Title: **Proposed Technical and Editorial USNB Comments on the MPEG-4 Systems CD (N1901)**
Source: **Alexandros Eleftheriadis, Columbia University**
Group: **NCITS L3.1**
Date: **January 26, 1998**
Number: **L3198-17**

1. Introduction

This document provides a list of technical and editorial problems that have been identified in the MPEG-4 Systems CD (N1901, 21 November 1997). The comments are organized by Clause number, and are characterized as technical or editorial.

2. List of Comments

2.1 TECHNICAL

2.1.1 Clause 7.1.2.9 Composition Memory

It is stated that the size of composition memory is not normatively specified. This should be cross-checked with profile/level definitions.

2.1.2 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, described in Clause 7.2.3.1.9.9."

In addition, 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. We recommend that it is explicitly stated that such data is raw formatted, i.e., they are not AL encapsulated. Doing otherwise would require a protocol different from TCP to transport the data, which would be a totally inappropriate use of the existing Web infrastructure.

For object descriptor URLs (via the ES descriptor field of object descriptors), the specification indicates that access is performed to an AL-packetized field with, or without, a preceding object descriptor. Such AL-packetized format, however, is not defined (again, transport *must* be performed using TCP to allow regular HTTP servers to be used). Clause 7.3.4.3 (Usage of URLs in Object Descriptors) fails to address these issues. We have defined a simple such structure as a contribution to the San Jose WG11 meeting, and we also attach it as an appendix to this document for inclusion as a recommended solution from the USNB.

2.1.3 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. This is in

line with traditional “predictive” coding techniques. Note also that ObjectDescriptor streams follow this approach as well.

2.1.4 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 purpose of this construct was to allow robustness by repeating scene information periodically (especially useful for broadcast channels). In order to accommodate this, an additional update command must be included in Clause 7.2.3.2.2 (Update Command Syntax). Note that the field is already 3 bits.

The actual syntax can be identical to ‘Scene Replacement’ (Clause 7.2.3.2.5.5), with the exception of the time semantics. If the repeat refers to the desired state of the tree at the time indicated by the timestamp of this command, then clearly no modification is necessary to the ‘SceneReplace’ syntax and can be duplicated as is. If, however, it is a repeat of the original BIFS tree, or an update thereof, both the syntax and semantics become more complex. Repetition command must be specified for each updateable component. Furthermore, time reference information must be included to indicate which update they repeat.

Both approaches have advantages and disadvantages. We defer the matter for discussion within NCITS.

2.1.5 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 concluded 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.

Also, BIFS-Anim streams may or may not be AL encapsulated. When they are accessed via a BIFS URL, they are not; when they are accessed via an Obect Descriptor they are (even if it is a URL). See comment below for URL use. Note that not including AL encapsulation is not a problem, as BIFS-Anim structures already contain timing information.

2.1.6 Clause 7.2.3.1.2 BIFSNodes

BIFSNodes can be considered a regular BIFS node. However, the current specification uses a ‘hasWorldInfo’ to determine if a ‘WorldInfo’ node is included or not. We believe that this unnecessarily breaks the regularity of BIFS node parsing, and we recommend that the definition of BIFSNodes should be no different than any other BIFS node.

2.1.7 Clause 7.2.3.1.3 BIFSNodes

The SFNode syntax uses the flag ‘predefined’, presumably to allow for the inclusion of non-predefined nodes. However, the syntax provided does not support the inclusion of data that relates to such non-predefined node and hence is incorrect.

We recommend that the SFNode syntax is changed as follows:

```
bit(1) predefined;
if (predefined) {
      ...   // same as before
}
else {
      // parse custom node data
      unsigned int(8) length;
```

```
        while (length>0) {
              unsigned char(8) data[length];
              unsigned int(8) length;
        };
  }
```

2.1.8 Clause 7.2.3.1.3 SFNode

Note that using node pointers via the ‘isReused’ option may create infinite loops. It is not stated if such loops are allowed or not. Note that due to the practically finite resolution of the display, such loops could be allowed (e.g., to create the effect of double mirrors). We recommend that it is explicitly stated that reusing a parent in a child node is not allowed.

2.1.9 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 preceeds the fields, i.e.:

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

2.1.10 Clause 7.2.3.1.9.4 SFImage

SFImage is not among the fundamental BIFS types, and it should be removed.

2.1.11 Clause 7.2.4.1.10 QuantizedField

Quantization of fields with infinite ranges using quantization code 13 is improperly defined. It only used as additional information the number of bits to use for uniform quantization. Since the dynamic range is infinite, this cannot be used. Explicit finite ranges must be used for the pertinent fields. However, their inclusion in the syntax itself defeats the whole purpose of quantization (compression). One could use the min and max values defined in the node coding tables, but for ‘float’ values this range will be huge. Hence a coding scheme with, say, 8 bits will be useless.

We suggest to the NCITS to consider proposing the removal of all quantization-related syntax from the BIFS specification. The current specification is broken, and fixing it seems to require major changes. In addition, the benefit in terms of coding efficiency are dubious. Finally, decoder complexity will be reduced as well.

Note that IM1 claims quantization support. These problems, however, have not been identified through the implementation, which is a cause for concern regarding claims made about the status of the reference software.

2.1.12 Lack of Server-Based Interaction

The current specification completely lacks the specification of the syntax of messages that may be transmited from the terminal back to the server, even though interaction is a key requirement of the MPEG-4 specification. The establishment of such “backchannel” connection is addressed in the DMIF specification. The Systems specification, however, should detail the precise format of these messages and

the mechanisms with which they can be triggered from the scene. We attach as an annex a contribution to the San Jose WG11 meeting, and we recommend that is adopted as a USNB recommended solution.

2.1.13 Lack of Extensibility

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

It is possible to rectify this by:

1) Making all NType have the same length (at least 7 bits).

2) Refer to all nodes by a single number (their SFWorldNode code).

In order to still capture the SFNode child field context, node data type tables can still be used.

We encourage NCITS to consider allowing this extensibility, to avoid neglect of Version 1 by the marketplace if a Version 2 is being worked on.

2.2 EDITORIAL

2.2.1 Table of Contents

Entry for item ‘InitialAnimQP’ (Clause 7.2.3.3.1.4) has a preceding ‘}’.

2.2.2 Clause 0.1

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

2.2.3 Annex A

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

2.2.4 Clause 7.2.1.1 Scope

Figure 7-3 included references to “hierarchically multiplexed” streams. This is a remnant of earlier versions of the specification. Streams are not hierarchically multiplexed, and hence this attribute should be removed from the figure.

2.2.5 Clause 7.2.1.3.1 Grouping of Objects

The MPEG-4 scene description is described as a “Directed Acyclic Graph.” This should be replaced by simply “tree.”

2.2.6 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”.

2.2.7 Clause 7.2.2.1 Global Structure of a BIFS Scene Description

Replace “behaviours” and “kinds” in the first paragraph with “behaviors” and “types” respectively. The former also appears in sub-item 3. Further single-word errors are not identified in this contribution, under the assumption that their correction is at the Editors’ discretion.

2.2.8 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).

2.2.9 Clause 7.2.2.12 Internal, ASCII and Binary Representation of Schenes

Reference to ‘MPEG-4’ in the first sentence should be replaced by “This Committee Draft of International Standard”.

2.2.10 Clause 7.2.2.13.3 Requirements on BIFS elementary stream transport

Reference to “Working Draft” should be replaced by “Committee Draft”.

2.2.11 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. Inclusion of 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.

2.2.12 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, and therefore time duration, is”.

2.2.13 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”.

2.2.14 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 the following, 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.

2.2.15 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'.

2.2.16 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'.

2.2.17 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.

2.2.18 Clause 7.2.3.1.6 NodeType

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

2.2.19 Clause 7.2.3.1.7 Field

The 'fieldType' argument should be 'int fieldType'.

2.2.20 Clause 7.2.3.1.8 MFField

The 'fieldType' argument should be 'int fieldType'.

2.2.21 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 is passed an NCT structure, and that declarations of QuantizedField structures is performed as follows:

```
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.

2.2.22 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:

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

with similar constructors for IN, OUT, and DYN.

2.2.23 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.

2.2.24 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.

3. Appendix A: AL-Packetized Format

The AL PDU format specified in the systems CD does not support AL packetized streams required to support URLs in the elementary stream descriptors (clause 7.3.3.2.1 in the systems CD). Further more there is a need for a format to store and exchange AL packetized streams. Since AL PDUs do not include payload size, the AL PDUs cannot be simply concatenated to create a stream of PDUs. The stream format proposed in this contribution satisfies the requirements for an AL PDU stream.

The properties of the stream and the AL header configuration are contained in the object descriptor.

```
// Stream of AL PDUs;
class AL_PDU_Stream {

      // File identification
      char(8) magic[[0]]='M';
      char(8) magic[[1]]='P';
      char(8) magic[[2]]='4';
      char(8) magic[[3]]='P';
      char(8) magic[[4]]='D';
      char(8) magic[[5]]='U';

      ObjectDescriptor od;

      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;
      }
}
```

4. Appendix B: Server-Based Interaction

Interactivity is very prominently placed in current MPEG-4 activity. Even though a back channel is specified to support interactivity, the syntax and semantics for such interaction messages is not specified. Existing standards such as DSM-CC and RTSP support traditional VCR-type interactivity to reposition a media stream during playback. These existing specifications are not adequate, as MPEG-4 applications potentially require complex interactive control beyond the VCR-type control offered by these models. The concept of generic, CGI-like application-specific interactive messages was proposed in our earlier contribution (MPEG97/M2888) to the 41st MPEG meeting. In this contribution, we detail the syntax and semantics for the control messages for user interaction exchanged between clients and servers (the terms clients and servers are functional notations and can represent clients and servers depending on their functionality and connectivity).

Interactive messages are caused by certain user actions and sent to the server, which cause it to modify the object stream it is delivering by adding an additional object, removing an object etc. The user actions may include clicking an object, inputting a text string, etc.

Interactivity provided is very application specific and MPEG cannot possibly define the interactive behavior completely in terms of user events. To support complete application-dependent interactivity, CGI-like approach to interactivity should be adopted. User input such as clicks and data entries causes an application-specific command sent to the server (if this type of event is specified). The server can then respond typically by sending a BIFS update command (that changes the view, adds a new object etc.). This allows total freedom for supporting complete interactivity as required by the applications.

4.1 Interactivity in MPEG-4

MPEG-4 essentially uses two modes of interactivity: local, and remote. Local interactivity can be fully implemented using the native event architecture of BIFS, as well as the BIFS update mechanism for events generated by a hosting application. No additional normative support is necessary.

Remote interactivity currently consists of URLs. There are two types of URLs defined in the MPEG-4 Systems CD. The first is BIFS URLs, where a BIFS field, instead of referencing an object descriptor it actually contains a URL string. The second type is object descriptor URLs. These are actually placed in the object descriptor (ES configuration descriptor). The difference between the two is that the former refer to raw data, while the latter refer to AL-formatted data.

While a simple form of remote interactivity can be fully supported by using BIFS URLs (that invoke CGI scripts at the server), this implies that the only means of interactivity in an MPEG-4 session can occur through an IP channel, and in particular using HTTP access methods (POST/GET). For MPEG-4 sessions that want to exchange messages using FlexMux connections (non-IP connections in general), this approach cannot be used (the HTTP server will not know how to parse FlexMux PDUs).

More importantly, URLs are currently used for data retrieval and are not integrated with the event facilities. For example, the trigger of a proximity sensor cannot be sent back to the server.

It is then appropriate to define a very simple mechanism to communicate information back to the server using the existing FlexMux channels set up using DMIF. Such facility has to be integrated with the native event routing mechanism of BIFS.

The generic format for these messages can indeed be extremely simple, as the syntax and semantics of control messages are entirely left to application designers. Also, one can assume that a fully reliable

connection is present. DMIF can ensure that the proper QoS attribution is associated with the upstream control channel so that this assumption holds true. Even though one could potentially associate different QoS categories to control messages, we believe that in practice such an approach would have little use and would unnecessarily complicate the syntactic structures.

In the following we detail the syntax for the command descriptor as well as the message being sent to the server (AL encapsulated). In order to use such a command descriptor, a modification is also proposed in the ROUTE definition to allow event targets to point, in addition to BIFS nodes, to command descriptors as well. Pointing to command descriptors by reference allows the reuse of commands at different ROUTEs and times in an MPEG-4 presentation. To accommodate command descriptors, descriptor tag 0x05 from the unused pool 0x05-0xFF (see Clause 7.3.2.3, Table 7-3 of MPEG-4 CD) can be used.

4.2 Command Descriptor

Continuing in the MPEG-4 Systems methodology of using descriptors to support key functionalities, we define a command descriptor to communicate control messages. Command descriptors define a format to communicate control messages between two entities (clients and/or servers) participating in an MPEG-4 session. A command descriptor specifies a format to communicate control messages. It does not define any specific control messages (e.g., it does not specify a command format to advance a particular object in time). Such specifics are application-dependent and are left to the application designers.

4.2.1 Syntax

```
class CommandDescriptor: bit(8) commandDescriptorTag = 0x05 {

      bit(10) CommandDescriptorID;


      // stream count; number of ES_IDs associated to this message
      unsigned int (8) count;

      // ES_Id of the streams
      unsigned int (16) ES_ID[count];

      // application-defined parameters
      do {
            unsigned int (8) paramLength;
            char (8) commandParam [paramLength];
      }
      while (paramLength!=0);
}
```

4.2.2 Semantics

`CommandDescriptorID`: 10-bit value uniquely identifying this command descriptor.

`count`: number of objects that are affected by this control message.

`ES_ID:` ES_ID of the objects affected by this message. This information can be useful, for example, for simple implementation of commands such as stop, pause etc. Since the content creator inserts these values, however, their meaning can be arbitrary.

`paramLength`: length of the command parameter.

`commandParam`: application-specific parameter string.

4.3 Modification Required in ROUTE

The definition of ROUTE (Clause 7.2.3.1.12.3.1) needs to be updated as follows, in order to support command descriptors.

```
class ROUTE {
      bit(1) isUpdateable;
      if (isUpdateable)
            bit(10) ROUTEid;
      }
      bit(10) outNodeID;
      outID outFieldReference;

      bit(1) isServerRoute;
      if (isServerRoute) {
            bit(10) CDid;     // command descriptor ID
      }
      else {
            bit(10) inNodeID;
            inID inFieldReference;
      }
}
```

The modification consists of the introduction of a single-bit flag (isServerRoute), which signals the presence of a command descriptor ID, rather than the inNodeID.

Note that nothing prohibits an event to be routed both to a local node, as well as to the server.

4.4 Server Command

This documents the syntax and semantics of the command, as it is transmitted back to the server. Transmission is performed using AL packetization. This allows timestamping to be used (if the AL is configured as such using DMIF facilities when setting up the back channel(s)).

4.4.1 Syntax

```
class Commmand {

      // which node generated this message
      unsigned int(16) nodeID;

      // the following are simply copied from the command descriptor
      // stream count; number of ES_IDs associated to this message
      unsigned int (8) count;

      // ES_Id of the streams
      unsigned int (16) ES_ID[count];

      // application-defined parameters
      do {
            unsigned int (8) paramLength;
            char (8) commandParam [paramLength];
      }
      while (paramLength!=0);
}
```

4.4.2 Semantics

`nodeID`: The nodeID of the node associated with the command. This field must be filled by the MPEG-4 terminal based on the node that directly pointed to the command descriptor.

All other fields are simply copied from the command descriptor.

4.5 Comments

This particular structure does not communicate the field value for the event at hand. Such functionality can be easily added, by including the inFieldReference as well as the value for the field that generated the event.