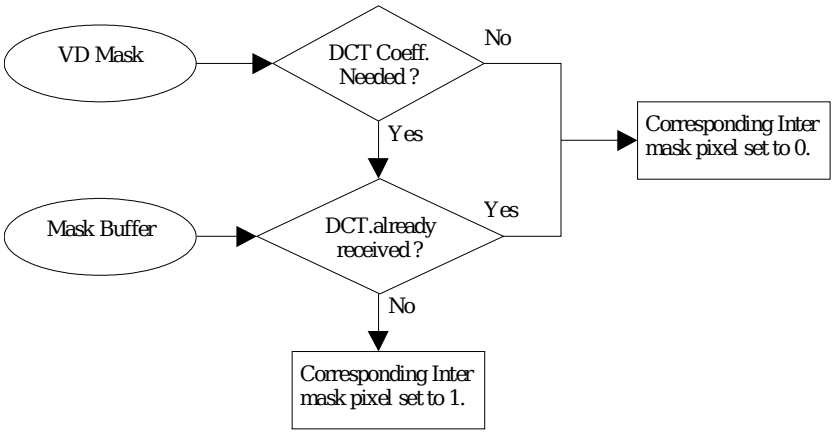


Figure D-29: Differential mask computation scheme

lD.4.6 DCT coefficients decoding

The DCT coefficients shall be decoded using the Video Intra mode, Separated Texture/Motion mode as described in Subclause 7.3 of Part 2 of this Committee Draft of International Standard.

lD.4.7 Texture update

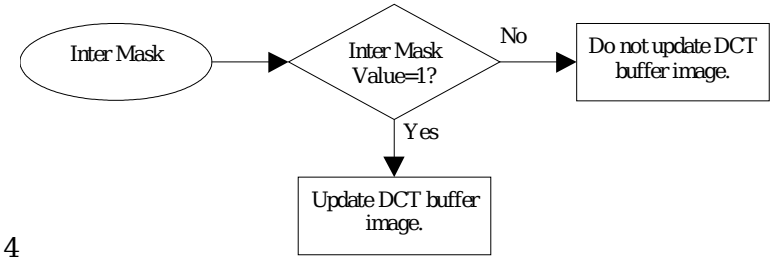
The Differential mask image shall be used to select which DCT coefficients of the buffered texture should be updated using the decoded DCT coefficients.

0Y component

1If the Differential mask is set to 0, the corresponding DCT value of the buffer shall be left unchanged, otherwise the value shall be updated with the previously decoded DCT coefficient.

2

3



4

5**Figure D-30: Texture update scheme**

6U and V component

The texture is coded in 4:2:0 format, as specified in Subclause 6.1.1.6 of Part 2 of this Committee Draft of International Standard, which shall imply that for each chrominance DCT coefficient, 4 Differential mask flags shall be available. The chrominance coefficients shall be received/transmitted if at least 1 of these 4 flags is set to 1.

lD.4.8 IDCT

The IDCT and de-quantization shall be performed using the same process as in the Video Intra mode as described in Subclause 7.3 of Part 2 of this Committee Draft of International Standard.