1. Introduction

This document provides a list of comments on the MPEG-4 Systems CD (ISO/IEC 14496-1) that have been identified by the author and other members of the WG11 community. The comments are organized as “Major Technical,” “Technical,” and “Editorial.” Within each category, organization is based on the Clause number of the Committee Draft the comment refers to.

2. Major Technical

2.1 Clause 7.2.2.7.1 Nodes
The format of the data accessed via a URL must be clarified. For BIFS URLs, the text does not provide any details about their format. It should be explicitly stated that such data can be in any format that MPEG-4 supports, including raw MPEG-4 media objects (without AL encapsulation, e.g., raw still images or visual/audio elementary streams), the forthcoming MPEG-4 Intermedia Format, etc.; the appropriate format is determined by the MIME type. Appropriate MPEG-4 types should be registered with IANA by WG11.

2.2 Clause 7.2.2.7.1 Nodes
For object descriptor URLs (specified via the ES descriptor structure of object descriptors), the specification indicates that access is performed to an AL-packetized stream with, or without, a preceding object descriptor. Using regular HTTP access, however, implies use of the TCP protocol. This protocol is stream-oriented, and hence does not provide framing capabilities for PDUs. As a result, the AL structure as defined in the 14496-1 specification cannot be used as is, since it assumes that the underlying transport layer will provide the necessary framing information. Note that this problem is not limited to TCP, but exists also for any transport layer that follows stream (rather than message) semantics. Clause 7.3.4.3 (Usage of URLs in Object Descriptors) fails to address these issues.
The USNB recommends the definition of an AL-packetized stream, called *Framed Adaptation Layer Stream* (FALS), in which framing is explicitly provided via a length indicator. The following is the exact syntax for such FALS streams.

```c
// MPEG-4 FALS
class FALS {

    ALConfigDescriptor config;

    int (32) size;
    while (size > 0){
        aligned (8) AL_PDU_Header alPDUHeader(od.esd.alConfigDescr);
        char (8) AL_PDU_data[size - lengthof (alPDUHeader)];
        int (32) size;
    }
}
```

The presence of the ALConfigDescriptor is required in order to set the options used in the AL headers.

With this definition of a FALS, it should be explicitly stated that “Object descriptor URLs can now point to any format supported by MPEG-4” (similarly to BIFS URLs). The ‘remoteODflag’ should be removed from Clause 7.3.3.2 (ES Descriptor) and the subclauses it contains.

The FALS format should also be identified in Figure 0-1 (Processing Stages in an audiovisual terminal) which identifies the layering of the various 14496-1 systems tools. In particular, it is an optional tool that is positioned below the AccessUnit Layer. This optional tools should be identified as the “FALS Layer”, and the interface it exposes “FALS Interface”. Since this layer is optional, there should be a straight line connecting the AU Layer directly to the FlexMux Layer.

### 2.3 Clause 7.2.2.17.2.1 Overview (of BIFS Updates)

The text mentions four update commands, but the item list includes five. The fifth one is the ‘RepeatScene’ command. The text states that it ‘enables to repeat all the updates from the last Replace Scene.’ However, no syntactic support is provided in Clause 7.2.3.2 (BIFS-Update Syntax).

The following solution is proposed. A 1-bit flag called ‘repeat’ is added in the UpdateCommand syntax (Clause 7.2.3.2.2) following the 3-bit ‘Command’ code. If this flag is set, this indicates that the command contained is a repetition of a previous command. This allows any command to be repeated.

The rationale for the ‘RepeatScene’ command must extend to object descriptor updates as well. A 1-bit flag should be added in Clause 7.3.2.2 (OD-Update Syntax and Semantics) before the 8-bit field ‘objectCount’. To maintain byte alignment, the ‘objectCount’ field can be changed to 7 bits.

### 2.4 Clause 7.2.4.1.10 QuantizedField

Quantization of fields with infinite ranges using quantization code 13 is improperly defined. It only employs as additional information the number of bits to use for uniform quantization (see also Clause 7.2.5.1.2.9, detailing the functionality, syntax, and semantics of the QuantizationParameter node). Since the dynamic range is infinite, such a scheme cannot be used. There are further problems, for example the definition of scope in non-local QuantizationParameter nodes, as well as the description of the detailed use of the fields of such nodes for quantization (e.g., normals).

These flaws indicate that the specification of quantization of BIFS fields in its current form is not yet sufficiently mature. Furthermore, it significantly increases complexity of decoders for a yet to be proven
improvement in compression efficiency. It is recommended that the QuantizationParameter node is removed from the specification. The syntax of all basic fields (SFInt32, etc.) will need to be updated as well, to remove the ‘if (QUANTIZED)’ options.

2.5 Clauses: 7.2.5.3.2.2” FBA up to “7.2.5.3.2.9 FBADefMesh”
The USNB has identified that the Face Animation part of Systems CD is inconsistent in many places (for example, the fieldValue of FBADefTransform cannot not have a variable type) and cannot be implemented as specified in the specification.

Therefore, we recommend that the text related to Face Animation be updated according to the agreement of experts using Annex A as a starting point.

2.6 Clause 7.3.3.5  IPI_Descriptor
The IPI data set is a hook to be used by systems external to the 14496 standard to aid in the identification of content. The 14496-1 specification allows the use of elementary streams in multiple contexts, to be retransmitted to other compliant devices and to be moved from one composition into another. The current specification doesn’t specify the handling of the IPI data set in these types of applications. When elementary streams that are identified by IPI data sets are moved into new contexts, they should continue to be identified by this IPI data set.

Change:

The IPI_Descriptor includes a mechanism to identify content. This is done by means of one or more IP_IdentificationDataSet included in the IPI_Descriptor.

To:

The IPI_Descriptor includes a mechanism to identify content. This is done by means of one or more IP_IdentificationDataSet included in the IPI_Descriptor. IPI Data Sets shall follow the streams with which they are associated in the event that such streams are to be reused in other contexts.

2.7 Clause 7.7.3.3.2 Content classification descriptor
The syntax for the content classification descriptor involves ‘N’ content classifier bytes. However, the value of ‘N’ cannot be inferred by the current specification. It is therefore recommended that either an appropriate table from which ‘N’ can be obtained is defined, or the descriptor be removed from the specification.

2.8 Clause 7.3.3.5.2: Semantics
This change is also motivated by the previous comment but shows that the syntax itself is broken if the IPI data set is inadvertently or maliciously removed.

Change:

IPLes_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.

To:
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 a session. The availability of this elementary stream is undefined if the ES_Descriptor corresponding to the IPI_ES_Id is not available.

2.9 Clause 7.3.3.5.3.2: Semantics

Version 1 MPEG-4 devices are not expected to handle version 2 (or greater) IPI data sets or streams associated with them. For example, IPR protection mechanisms such as encryption will likely become part of the later versions. Version 1 devices will not be able to handle such streams.

Change:

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.

To:

compatibility – shall be set to 00. If an MPEG-4 version 1 player reads a value different from 00, the behavior of the Version 1 system is unspecified.

2.10 Lack of Extensibility

The current coding of nodes based on the node data type prohibits a backwards compatible extension of the current specification. In other words, if a Version 2 is specified that introduces new nodes, and content is created which only uses nodes of Version 1, this content will not be playable by a Version 1 player.

The following solution is proposed.

1) Making all NType fields have the same length of 8 bits (Clause 7.2.3.1.3). The value 0xFF should be reserved for further extending the number of nodes in the future.

2) Refer to all nodes by a single number (their SFWorldNode code, expanded to 8 bits).

In order to still capture the SFNode child field context (to indicate which nodes are valid childs of a parent node), node data type tables can still be used. However, a single node coding table is employed regardless of the node data type of the parent field.

A similar problem exists with field codes. In order to allow future extension of the fields contained in a node, it is recommended that each field ID code (DEFid, Inid, OUTid, DYNid) is expanded to include an escape code (all ‘1’s) that can be used in the future to expand the range of fields. This requires a modification of the node coding tables to ensure that the number of bits used is sufficient to include this escape value.

2.11 Clause 7.3.2: Object Descriptor Protocol

The careful review of the object descriptor protocol contained in the 14496-1 CD has revealed several flaws and concerns. Due the extensive nature of the comments, they are contained in Annex B of this document. Note that this Annex separately identifies 5 major technical, 4 technical, and 1 editorial comment.
2.12 Clause 7.2.5.1.2.4 AudioSource
Coding of Structured Audio sound samples is currently very inefficient in cases when the Sample Bank Format cannot be used. Use of natural audio decoders for this purpose is desired.

See Annex C for details and suggested solution. This comment also affects Clause 5.6.2 of 14496-3.

3. Technical

3.1 Clause 7.2.2.14.1 Overview of Sound Node Semantics
This comment also affects Clauses 5.5.6.12, 5.5.6.14, 5.5.7.7, 5.5.11.4, 5.5.11.5, 5.5.13.3, and 5.9.4.1.1 of 14496-3 (Audio CD).

The resampling specification for audio composition in Section 7.2.2.14.1 of CD 14496-1 is inadequate for producing high-quality audio output. In addition, there are numerous places in both the current Systems CD and the current Audio CD where audio sample-rate conversion (and bandlimited interpolation, a very similar technique) are discussed. These include Clause 7.2.2.14.1.1 of CD 14496-1, and Clauses 5.5.6.12, 5.5.6.14, 5.5.7.7, 5.5.11.4, 5.5.11.5, 5.5.13.3, and 5.9.4.1.1 of CD 14496-3 (there is only a placeholder for interpolation in several of these sections in the current CD). In the interests of “one tool for one function”, these discussions should be unified in a single place within MPEG-4 and referenced as needed by various components of the standard. USNB experts have begun to assemble suggested revisions for this purpose.

3.2 Clause 7.2.2.7.1 Nodes
In the last sentence, it is stated that “Object Descriptors are denoted in the URL field with the scheme “mpeg4od:<number>”. This is incorrect; the proper syntax involves a 1-bit flag (see Clause 7.2.3.1.9.9). The text should be replaced by “Reference to URLs or Object Descriptors is performed with SFURL fields,

3.3 Clause 7.2.2.17.2 External Modification of the scene: BIFS Update
The current specification requires a completely different syntax for the description of the initial BIFS tree, compared with updates. In particular, the initial tree (BIFSScene) is not encapsulated in an ‘Update’ command. This creates unnecessary complication for a scene decoder, and it further complicates editing of BIFS streams.

It is recommended that the initial BIFSScene node is provided using the ‘Update’ syntax, using the already defined ‘ReplaceScene’ command. Upon initial processing of a BIFS stream, a decoder would then have to just ignore initial commands until the first ‘ReplaceScene’ command is located. Note also that ObjectDescriptor streams follow this approach as well.

3.4 Clause 7.2.3.1.2 BIFSNodes
BIFSNodes can be considered a regular BIFS node. However, the current specification uses a custom syntax (e.g., a ‘hasWorldInfo’ flag is used to determine if a ‘WorldInfo’ node is included or not). This does not provide any benefit, and makes implementations more complex thus reducing robustness and increasing their footprint. It is therefore recommended that the BIFSNodes be defined as a regular node. A new node data type must be created, SFBIFSNodesNode, containing only this node. This node data type should be the type of the field ‘nodes’ of ‘BIFSScene’ (Clause 7.2.3.1.1).
The fields of a BIFSNodes should be defined as regular ‘field’ structures, with DEFids 0 for the ‘info’ field that has type SFWorldInfoNode, and a DEFid of 1 for the ‘node’ field that has type SFTopNode.

3.5 Clause 7.2.3.1.3 BIFSNodes
The SFNode syntax uses the flag ‘predefined’. However, the syntax provided does not specify any alternate syntax for the case that this flag is set to 0. It is recommended that this flag is removed.

3.6 Clause 7.2.3.1.3 SFNode
Using node pointers via the ‘isReused’ option may create infinite loops. It is not stated if such loops are allowed or not. A similar situation can arise with ROUTE constructs. It should be explicitly stated that such loops are not allowed.

3.7 Clause 7.2.3.1.8 MFField
If an MFField is empty, but its default is not empty, then the MFListDescription cannot be used because the termination flag has to follow at least one non-empty field. Note that this is not a problem with the MFVectorDescription. To rectify this shortcoming, we recommend that the termination flag precedes the fields, i.e.:

```c
class MFListDescription(fieldType) {
    bit(1) moreFields;
    while (moreFields) {
        SFField(fieldType) field;
        bit(1) moreFields;
    }
}
```

3.8 Clause 7.2.3.1.9.4 SFImage
SFImage is not among the fundamental BIFS types, and it should be removed. It is not used anywhere in the specification.

3.9 Clause 7.2.3.2.3.3 ROUTE Insertion
This command is missing the ROUTEID. As a result, inserted ROUTEs cannot be removed later on. The following construct should be added before ‘departureNodeID’:

```c
bit(1) isUpdateable;
if (isUpdateable) {
    unsigned int(10) ROUTEID;
}
```

3.10 Clause 7.2.3.1.9.7 SFString
SFString fields cannot be quantized. Hence the ‘if (QUANTIZED)’ part of the syntax should be removed.

Also ‘UTF8String’ is not defined. The correct syntax here is:

```c
int(8) length;
char(8) value[length];
```

3.11 Clause 7.2.5.3.2.3 Face
The description of the main functionality of this node mentions a ‘url’ field which does not appear in the semantic table. In addition, the detailed semantics section does not provide any further reference of that
field, but it does mention another field, ‘objectDescriptorID.’ None of these fields is present in the node coding table for Face (Clause 7.2.6.3.55).

The recommended correction is the addition of a ‘url’ field in the ‘Face’ node (this correction must also be made in the node coding table). Also, in the detailed semantics section, ‘objectDescriptorID’ should be

Note that in all Face animation-related nodes, improper references to fields as SFNodes are contained (e.g., FBADefTransform in Clause 7.2.5.3.2.8, or FBADefMesh in Clause 7.2.5.3.2.9). In addition, sometimes these fields are referred to as containing node IDs that point to the relevant nodes. The node coding tables, however, have a different syntax, identifying such fields as SF3DNodes.

AE: The Face anim stuff requires significant revision. It is not clear to me that the references to the various node structures are consistent. Perhaps a major tech for reject should be recommended, in order to allow cleanup of all such nodes.

3.12 Clause 7.2.5.3.16 LOD
The USNB notes that the current specification of the BIFS node LOD is based on the VRML node LOD. The extended functionality of the 14496-1 LOD node allows to specify 3D models that are to be used depending on the frame rate at which the decoder is able to render. USNB supports this functionality. The USNB considers the fpsRange field to be important for applications requiring a certain frame rate and requests that this field not be removed during any changes based on maximal alignment with VRML.

3.13 Clause 7.2.5.4.2.3 Detailed Semantics (of Composite2DTexture)
It is mentioned that the ‘size’ field is in pixels, in which case its type should be ‘SFInt32’ rather than ‘SFVec2f’. It is suggested that regular non-pixel metrics are used, and that ‘pixels’ is removed from the sentence.

3.14 Clause 7.2.5.4.2.4.3 Detailed Semantics (of Composite3DTexture)
It is mentioned that the ‘size’ field is in pixels, in which case its type should be ‘SFInt32’ rather than ‘SFVec2f’. It is suggested that regular non-pixel metrics are used, and that ‘pixels’ is removed from the sentence.

3.15 Support for audiovisual content representation in formats not defined by ISO/IEC
It is recommended that 14496-1 explicitly supports audiovisual representation formats that have not been designed by ISO/IEC WG11. More specifically, this includes GIF, JPEG, and AC-3. Support here means that elementary streams containing data complying to these specifications can be included in scene descriptions and decoded by MPEG-4 terminals.

3.16 Clause 7.8.1 Scene Description Profiles
The scene description profiles defined in the CD do not provide a suitable profile for low-complexity scene description of “traditional” audio-video material.

Insert the following text before 7.8.1.1:

7.8.1.1 Basic profile
Applications supporting only traditional audio-video capabilities have to conform to the “Basic Systems Profile” as specified in this Subclause. This is intended for systems that implement low-complexity video and audio playback.

The “Basic Systems Profile” is defined by the following nodes: Layer2D, VideoObject2D, Sound2D, and SoundSource.

4. Editorial

4.1 Table of Contents
Entry for item ‘InitialAnimQP’ (Clause 7.2.3.3.1.4) has a preceding ‘}’.

4.2 Clause 0.1
Figure 0-1 should contain a straight line connecting the AU Layer directly to the TransMux Layer, so that it is made obvious that the use of the FlexMux layer defined by the 14496-1 specification is optional.

The caption for Figure 0-1 has an embedded newline character that should be removed (it also appears in all references to that figure).

4.3 Annex A
No bibliography is provided. The Systems chair and editors should assemble and include a list of pertinent references.

4.4 Clause 7.2.1.1 Scope
Figure 7-3 includes references to “hierarchically multiplexed” streams. Streams are not hierarchically multiplexed; hence this attribute should be removed.

4.5 Clause 7.2.1.3.1 Grouping of Objects
The MPEG-4 scene description is described as a “Directed Acyclic Graph.” However, a BIFS structure without node pointers is a simple tree. We recommend that the term ‘tree’ be used in this clause. If pointers are used, then the structure is not a tree anymore, but it indeed becomes a restricted version of a directed acyclic graph. This restriction, however, should be made explicit.

4.6 Clause 7.2.1.3.3 Attribute Value Selection
The definition of a “Media Object” included in the first paragraph should also be included in the list of definitions (Clause 4).

Also, the word “section” in the second paragraph should be replaced by “subclause”.

4.7 Clause 7.2.2.1 Global Structure of a BIFS Scene Description
Replace “kinds” in the first paragraph with “types”.

Spelling errors are not identified in this contribution, under the assumption that their correction is at the Editors’ discretion.
4.8 Clause 7.2.2.17.3 External animation of the scene: BIFS-Anim
It is not clear from the specification where a BIFS animation can be attached. Our examination indicates
that AnimationStream nodes can be attached anywhere in the tree, and can affect any node in the tree (in
accordance, of course, to node data type restrictions). This should be explicitly stated.

4.9 Clause 7.2.2.11 Scene Structure and Semantics
The nodes ‘Inline’ and ‘Inline2D’ are not included in any category. As this section describes MPEG-4
specific nodes, ‘Inline2D’ should be included in the ‘Mixed 2D/3D Nodes’ category (Clause 7.2.2.11.5).

4.10 Clause 7.2.2.12 Internal, ASCII and Binary Representation of Scenes
Reference to ‘MPEG-4’ in the first sentence should be replaced by “This Committee Draft of International

4.11 Clause 7.2.2.13.3 Requirements on BIFS elementary stream transport
Reference to “Working Draft” should be replaced by “Committee Draft”.

4.12 Clause 7.2.2.13.6 Multiple BIFS Streams
The last sentence of the second paragraph states “It is forbidden to reference parts of the scene outside the
name scope of the BIFS stream.” This refers to included BIFS trees via inline nodes. This sentence is
misleading and should be removed. Since a new BIFS object descriptor introduces a new scope, it is
impossible for an included tree to refer to nodes of the parent tree.

4.13 Clause 7.2.2.13.7 Time Fields in BIFS nodes
The last sentence indicates “any time duration is therefore”. This should be replaced by “any time instant,

4.14 Clause 7.2.2.16 Bounding Boxes
It is stated that when a bounding box is specified, it should enclose the shape. It should be clarified if it
should be the smallest such rectangle, otherwise the term ‘bounding box’ may be misleading.
Also, the last sentence of the paragraph refers to a “browser”. This should be replaced by “terminal”.

4.15 Clause 7.2.3 BIFS Binary Syntax
As the syntax specification for BIFS nodes does not fully adhere to the MSDL specification (Clause 7.6),
this should be stated in the introduction to this section to avoid misunderstanding. The following
paragraphs are suggested:

The syntax of all BIFS-related structures is described in this clause. Coding of individual nodes is
very regular; however, identification of nodes and fields within a BIFS tree depends on the context.
Each field of a BIFS node can accept a specific set of children nodes. This is referred to as a Node
Data Type (or NDT). Each node belongs to one or more node data types. Node data type tables are
provided in Clause 7.2.6.2. Identification of particular node type depends on the context of the
Node Data Type specified for its parent field. For example, ‘MediaTimeSensor’ is identified by the
5-bit code 0b0000.1 when the context of the parent field is SF2DNode, whereas the 7-bit code
0b0000.110 is used in the context of a SFWorldNode field.
The syntactic description of fields is also dependent on the context of the node. In particular, fields are identified by code words that have node-dependent (but fixed) lengths. The type of the field is inferred by the code word. Field codes are provided in Clause 7.2.6.3 (Node Coding Tables).

In what follows, a pseudo-MSDL syntax is used in order to describe the BIFS syntax. The pertinent node data type (NDT) and node coding table (NCT) are assumed to be available as two structures names NDT and NCT respectively; their members provide access to particular values in the actual tables.

4.16 Clause 7.2.3.1.3 SFNode

The ‘Ntype’ is simply a binary field, so its declaration should be:

```
bit(NDT.nbits) NType;
```

It should also be mentioned that NDT.nbits is the last column of the header of the Node Coding Tables.

In the same text, the MaskNodeDescription and ListNodeDescription entries should be ‘fields’, not ‘node’.

4.17 Clause 7.2.3.1.4 MaskNodeDescription

The ‘if mask’ requires parentheses surrounding ‘mask’. Also ‘NumberOfFields’ should be written as:

```
NCT.numDEFfields
```

where NCT refers to the particular node coding table, and numDEFfields is the number of DEF fields for that node.

Also, ‘Field(fieldType) value;’ should be replaced by:

```
Field(NCT.fieldType[i]) value;
```

where fieldType[i] refers to the type of the field with DEFid ‘i’.

4.18 Clause 7.2.3.1.5 ListNodeDescription

The syntax should be rewritten as follows (note that a ‘!’ is missing from the ‘endFlag’ test):

```
bit(1) endFlag;
while (!EndFlag) {
  bit(NCT.numDEFbits) fieldReference;
  Field(NCT.fieldType[fieldReference]) value;
  bit(1) endFlag;
}
```

Alternatively, this can be incorporated in the original declaration of defID in Clause 7.2.3.1.11.1.

4.19 Clause 7.2.3.1.6 NodeType

This clause can be removed (assuming the correction described above for SFNode is adopted).

4.20 Clause 7.2.3.1.7 Field

The ‘fieldType’ argument should be ‘int fieldType’.

4.21 Clause 7.2.3.1.8 MFField

The ‘fieldType’ argument should be ‘int fieldType’.
4.22 Clauses 7.2.3.1.9.1 – 7.2.3.1.9.11 (Basic Types Syntax)

The use of ‘QuantizedField Value’ should be accompanied with the type of quantization pertinent for the field at hand. This depends on the node coding table. We suggest that each field be passed an NCT structure, and that declarations of QuantizedField structures is performed as follows:

```c
if (QUANTIZED) {
    QuantizedField value(NCT.field[fieldReference].Quant);
} else {
    ... // as before
}
```

where Quant is the ‘Q’ column of the node coding table for the parent node.

4.23 Clause 7.2.3.1.11 Field IDs Syntax

These definitions are superfluous if proper definition of each ID using NCT (see above for ListNodeDescription) is adopted. Alternatively, instead of vlcNbBits, the following should be used:

```c
bit(NCT.numDEFbits) id;
```

with similar constructors for IN, OUT, and DYN.

4.24 Clause 7.2.6.2 Node Data Type Tables

The table used the term ‘nodeID’ to refer to the code that identifies each node within the various node data types. However, this term is already used to denote the 10-bit node identifier for updateable nodes. It is recommended that the term ‘NType’ is used, which is also the one used in the generic node syntax in Clause 7.2.3.1.2.

4.25 Clause 7.6 Syntactic Description Language

It is suggested that this subclause become the last clause of Clause 7, i.e., that it becomes Clause 7.7. Furthermore, the MPEG-4 Systems Editors should ensure that the use of MSDL is correct throughout the document.

4.26 Clause 7.3.3.3.2 Semantics (of DecoderConfigDescriptor)

The current definitions for maxBitrate and avgBitrate are as follows:

- **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.

These definitions simply do not make sense. For example, for maxBitrate the term ‘instantaneous’ is meaningless. Furthermore, the specific association of a time scale (1 sec) may not properly capture the dynamic of some elementary stream types.

As these bit rates affect buffer management and allocation, their definitions should be directly linked to quantities useful for such purposes. We therefore recommend that these definitions are modified as follows:

- **maxBitrate**: the smallest bit rate such that over any time interval of length $dt$, $bits_{received} \leq \text{maxBitrate} \times dt + 1$. 
avgBitrate: a bit rate such that, over all intervals of length dt, \( \text{bits} \text{received} \leq \text{avgBitrate} \times dt + 1 \). A value of zero indicates that this Elementary Stream does not have a constant average bit rate.

4.27 Clause 7.3.3.2.2 Semantics (of ES__descriptor)
The formula for the computation of the ES_id is wrong:

\[
\text{ES}_\text{Id} = \text{objectDescriptorID} \ll 5 \& \text{ES}_\text{Number}
\]

The correct formula is:

\[
\text{ES}_\text{Id} = \text{objectDescriptorID} \ll 5 \mid \text{ES}_\text{Number}
\]

4.28 Clause 7.2.3.1.8 MFField
The ‘moreFields’ flag should be changed to ‘endFlag’, for uniformity with similar constructs in other parts of the specification.

4.29 Clause 7.2.6.1 Constant-Length Direct Representation Bit Fields
The type ‘float’ should be explicitly mentioned in the list of fundamental types of SDL. It should also be mentioned that its representation uses the standard IEEE 32-bit format.

4.30 Clause 7.2.5.2.2.6.1 Semantic Table (of Form)
The ‘children’ field should have type MF2DNode, instead of MFNode. The same error appears in the semantic tables of Clauses 7.2.5.2.2.7.1, 7.2.5.2.2.12.1, 7.2.5.2.2.23.1. The node coding tables are correct.

4.31 Clause 7.2.5.2.2.15 VideoObject2D
This clause is duplicated in Clause 7.2.5.2.2.24. It is recommended that Clause 7.2.5.2.2.15 is removed.

4.32 Clause 7.2.5.3.2.4.2
Reference to contribution W1825 should be removed from the text.

4.33 Clause 7.2.5.3.3.12.1 Semantic Table (of Group)
The various children fields should have type MF3DNode, instead of MFNode. The node coding table is correct.

4.34 Clause 7.2.5.3.3.26.1 Semantic Table (of Transform)
The various children fields should have type MF3DNode, instead of MFNode. The node coding table is correct.

4.35 Clause 7.2.5.4.2.1.1 Semantic Table (of Layer2D)
The various children fields should have type MF2DNode, instead of MFNode. The node coding table is correct.

4.36 Clause 7.2.5.4.2.2.1 Semantic Table (of Layer3D)
The various children fields should have type MF3DNode/SF3DNode, instead of MFNode/SFNode. The ‘childrenLayer’ field in particular should have type ‘MFLayerNode’. The node coding table is correct.
4.37 Clause 7.2.5.4.2.3.1 Semantic Table (of Composite2DTexture)
The ‘children’ field should have type MF2DNode, instead of ‘MFNode’. The node coding table is correct.

4.38 Clause 7.2.5.4.2.4.1 Semantic Table (of Composite3DTexture)
The ‘children’ field should have type MF3DNode, instead of ‘MFNode’. The node coding table is correct.

4.39 Clause 7.2.5.4.2.5.1 Semantic Table (of CompositeMap)
The ‘children2D’ field should have type MF2DNode, instead of ‘MFNode’. The node coding table is correct.

4.40 Incorrect use of SFString for SFUrl fields
The semantic tables contained in the following clauses mistakenly identify the type of URL fields as SFString instead of SFUrl: 7.2.5.1.2.1.1, 7.2.5.1.2.4.1, 7.2.5.1.2.10.1, 7.2.5.1.3.2.1, 7.2.5.1.3.6.1, 7.2.5.1.3.7.1, 7.2.5.2.2.1.1, 7.2.5.2.2.8.1, 7.2.5.2.2.11.1, 7.2.5.2.2.24, and 7.2.5.3.3.15.1.
Note that the corresponding node coding tables are correct.

4.41 Annex C.2: MPEG-4 Content Embedded in MPEG-2 DSM-CC Data Carousel
Editorial changes are required to bring this Annex into alignment with the DMIF CD.

4.41.1 Clause C.2.3 “General Concept” sub-section
Current text is not consistent with the current DMIF specifications. The purpose of the Stream Map Table is to bind each Elementary Stream ES_id value to a particular connection identified by a channelAssociationTag. The channelHandle is generated at runtime by the underlying DMIF session. It is recommended to change the following text:

*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).*

To:

*The Stream Map Table associates each Elementary Stream identifiers (ES_Id) with a module a channelAssociationTag which uniquely identifies the channel in the underlying DMIF session.*

4.41.2 Clause C.2.3 “General Concept” sub-section
The name of the DMIF interfaces and primitives has changed. It is recommended to change current text to reflect new interface names. More specifically, it is proposed to change the text:

*Each of these messages is viewed as the payload of an MPEG-4 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.

To:

Each of these messages is viewed as the payload of an MPEG-4 FlexMux AL-PDU. The application requests to open a channel by means of the DA_ChannelAdd() primitive of the DAI (DMIF Application Interface). The ES_id field is an input argument identifying the desired elementary stream. The underlying DMIF session replies by providing an handle to the multiplexed channel carrying the elementary stream. This handle is the channelHandle and is unique in the application regardless how many sessions are open. Subsequently, the DMIF session informs the application of the arrival of new data by means of the DA_Data() primitive of the DAI.

4.41.3 Clause C.2.5.4 “Stream Map Table” sub-section

The current text does not reflect proper use of the Stream Map Table. The Stream Map table provides a binding between the Elementary Stream Identifier (ES_id) and the DMIF channelAssociationTag. Furthermore, there is no suggested relationship between the channelAssociationTag and the TransMuxChannelAssociationTag. Therefore, it is recommended to change the text

The Stream Map Table links Elementary Stream Identifiers (ES_id) used by BIFS and the MPEG-4 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.

To:

The Stream Map Table links Elementary Stream Identifiers (ES_id) used by BIFS and the MPEG-4 application to the channelAssociationTag (CAT) value used by DMIF to refer to the connection conveying the stream. The channelAssociationTag is a 4 byte field. A suggested choice for channelAssociationTag is (TransmuxAssociationTag << 16 + moduleId and the transmuxAssoicationTag is the associationTag field defined in the MPEG-2 associationTag descriptor.

4.41.4 Table C-13, “DownloadInfoIndication message”

The channelHandle field is generated at runtime by the underlying DMIF session and in particular cannot be preset. Therefore, it is recommended to change the text:

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.

To:
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.

4.41.5 New Section

The MPEG-4 CD defines an informative Elementary Stream Interface. To promote the use of this interface as well as of the DMIF DAI interface, it is recommended to append the following text at the end of the current Annex:

C.2.6 Elementary Stream Interface

The following interface primitives are used between the Compression Layer and the Access Unit Layer in the application:

The client application requests a reference to a particular module (flexmux channel) by means of the following interface:
ESI.channelOpen.request(AppSessionId, ES_id, ALHeaderConfigDescriptor).

The Access Unit Layer exposes the channelHandle through the following interface:
ESI.channelOpen.confirm(channelHandle, Response)

Systems informs user that elementary stream is available on channel:
ESI.channelOpen.indication(appSessionId, ES_id, ALHeaderConfigDescriptor, channelHandle)
The application acknowledges with:
ESI.channelOpen.response(Response)

DMIF APPLICATION INTERFACE

The application sends to the underlying DMIF session the request to open a new channel by mean of the following interface primitive:
DA_ChannelAdd.request(serviceSessionId, loop(ES_id))
The underlying DMIF session replies by means of the following interface primitive:
DA_ChannelAdd.confirm(loop(channelHandle, response))
The application then informs DMIF session that it is ready to receive data:
DA_ChannelReady(channelHandle))
DMIF provides application with data through the following interface primitive:
DA_DataCallBack(channelHandle, streamDataBuffer, streamDataLen, errorFlag)

5.
Annex A: Changes to FBA Text

Replace “7.2.5.3.2.2” FBA up to “7.2.5.3.2.9 FBADefMesh” with the following:

5.1.1.1.1 FBA

5.1.1.1.1.1 Semantic Table

FBA {
  ExposedField SFFaceNode face NULL
  ExposedField SFBodyNode body NULL
}

5.1.1.1.1.2 Main Functionality

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

5.1.1.1.1.3 Detailed Semantics

face contains a Face node

body contains a Body node, not yet defined

5.1.1.1.2 Face

5.1.1.1.2.1 Semantic Table

Face {
  exposedField SFFITNode fit NULL
  exposedField SFFDPNode fdp NULL
  exposedField SFFAPNode fap NULL
  exposedField SFURL url NULL
  Field MF3DNode renderedFace NULL
}

5.1.1.1.2.2 Main Functionality

Organizes definition and animation of a face. The FAP node is mandatory, the FDP node, defining 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 FIT node is optional. When present it allows a set of Facial Animation Parameters (FAPs) to be defined in terms of another set of FAPs.

The url field specifies the data source to be used.

5.1.1.1.2.3 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
url  The url field specifies the data source to be used

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

5.1.1.1.1.3  FAP

5.1.1.1.1.3.1  Semantic Table

<table>
<thead>
<tr>
<th>ExposedField</th>
<th>SFNode</th>
<th>viseme</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExposedField</td>
<td>SFNode</td>
<td>expression</td>
<td>0</td>
</tr>
<tr>
<td>ExposedField</td>
<td>SFInt32</td>
<td>open_jaw</td>
<td>0</td>
</tr>
<tr>
<td>ExposedField</td>
<td>SFInt32</td>
<td>lower_t_middlip</td>
<td>0</td>
</tr>
<tr>
<td>ExposedField</td>
<td>SFInt32</td>
<td>raise_b_middlip</td>
<td>0</td>
</tr>
<tr>
<td>ExposedField</td>
<td>SFInt32</td>
<td>stretch_l_corner</td>
<td>0</td>
</tr>
<tr>
<td>ExposedField</td>
<td>SFInt32</td>
<td>stretch_r_corner</td>
<td>0</td>
</tr>
<tr>
<td>ExposedField</td>
<td>SFInt32</td>
<td>lower_t_lip_Lm</td>
<td>0</td>
</tr>
<tr>
<td>ExposedField</td>
<td>SFInt32</td>
<td>lower_t_lip_Rm</td>
<td>0</td>
</tr>
<tr>
<td>ExposedField</td>
<td>SFInt32</td>
<td>lower_b_lip_Lm</td>
<td>0</td>
</tr>
<tr>
<td>ExposedField</td>
<td>SFInt32</td>
<td>lower_b_lip_Rm</td>
<td>0</td>
</tr>
<tr>
<td>ExposedField</td>
<td>SFInt32</td>
<td>raise_l_cornerlip</td>
<td>0</td>
</tr>
<tr>
<td>ExposedField</td>
<td>SFInt32</td>
<td>raise_r_cornerlip</td>
<td>0</td>
</tr>
<tr>
<td>ExposedField</td>
<td>SFInt32</td>
<td>thrust_jaw</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>shift_jaw</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>push_b_lip</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>push_t_lip</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>depress_chin</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>close_t_l_eyelid</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>close_t_r_eyelid</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>close_b_l_eyelid</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>close_b_r_eyelid</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>yaw_l_eyeball</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>yaw_r_eyeball</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>pitch_l_eyeball</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>pitch_r_eyeball</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>thrust_l_eyeball</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>thrust_r_eyeball</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>dilate_l_pupil</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>dilate_r_pupil</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>raise_l_l_eyebrow</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>raise_r_l_eyebrow</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>raise_l_m_eyebrow</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>raise_r_m_eyebrow</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>raise_l_o_eyebrow</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>raise_r_o_eyebrow</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>squeeze_l_eyebrow</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>squeeze_r_eyebrow</td>
<td>0</td>
</tr>
<tr>
<td>exposedField</td>
<td>SFInt32</td>
<td>puff_l_cheek</td>
<td>0</td>
</tr>
</tbody>
</table>
5.1.1.1.3.2 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).

5.1.1.1.3.3 Detailed Semantics

**Viseme**
Contains a Viseme node.

**Expression**
Contains and Expression node.

The semantics for these parameters are described in the Annex C of the Visual CD, and in particular in the Table 12-1.

**open_jaw**, …,
**pupil_r_ear**
A FAP of value +I is assumed to be uninitialized.
5.1.1.1.4.1 Semantic Table

Viseme {
    Field SFInt32 viseme_select1 0
    Field SFInt32 viseme_select2 0
    Field SFInt32 viseme_blend 0
    Field SFBool viseme_def 0
}

5.1.1.1.4.2 Main Functionality
The Viseme node defines a blend of two visemes from a standard set of 14 visemes as defined in table 12-5 in Visual CD.

5.1.1.1.4.3 Detailed Semantics
Viseme_select1  Specifies viseme 1.
Viseme_select2  Specifies viseme 2.
Viseme_blend    Specifies the blend of the two visemes.
Viseme_def      If viseme_def is set, current FAPs are used to define a viseme and store it.

5.1.1.1.5 Expression

5.1.1.1.5.1 Semantic Table
Expression {
    field SFInt32 expression_select1 0
    field SFInt32 expression_intensity1 0
    field SFInt32 expression_select2 0
    field SFInt32 expression_intensity2 0
    field SFBool init_face 0
    field SFBool expression_def 0
}

5.1.1.1.5.2 Main Functionality
The Expression node is used to define the expression of the face as a combination of two expressions out of the standard set of expressions defined in table 12-3 in Visual CD.

5.1.1.1.5.3 Detailed Semantics
expression_select1 specifies expression 1.
expression_intensity1 specifies intensity for expression 1.
expression_select2 specifies expression 2.
expression_intensity2 specifies intensity for expression 2.
init_face      If init_face is set, neutral face may be modified before applying FAPs 3-68.
expression_def  If expression_def is set, current FAPs are used to define an expression and store it.

5.1.1.1.1.6  FIT

5.1.1.1.1.6.1 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
}

5.1.1.1.1.6.2 Main Functionality

The FIT node allows a smaller set of FAPs to be sent during a facial animation. This small set can then be
used to determine the values of other FAPs, using a rational mapping between parameters. For example,
the top inner lip FAPs can be sent and then used to determine the top outer lip FAPs. The inner lip FAPs
would be mapped to the outer lip FAPs using a rational function that is specified in the FIT node.

To make the scheme general, sets of FAPs are specified, along with a graph between the sets that specifies
which sets are used to determine which other sets. In some situations, a set of FAPs can be determined from
more than one other set of FAPs, in which case the arcs that determine the relationship between sets of
FAPs also have a priority.

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.
The interpolation process based on the FAP interpolation graph is described using pseudo C code as follows:

```c
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.

Each directed link in an FIG is a set of interpolation functions. Suppose $F_1, F_2, \ldots, F_n$ are the FAPs in a parent set and $f_1, f_2, \ldots, f_m$ are the FAPs in a child set. Then, there are $m$ interpolation functions denoted as:

- $f_1 = I_1(F_1, F_2, \ldots, F_n)$;
- $f_2 = I_2(F_1, F_2, \ldots, F_n)$;

Figure 1: An example FIG
\[ f_m = I_m(F_1, F_2, \ldots, F_n); \]

Each interpolation function \( I_k() \) is in a rational polynomial form

\[ I(F_1, F_2, \ldots, F_n) = \sum_{i=0}^{K-1} \left( \sum_{j=1}^{n} c_i \prod_{j=1}^{n} F_j^{l_{ij}} \right), \]

where \( K \) and \( K \) are the numbers of polynomial products, \( c_i \) and \( c_j \) are the coefficient of the \( i \)th product. \( l_{ij} \) and \( l_{ij} \) are the power of \( F_j \) 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. The encoder should send an interpolation function table which contains all \( K, K, c_i, c_j, l_{ij}, l_{ij} \) to the decoder for each child FAP.

5.1.1.1.6.3 Detailed Semantics

To aid in the explanation below, we assume that there are \( N \) different sets of FAPs, and that each set has \( n_i \), \( i=1,\ldots,N \) parameters. We assume that there are \( M \) sets of child FAPs and that they have \( m_i \), \( i=1,\ldots,M \) parameters each. We assume that there are \( L \) sets of parent FAPs.

- **FAPs**: a list of FAP-indices specifying which animation parameters form sets of FAPs. Each set of FAP indices is terminated by a -1. There should be a total of \( N + n_1 + n_2 + \ldots + n_N \) numbers in this field, with \( N \) of them being -1.

- **graph**: A list of pairs of integers, specifying a directed arc between sets of FAPs. The integers refer to the sets specified in the **FAPs** field. Each set is referenced by an integer that enumerates its position in the list, starting with 1. 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. This field should have a total of \( 2L \) numbers, corresponding to the arcs between the parents and children in the FIG.

- **numeratorTerms**: A list of the number of terms in the polynomials in the numerators of the rational functions used for interpolating parameter values. Each element in the list corresponds to \( K \). Each child parameter in **FAPs** must have one value specified, with the order in the **numeratorTerms** list corresponding to the order that the FAP appears in the **FAPs** field. There should be \( m_1 + m_2 + \ldots + m_M \) numbers in this field.

- **denominatorTerms**: A list of the number of terms in the polynomials in the denominator of the rational functions controlling the parameter value. Each element in this list corresponds to \( P \). Each child parameter in **FAPs** must have one value, with the order in the **denominatorTerms** list corresponding to the order that the FAP appears in the **FAPs** field. There should be \( m_1 + m_2 + \ldots + m_M \) numbers in this field.

- **numeratorExp**: A list of exponents of the polynomial terms in the numerator of the rational function controlling the parameter value. This list corresponds to \( l_{ij} \). The list should have \( n*K \) terms in row order for each child parameter specified in the **FAPs** field. Note that \( n \) and \( K \) may be different for each parameter. Each child parameter
in FAPs must have all the exponents of the corresponding rational function specified, with the order in the numeratorExp list corresponding to the order that the FAP appears in the FAPs field.

**denominatorExp** A list of exponents of the polynomial terms in the denominator of the rational function controlling the parameter value. This list corresponds to $l_j$. The list should have $n\times P$ terms in row order for each parent parameter specified in the FAPs field. Note that $n$ and $P$ may be different for each set of parameters. Each child parameter in FAPs must have all the exponents of the corresponding rational function specified, with the order in the denominatorExp list corresponding to the order that the FAP appears in the FAPs field.

**numeratorCoefs** A list of coefficients of the polynomial terms in the numerator of the rational function controlling the parameter value. This list corresponds to $c_i$. The list should have $K$ terms for each child parameter specified in the FAPs field, with the order in numeratorCoefs corresponding to the order in FAPs. Note that $K$ is dependent on the polynomial, and is not a fixed constant.

**denominatorCoefs** A list of coefficients of the polynomial terms in the numerator of the rational function controlling the parameter value. This list corresponds to $c_i$. The list should have $P$ terms for each child parameter specified in the FAPs field, with the order in DenomenatorCoefs corresponding to the order in FAPs. Note that $P$ is dependent on the polynomial, and is not a fixed constant.

5.1.1.1.6.4 Example

After the FIT node given below, we explain each field separately.

```
FIT {
  FAPs [ 3 4 5 -1 6 7 -1]
  graph [ 1 2 ]
  numeratorTerms 2
  denominatorTerms 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]
}
```

- FAPs [ 3 4 5 -1 6 7 -1]
  We define two sets of FAPs, the first with FAPs number 3, 4, and 5, and the second with FAPs number 6 and 7. We will refer to these as $F_3, \ldots, F_7$
- graph [ 1 2 ]
  We will make the first set the parent of the second set, so that FAPs number 6 and 7 will be determined by FAPs 3, 4, and 5.
- numeratorTerms [ 4 1 ]
  We select the rational functions that defines $F_6$ and $F_7$ to have 4 and 1 terms in their numerator, respectively. Because $F_6$ is the first child node, the 4 refers to it.
We select the rational functions that defines $F_6$ and $F_7$ to have 2 and 1 terms in their denominator, respectively.

The numerator we select for the rational function defining $F_6$ is $F_3 + 2F_4 + 3F_5 + 4F_3F_4^2$. There are 3 parent FAPs, and 4 terms, leading to 12 exponents for this rational function. For $F_7$, the numerator is just $F_4$, so there are three exponents only (one for each FAP).

The denominator we select for the rational function defining $F_6$ is $5F_5 + 6F_3F_4F_5$, so there are 3 parent FAPs and 2 terms and hence, 6 exponents for this rational function. For $F_7$, the denominator is just 1, so there are three exponents only (one for each FAP).

There is one coefficient for each term in the numerator of each rational function.

There is one coefficient for each term in the denominator of each rational function.

This FIT node gives us the following rational functions for defining the child set of FAPs:

$$F_6 = \frac{(F_3 + 2F_4 + 3F_5 + 4F_3F_4^2)}{(5F_5 + 6F_3F_4F_5)}$$

$$F_7 = F_4$$

5.1.1.1.7 FDP

5.1.1.1.7.1 Semantic Table

FDP {
  exposedField SFCoordinateNode featurePointsCoord NULL
  exposedField SFTecTextureCoordinate textureCoords NULL
  exposedField MFBBADefTableNode FBADefTables NULL
  exposedField MF3DNode faceSceneGraph NULL
}

5.1.1.1.8 Main Functionality

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

1. calibration information is downloaded, so that the proprietary face of the receiver can be configured using facial feature points and optionally a scene graph. In this case, the field featurePointsCoord has to be set. FeaturePointsCoord contains the coordinates of facial feature points, as defined in figure 12-1 in the Visual CD, corresponding to a neutral face. If the coordinate of a feature point is set to +1, the coordinates of this feature point are to be ignored. The field faceSceneGraph is optional. If set, the faceSceneGraph has to show a face in neutral position. It can consist of one or more surfaces; however only eyes, tongue and teeth regions can be contained in separate surfaces, and the rest of the face must necessarily be contained in a single surface. When faceSceneGraph is set, featurePointsCoord define the position of the feature points on the surface of the facemodel. Each feature point must correspond (i.e. its coordinates must be equal to) exactly one vertex on one of the surfaces in the faceSceneGraph. If faceSceneGraph is set, it has to include a complete face mesh.
all feature points in FeaturePointsCoord have to be set, i.e. feature points can not be ignored. The faceSceneGraph must be used by the decoder to calibrate the shape of its own face model. For the calibration, the decoder must evaluate the multiple surfaces and related texturemaps of the faceSceneGraph and other information. TextureCoords, if set, are used to calibrate the decoder’s model using only feature points and a texture, i.e. without a calibration mesh. They correspond to the feature points, i.e. each defined feature point must have corresponding texture coordinates. In this case, the faceSceneGraph must contain exactly one texture image, and any geometry it might contain is ignored. The decoder interprets the feature points, texture coordinates, and the surface(s) of the faceSceneGraph in the following way:

a) Feature points of the decoder’s face model are moved to the coordinates of the feature points supplied in featurePointsCoord, unless a feature points is to be ignored as explained above.
b) If faceSceneGraph is set, and textureCoors is not set, the remaining vertices of the decoder’s model are moved in such a way that they lie on one of the surfaces supplied in faceSceneGraph. The correspondence between the surfaces of the decoder model and the surfaces of the faceSceneGraph is established through the feature points.
c) If the conditions under b) are met, and if the surface(s) supplied in faceSceneGraph are textured, the textures are mapped on the surface(s) of the decoder’s model. The mapping is derived from the feature points correspondence between the decoder’s model and the supplied model.
d) If textureCoors is set, the texture supplied in the faceSceneGraph is mapped on the proprietary model. The texture coordinates are derived from the texture coordinates of the feature points supplied in textureCoords.

2. a face model is downloaded with the animation definition of the Facial Animation Parameters hence the fields featurePointsCoord, 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). If desired, the faceSceneGraph contains the texture maps 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 featurePointsCoord in order to allow a low-complexity decoder to adapt their own face model. Therfore,featurePointsCoord define the position of the feature points on the surface of the facemodel in faceSceneGraph. Each feature point must correspond (i.e. its coordinates must be equal to) exactly one vertex on one of the surfaces in the faceSceneGraph.

5.1.1.1.9 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).

textureCoors contains a Coordinate node. Specifies texture coordinates for the feature points. The coordinates are listed in the ‘point’ field in the Coordinate node in the prescribed order, that a feature point with a lower label is listed before a feature point with a higher label (e.g. feature point 3.14 before feature point 4.1).

FBADefTables contains FBADefTable nodes. The behavior of FAPs is defined in this field for the face in faceSceneGraph.

faceSceneGraph contains a Group node. In case of option 1, this is used for model calibration as explained above. In case of option 2, this is the grouping node for face model rendered in the compositor and has to contain the face model. In this case, the effect of Facial Animation
Parameters is defined in the ‘FBADefTables’ field.

5.1.1.1.10  FBADefTable

5.1.1.1.10.1 Semantic Table

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

5.1.1.1.10.2 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.

5.1.1.1.10.3 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.

5.1.1.1.11  FBADefTransform

5.1.1.1.11.1 Semantic Table

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

5.1.1.1.11.2 Main Functionality

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

5.1.1.1.11.3 Detailed Semantics

faceSceneGraphNode 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

5.1.1.1.12  FBADefMesh

5.1.1.1.12.1 Semantic Table

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

5.1.1.1.12.2 Main Functionality

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

5.1.1.1.12.3 Detailed Semantics

- **faceSceneGraphNode**: 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 ]
}

1. 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.
Annex B: Object Descriptor Protocol

The careful review of the object descriptor protocol contained in the 14496-1 CD reveals these flaws and concerns:

6.1 Class Tag Syntax

Subclauses 7.3.2.2.1.1, 7.3.2.2.2.1, 7.3.2.2.3.1, 7.3.2.2.4.1 (Editorial)

6.1.1 Comment

The syntax for the Class Tag of the four classes which constitute the message set is invalid. The ObjectDescriptorUpdate class illustrates the concern:

```c
aligned(8) class ObjectDescriptorUpdate : uint(8) ObjectDescriptorUpdateTag {
    declarations;
};
```

The ObjectDescriptorUpdateTag field is an example of the ClassTag construct of Syntax Description Language. The motivation for the Class Tag field in the four classes which constitute the protocol message set (of which Object DescriptorUpdate is one) is to serve as an opcode. (It is regrettable that the motivation does not extend beyond this; see the discussion on extensible design below.) Since the Class Tag field is the first field of each message, the software which decodes the message stream extracts the ClassTag field to detect which of the four message signatures to expect for the rest of the message.

This is a valid usage of the Class Tag construct, but the syntax is not valid:

```c
aligned(8) class ClassName : uint(8) ClassTag {
};
```

The correct syntax requires that a) the ClassTag appear before the {class definition} and b) that the ClassTag declaration assign its value. The specification includes a table of descriptor tag values, but should also include these values with the class declarations.

6.1.2 Recommendation

Change the syntax in the four relevant subclauses as follows:

```c
aligned(8) class ClassName uint(8) ClassTag=<value> {
    declarations;
};
```

where <value> is between 1 and 255.

6.2 Protocol State

Subclause 7.3.2 (Major Technical)

6.2.1 Comment

It is not clear when the protocol state is complete and consistent. The message stream might contain all four messages in arbitrary sequences. For example, the target device might receive the sequence:

a) ObjectDescriptorUpdate[O[i=1],S[j=1]]

b) ObjectDescriptorUpdate[O[i=2],S[j=1,2]]

c) ObjectDescriptorRemove[O[i=1]]

d) ES_DescriptorRemove[O[i=2],S[j=2]]

e) ES_DescriptorUpdate[O[i=2],S[j=2]]

where Message[O[i]] denotes an object descriptor and Message[O[i],S[j]] denotes a stream descriptor within an object descriptor. (A declaration of the complete signature would just serve to confuse the point.)

The concern is that it is not clear when the protocol state is complete and consistent. It protocol state might be complete after receipt of the b) message and again after receipt of the e) message. (To the extent that one can speculate about plausible sequences, the state after receipt of ObjectDescriptor Remove is perhaps transient, while the state after receipt of ObjectDescriptor Update is perhaps stable.) In the absence of clear
semantics, sequences other than b) and e) in the example might be valid. Perhaps the intent was that the protocol state is complete after receipt of a) or b) or c) or d) or e). The message set suggests a state machine, with transient and stable states, but it's impossible to define the state machine without clear semantics.

6.2.2 Recommendation

Define clear semantics about when the protocol state is complete and consistent, by adopting one of the following four solutions:

a) Adopt the semantics that receipt of the ObjectDescriptorUpdate class or receipt of the ES_DescriptorUpdate class implies that the protocol state is complete. The solution does not introduce messages to the message set. (The appendix provides the state machine consistent with this solution.) The concern is the situation where the protocol state is, in fact, stable after receipt of ObjectDescriptorRemove or ES_DescriptorRemove. The source node, in this case, transmits a fictitious ObjectDescriptorUpdate without object descriptors inside, or transmit a fictitious ES_DescriptorUpdate without stream descriptors inside.

b) Add an explicit bit to each message with the semantics that if the value is true the protocol state is complete. (This is, to the author, the least attractive solution. If the design is to adopt reasonable alignment rules, the second solution consumes less bits.)

c) Exploit the subclass feature of Syntax Description Language to distinguish between a class where the protocol state is transient, versus a class where the protocol state is complete. For example if the Class Tags for the current message set is:

- ObjectDescriptorUpdate ClassTag=1
- ObjectDescriptorRemove ClassTag=2
- ES_DescriptorUpdate ClassTag=3
- ES_DescriptorRemove ClassTag=4

The solution would extend these classes with four classes with subclasses the receipt of which confirms that the protocol state is complete:

- ObjectUpdateConfirm extends ObjectDescriptorUpdate ClassTag=5
- ObjectRemoveConfirm extends ObjectDescriptorRemove ClassTag=6
- StreamUpdateConfirm extends ES_DescriptorUpdate ClassTag=7
- StreamRemoveConfirm extends ES_DescriptorRemove ClassTag=8

Since the signatures already includes the ClassTag field, the subclass solution does not consume extra bits. The solution still requires a state machine, which would be comparable to the design found in the appendix.

d) Provide explicit open() and close() functions. The semantics of open() are that the protocol state is transient; the semantics of close() are that the protocol state is complete. The semantics are clear. In addition the functions anticipates the situation where either multiple source nodes can set() the protocol state, or multiple target nodes can get() the protocol state. The functions anticipate transition semantics.

6.3 Extensibility

Subclause 7.3.2.2 (Major Technical)

6.3.1 Comment

The data objects which the four message objects contain are not extensible. With the exception of the four classes which constitute the message set, the current design collapses the protocol state into a single monolithic structure. This frustrates its evolution. For example, assume that it is found essential, after the release of the protocol, to add a boolean to the ObjectDescriptor class. The
implication is that *all* the fields which follow shift one bit, which will cause all the devices which expect
the old protocol is break.
The conclusion is that, if the schema of the data objects was to evolve, the design must version the *entire
protocol*. Because the design collapses multiple data objects into a single structure, change to a single
element changes the entire message.

6.3.2 Recommendation
The concepts which provide extensible design are found in the current protocol. First, the current design
divides the state into multiple small grain data objects. This makes the design clear, in that a specific class
describes specific state. Second, the four classes which constitute the message set adopt the Class Tag
construct. Since these classes provide a class tag, the first field of the class describes itself, that is declares
the class name for the fields which follow.
The solution which would cause the design to be extensible is to exploit the ClassTag construct of Syntax
Description Language for *all* the data objects. (The appendix, "Compact Object Descriptor Protocol"
describes a solution.) The concept is to preserve the scope of the current design, preserve the structure of
the current design, preserve the data objects of the current design, preserve the fields of the current design,
and preserve the field sizes of the current design. The difference is that the fine grain data objects, which
the current design isolates as distinct classes, become visible.
The software which parses the protocol would exploit the fact that the first field for each object is its
ClassTag field. The field encodes its class name. The software can treat the field as an index to a dispatch
table, which directs execution to code which understands the precise signature for its class.
The implication for evolution is this: Because each data object describes itself, a change to the data object
isolates to just the data object. It does not shift fields of other data objects.
There are two situations to anticipate. The first situation is where a future generation design adds a data
object. The software which expects the current generation design can detect this as a Class Tag which it
does not recognize. If the new class also describes its size, the software can skip its fields and advance to
the next data object. It becomes feasible to transmit a single (next generation) protocol which previous
generation designs will understand, and which isolates changes to just the data objects which evolve.
The second situation is where a future generation design subclasses a current generation class. If the design
reserves certain Class Tag values for this evolution, the current generation designs can detect the subclass.
If the rule is that a subclass must append fields to its superclass, the current generation design can parse
the old fields, and skip the extension fields.
The conclusion is that, with simple precautions which build on the concepts of the current design, it is
feasible to design an evolvable protocol.

6.4 Compliance
(Technical)

6.4.1 Comment
The semantics of certain data objects which the four message objects contain are not clear. It is not obvious
what constitutes compliance.
The scope of the protocol is extensive. The heart of the protocol relates to certain interoperation state. This
includes a) the object descriptors bound to the session, b) the stream descriptors bound to the object
descriptors, c) the adaptation configuration state, and d) the decoder configuration state.
The other fields are subtle. These fields describe the *semantics* of the session. Because these fields
describe semantics, it is difficult to characterize compliance.
6.4.2 Recommendation

Improve the semantics. The specific examples are:

a) Object Descriptor Extension: The expectation is that this is a private field which a target node is free to ignore.

b) Stream Descriptor Profile and Level Indication: The expectation is that this field will encode the profile and level for the session, when the formalization of these is complete. The specification would contain a cross-reference to the Profile and Level section.

c) Stream Descriptor Extension: The expectation is that this is a private field which a target node is free to ignore.

d) StreamType Class: StreamType Field: The expectation is that this field encodes a value within a content name space. Given the name the target node can select the decoder to bind to the stream. (If the value was cast as a Class Tag, it becomes feasible to define decoder classes with configuration state specific to the decoder.)

e) StreamType Class: StreamInfo Fields: The expectation is that this is a private field which a target node is free to ignore.

f) Intellectual Property Descriptor: Information Data Set: The expectation is that this is a private field which a target node is free to ignore.

g) Quality of Service Descriptor: StreamPriority: The expectation is that the scope of the value is the session. It describes a priority specific to this session, not a global priority across sessions. The expectation is that a value of n+1 is greater priority than a value n. The specification is silent on conformance; the value is informative.

h) Quality of Service Descriptor: Quality of Service Qualifier: The expectation is that this is a private field which a target node is free to ignore.

6.5 Robustness

(Major Technical)

6.5.1 Comment

Because the scope of the protocol is extensive, the protocol is complex. If a protocol is complex, it is difficult to comprehend, and difficult to implement, which increases the prospect the implementation at the source node or the implementation at the target node will contain flaws. If the protocol is complex, it is difficult to parse. If it is difficult to parse, it is less efficient. If it is difficult to parse, it is less robust. It is difficult to detect bit stream violations. It is difficult to recover from bit stream violations.

The objective should be that simple scenarios remain simple. It is laudable that the protocol scales to complex scenarios, but simple simple scenarios should remain simple. The protocol, in part, reflects this requirement. The protocol allows certain data objects to be options.

The concern is that the software which decodes the bit stream still must understand these options and parse booleans which encode whether the fields follow. If the option is present, the software which decodes the bit stream is often free to ignore it. But it must detect it, and parse it, before it can discard it.

The concept which simplifies the protocol, and causes it to be scalable, is found in the current design. It is *inclusion.* The current design adopts the concept for object descriptors, which the session contains, and for stream descriptors, which the object descriptors contain. These is no boolean to parse. The source node links data objects into a state graph. The target node traverses the state graph. If the option is present, it detects a link. The Class Tag of the instance to which the link refers encodes the data object class. If the option is not present, the target node detects no link. In other words, it encounters *just* the objects which the scenario requires.
6.5.2 Recommendation

The solution is to extend the concept to the data objects below the stream descriptors. Just as the current design, in essence, graft(s) and prune(s) object descriptors to the session, and graft(s) and prune(s) stream descriptors to object descriptors, the design would graft() and prune fine grain data objects to stream descriptors. This results in simple protocol for simple scenarios. It also anticipates future profiles. If a profile was to decide that a field in the current design was to be an option, the protocol must change, unless it contains a boolean for the field. (This concern, in fact, perhaps explains the multiple booleans of the current design.) The adoption of inclusion finesse the question. The protocol for a profile would contain just the data objects which are valid. (The appendix describes a "Compact Object Descriptor Protocol" which adopts inclusion.)

The objection to inclusion might be that it requires more bits than the boolean. This is not, in practice, correct. Consider the example below. The current design template is:

```c
class DataObject1 : uint(8) ClassTag1=1 {
  // Data Objects which must be present.
  // Data Object which is optional.
  uint(1) aFlag;
  if (aFlag) {
    DataObject2 aDataObject2;
  };
```

The complication is that the boolean mis-aligns the fields which follow. This frustrates efficient execution. To solve this the declaration becomes:

```c
class DataObject1 : uint(8) ClassTag1=1 {
  // Data Objects which must be present.
  // Data Object which is optional.
  uint(1) aFlag;
  uint(7) aPad;
  if (aFlag) {
    DataObject2 aDataObject2;
  };
```

For the case where the option is present, the result is the same bit count, eight bits, as the inclusion solution, which requires eight bits for the Class Tag field. For the case where the option is not present, the result eight bits more than the inclusion solution.

6.6 Alignment

(Major Technical)

6.6.1 Comment

The fields of certain data objects fail to adopt reasonable alignment rules. The examples include:

6a) Elementary Stream Descriptor: The current design is:

```c
class ES_descriptor {
  uint(5) ESNumber;
  uint(1) StreamDependence;
  if (StreamDependence) {
    uint(5) DependsOnESNumber;
  };
  uint(1) URLFlag
  // The cumulative length is either 7 0r 12 bits.
  if (URLFlag) {
    uint(8) URLLength;
    uint(8*URLLength) URLString;
  } else {
    // The cumulative length is either 7 or 12 bits.
    uint(1) ExtensionFlag;
    // The cumulative length is either 8 or 13 bits.
    uint(8) ESOBJECTProfileLevelIndication;
    DecoderConfigDescriptor aDecoderConfigDescriptor;
    ALHeaderConfigDescriptor aALHeaderConfigDescriptor;
    IPIDescriptor aIPIDescriptor;
    QoSDescriptor aQoSDescriptor;
    if (ExtensionFlag) {
      uint(8) DescriptorLength;
```
6.6.2 Recommendation
The solution depends on whether data objects adopt a consistent design template which exploits inclusion. (See previous comments.) If the decision is to not adopt inclusion, then the best which can be done is shown below. (Observation: Don’t use multiple conditionals.)

```c
class ES_descriptor {
    uint(5) ESNr;
    uint(1) StreamDependence;
    if (StreamDependence) {
        uint(5) DependsOnESNumber;
    } else {
        uint(5) reserved;
    }
    // The cumulative length is 11 bits.
    uint(1) URLFlag
    // The cumulative length is 12 bits.
    uint(4) reserved;
    // The cumulative length is 16 bits.
    if (URLFlag) {
        uint(8) URLLen;
        uint(8*URLLen) URLString;
    } else {
        uint(1) ExtensionFlag;
        uint(7) reserved;
        // The cumulative length is modulo(8) bits.
        uint(8) ESObjectProfileLevelIndication;
        DecoderConfigDescriptor aDecoderConfigDescriptor;
        ALHeaderConfigDescriptor aALHeaderConfigDescriptor;
        IPIDescriptor aIPIDescriptor;
        QoSDescriptor aQoSDescriptor;
        if (ExtensionFlag) {
            uint(8) DescriptLength;
            uint(8*Descriptor) ExtensionDescriptor;
        }
    }
};
```

6b) Intellectual Property Descriptor: The current design is:
```c
class IPI_Descriptor {
    uint(1) IPI_Pointer;
    // The cumulative length is 1 bit.
    if (!IPI_Pointer) {
        uint(16) IPI_ES_Id;
    } else {
        uint(8) IPI_Len;
        uint(8*IPI_Len) IP_InformationDataSet;
    }
};
```

6.6.3 Recommendation
The solution depends on whether data objects adopt a consistent design template which exploits inclusion. If the decision is to not adopt inclusion, then the best which can be done is to add pad bits as shown below:
```c
class IPI_Descriptor {
    uint(1) IPI_Pointer;
    uint(7) reserved;
    // The cumulative length is 8 bits.
    if (!IPI_Pointer) {
        uint(16) IPI_ES_Id;
    } else {
        uint(8) IPI_Len;
        uint(8*IPI_Len) IP_InformationDataSet;
    }
};
```

6c) Intellectual Property Information: The current design is:
```c
class IP_InformationDataSet {
    uint(2) Compatibility;
```
uint(1) TypeOfContentPresent;
// The cumulative length is 3 bits.
if (TypeOfContentPresent) {
    uint(8) TypeOfContent;
};
uint(1) ContentIdentifierPresent;
// The cumulative length is 4 or 8 bits.
if (ContentIdentifierPresent) {
    uint(8) TypeOfContentIdentifier;
    uint(8) LengthOfContentIdentifier;
    uint(8*LengthOfContentIdentifier) ContentIdentifier;
};
uint(8) NumberOfSupplementalContentIdentifier;
if (NumberOfSupplementalContentIdentifier > 0) {
    uint(24) LanguageCode;
    for (i=0;i<NumberOfSupplementalContentIdentifier;i++) {
        uint(8) LengthOfTitle;
        uint(8*LengthOfTitle) SupplementalContentTitle;
        uint(8) LengthOfValue;
        uint(8*LengthOfValue) SupplementalContentValue;
    }
};
}

6.6.4 Recommendation
The solution depends on whether data objects adopt a consistent design template which exploits inclusion. If the decision is to not adopt inclusion, then the best which can be done is to add pad bits as shown below:

class IP_InformationDataSet {
    uint(2) Compatibility;
    uint(1) TypeOfContentPresent;
    uint(5) reserved;
    // The cumulative length is 8 bits.
    if (TypeOfContentPresent) {
        uint(8) TypeOfContent;
    }

    uint(1) ContentIdentifierPresent;
    // The cumulative length is 8 or 16 bits.
    if (ContentIdentifierPresent) {
        uint(8) TypeOfContentIdentifier;
        uint(8) LengthOfContentIdentifier;
        uint(8*LengthOfContentIdentifier) ContentIdentifier;
    }
    uint(8) NumberOfSupplementalContentIdentifier;
    if (NumberOfSupplementalContentIdentifier > 0) {
        uint(24) LanguageCode;
        for (i=1;i<NumberOfSupplementalContentIdentifier;i++) {
            // Note correction of for {i=1,,} initial value.
            uint(8) LengthOfTitle;
            uint(8*LengthOfTitle) SupplementalContentTitle;
            uint(8) LengthOfValue;
            uint(8*LengthOfValue) SupplementalContentValue;
        }
    }
};

6.7 Inconsistent Alignment
(Technical)

6.7.1 Comment
The alignment rules of the object descriptor message set versus stream descriptor set are inconsistent. The remove() function for the object descriptor packs the ten bit fields together:

class ObjectDescriptorRemove : uint(8) ObjectDescriptorRemoveTag=2;
// The above declaration includes the missing class tag value ^^^
// Just those fields which relate to the alignment question are shown.
for (j=0;j<OD_Count;j++)
    uint(10) ObjectId;
};
// The declaration which is more compact would be:
The remove() function for the stream descriptor includes pad bits:

```c
class ES_Descriptorremove: uint(8) ES_DescriptorRemoveTag=4;
// The above declaration includes the missing class tag value ^^^
// Just those fields which relate to the alignment question are shown.
for(j=0; j<StreamCount; j++) {
    uint(5) ES_Number;
    uint(3) aPad;
};
```

### 6.7.2 Recommendation

Decide which alignment rule to apply. The speculation is that, since neither the object descriptor code (ten bits) nor the stream descriptor code (five bits) are modulo(8), the intent was to pack the bits within the sequences.

### 6.8 Inconsistent Field Size

**(Technical)**

#### 6.8.1 Comment

The object count fields of the ObjectDescriptorUpdate and ObjectDescriptorRemove messages are inconsistent.

The declaration for the ObjectDescriptorUpdate class is:

```c
class ObjectDescriptorUpdate: uint(8) ObjectDescriptorRemoveTag=1;
// The above declaration includes the missing class tag value ^^^
// Just those fields which relate to the inconsistent assignment are shown.
    uint(8) ObjectCount;
};
```

The declaration for the ObjectDescriptorRemove class is:

```c
class ObjectDescriptorRemove: uint(8) ObjectDescriptorRemoveTag=2;
// The above declaration includes the missing class tag value ^^^
// Just those fields which relate to the inconsistent assignment are shown.
    uint(6) ObjectCount;
};
```

It is curious that neither the uint(8) ObjectCount nor the uint(6) ObjectCount are consistent with the Object Descriptor size, which is uint(10).

#### 6.8.2 Recommendation

Assign consistent field sizes.

### 6.9 Reset Function

**(Major Technical)**

#### 6.9.1 Comment

It should be feasible to reset a) the session context, b) specific object descriptor contexts, or c) specific stream descriptor contexts.

There are scenarios where, while the source node(s) attempts to shadow the target nodes) context, the contexts might differ. The scenarios include a) broadcast where it is not feasible for the target node to confirm its state, b) situations where corruption of the Object Descriptor Protocol is possible, or c) situations where the target node joins the session after the session has begun.

The protocol should provide a message set which resets the session context, object descriptor context, or stream descriptor context, so as to later define the current valid state.
6.9.2 **Recommendation**
Provide the reset function(s).

6.10 **Defaults**
*(Technical)*

6.10.1 **Comment**
The specification should clarify defaults for those data objects for which defaults are plausible.
It is essential that the session context, object descriptor context, and stream descriptor context reset to
default values after a) the session begins, b) the session is reset, c) object descriptor sequences are reset,
or d) stream descriptor sequences are reset. The design becomes more robust.

6.10.2 **Recommendation**
Assign default values.

7. **Annex C: Audio Source**

[note: This comment addresses issues in both CD 14496-1 (Systems) and CD 14496-3 subpart 5
(Audio/Structured Audio). The comment should be addressed in both subgroups.]

**Comment:**
As it currently stands, sample-based synthesis can only be coded efficiently in MPEG-4 under a particular
set of constraints: when the set of samples to be used is appropriate for encoding in the Structured Audio
Sample Bank Format. In order for this to be the case, the samples must represent a well-spaced, carefully
configured group, each critically sampled, which together represent a subset of the sounds realizable by an
instrument.

However, many samples used in sample-based synthesis do not fall into this category. For example, other
kinds of sounds very desirable to transmit as samples are: drum loops, environment sounds such as bird
songs, crowd noise, noisy textures such as traffic noise, samples of existing songs, and many others.
Currently, there is no provision for transmission of these samples in MPEG-4 except as inclusion as flat
(uncompressed) PCM files in the Structured Audio bitstream. This is inefficient and silly, since MPEG-4
contains the state-of-the-art suite of tools for natural sound compression.

It is highly desirable to allow the coding of sound samples with tools such as AAC or the Parametric coder;
to do so will reduce the size of many Structured Audio bitstreams by a factor of 10 or more, with no loss of
quality.

**Solution (overview)**
Allow the **AudioSource** BIFS node to take children, in the special case when the ES pointed to by the node
is a Structured Audio stream. Only **AudioClip** nodes may be used as children (although they in turn will
have children). The Structured Audio decoder instantiated by the **AudioSource** node then is allowed
access to the buffered audio data in those **AudioClip** nodes for use as sound samples.

To provide this access in the Structured Audio decoder, the parameter to the **sample** core wavetable
generator may be an integer rather than a symbol. If it is an integer, then one channel of audio data from
the children AudioClip nodes is placed in the wavetable. If it is a symbol, then the sample data is contained in the Structured Audio bitstream as usual.

In this solution, no new BIFS nodes are required; one new field is required in the AudioSource node; normative semantics regarding this new field are required, especially regarding this new form of interconnection between the BIFS and Structured Audio decoders; an expansion of the syntax of the sample core wavetable generator is required; and normative semantics regarding this expanded syntax are required. These aspects are provided below.

Suggested revisions:

8. In Systems CD (new text in *italics*)

8.1.1.1.1.1.1 7.2.5.1.2.4.1 Semantic Table

AudioSource {
  exposedField MFString url NULL
  exposedField SFFloat pitch 1
  exposedField SFFloat startTime 0
  exposedField SFFloat stopTime 0
  field MFNode children NULL
  field SFInt32 numChan 1
  field MFInt32 phaseGroup NULL
}

8.1.1.1.1.1.2 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.

8.1.1.1.1.1.3 7.2.5.1.2.4.3 Detailed Semantics

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 \( \text{phaseGroup}[i] = \text{phaseGroup}[j] \) then channel \( i \) and
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 Error! Reference source not found.). The children field allows buffered AudioClip data to be used as sound samples within a Structured Audio decoding process. Only AudioClip nodes shall be children to an AudioSource node, and only in the case where url indicates a Structured Audio bitstream.

### Calculation

8.1.1.1.1.1.4 7.2.5.1.2.4.4

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.

If AudioClip children are provided for a Structured Audio decoder, the audio data buffered in the AudioClip(s) must be made available to the decoding process. See CD 14496-3 Subclause 5.6.2.

### In Audio CD Subpart 5

9.1 5.6.2 Sample

tl table(sample, size, which[, skip])

The sample core wavetable generator allows the inclusion of audio samples (or other blocks of data) in the bitstream and subsequent access in the orchestra.

If size is −1, then the size of the table shall be the length of the audio sample. If size is given, and smaller than the length of the audio sample, then the audio sample shall be zero-padded at the end to length size. If size is given, and longer than the length of the audio sample, only the first size samples shall be used.

The which field identifies a sample. It is either a symbol, in which case the generator refers to a sample in the bitstream, by symbol number; or a number, in which case the generator refers to a sample stored as an AudioClip in the BIFS scene graph (ISO 14496-1 Section XXX).

In the case where the generator refers to a sample in the bitstream, for compliant bitstream implementations, the sample data is simply a stream of raw floating-point values. This sample block of data shall be placed in the wavetable. If the bitstream sample data block contains sampling rate, loop start, loop end, and/or base frequency values, these parameters of the wavetable shall be set accordingly. If the sampling rate is not provided, it shall be set to the orchestra sampling rate by default. Any other parameters not so provided shall be set to 0.

In the case where the generator refers to a sample stored as an AudioClip, other audio coders described in this Part of ISO 14496 may be used to compress samples. The children fields of the AudioSource node responsible for instantiation of this orchestra refer to AudioClip nodes. Each AudioClip contains, after buffering as described in ISO 14496-1 Section XXX, several channels of audio data. If the first child has \( n_1 \) channels, the second \( n_2 \) channels, and so forth up to child \( k \), then this AudioSource node has \( K = n_0 + n_1 + \ldots + n_k \) channels in all, and which shall be a value between 0 and \( K \). Channel which (where which is rounded to the nearest integer if necessary), numbering in order across children and their channels, shall be placed in the wavetable. The sampling rate of the wavetable shall be set to the
sampling rate of the AudioClip node from which channel which is taken. The loop start, loop end, and base frequency values shall be set to 0.

If the isReady flag of the selected AudioClip node is not set when the generator is executed, then the bitstream is in error. That is, this form of this generator must only be used in cases where there is time allotted in the bitstream for the other decoders to produce samples (in real-time) before the generator executes. This is likely done by including the table generator in a score line scheduled to execute after the Presentation Time Stamp (see ISO 14496-1 Section XXX) of the last audio Access Unit needed in the AudioClip node.

For standalone systems such as authoring tools, implementors are encouraged to provide access to other audio file formats and disk file access using this field (for example, to allow a filename as a string constant here). However, the only normative aspect is that in which the tokenized bitstream element for the generator refers to a sample element in the bitstream.

If skip is provided and is a positive value, it is rounded to the nearest integer, and the data placed in the wavetable begins with sample skip+1 of the bitstream or AudioClip sample data.