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

ISO/IEC JTC1/SC29/WG11
MPEG97/2536
Stockholm, Sweden

Title: **Stored File Format for MPEG-4 (Rev. 2.0)**

Source: **Alexandros Eleftheriadis and Hari Kalva, Columbia University, USA**
Atul Puri and Robert Schmidt, AT&T Research, USA

Group: **MPEG-4 Systems**
Date: **July 15, 1997**

1. Introduction

The MPEG-1 and MPEG-2 standards specify bitstream representations for Audio, Video and Systems. Although they are intended to support interactivity, the flexibility they offer is rather limited. Among the reasons for these limitations is the lack of a stored file format defined by MPEG. Since MPEG-4 offers opportunities for much more flexibility in terms of types of audiovisual objects, editing of objects, composition of objects and user controlled presentation, it seems even more necessary to standardize a flexible and extensible file format. In other words, defining a standardized MPEG-4 file format will facilitate the following:

- Random Access to AV Objects
- Editing of AV Objects - insertion, deletion, modification
- Random Access to Composition Data
- Editing of Composition Data
- Unified representation of MPEG-1, MPEG-2: audio, video and systems data files, and coded MPEG-4 data

The proposed file format is portable and independent of the platform or the storage media used (e.g., DVD or hard disk or CD-ROM). The format specifies a way to segment and index encapsulated AV objects to help random access. All AV objects and BIFS information are encapsulated in adaptation layer (AL) PDUs.

2. MPEG-4 File Format Overview

MPEG-4 files can use .mp4 as the format-identifying extension. All stored AV Objects related to a session and conforming to MPEG-4 reside in one or more such files. A session does not need to be contained in only one file; a set of files can be used to form a complete session, with one of them acting as the master file. Other objects (hereafter referred to as 'logical objects' or 'remote objects') are referred to from the master (or other) files using URLs. These objects can be on a different file on the same file system or in a remote file system.

A pictorial overview of the proposed format is shown in Figure 1. A detailed description of the syntax is provided at the end of this overview (Section 3).

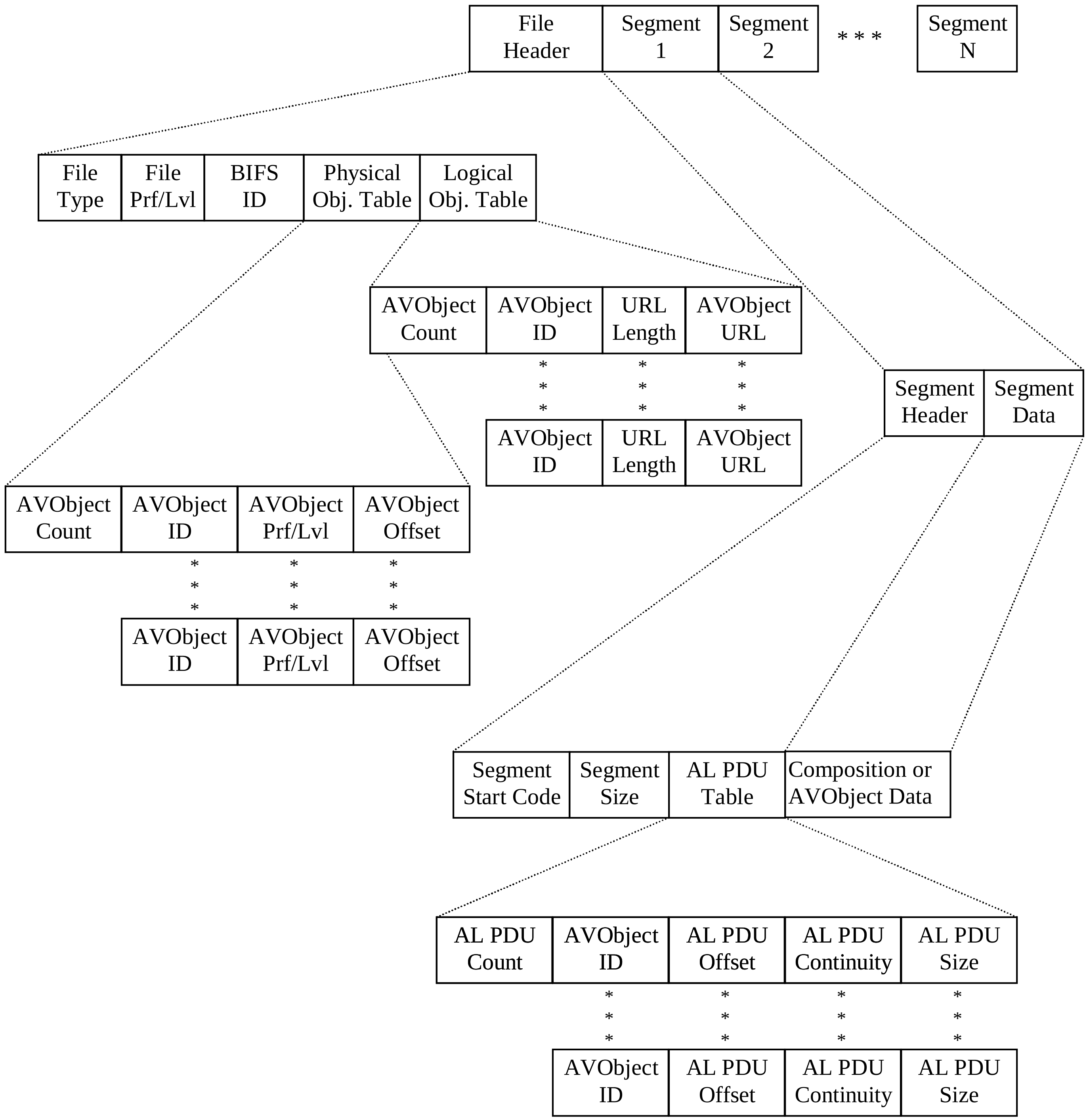

Figure 1: File format structure

An MPEG-4 file consists of a file header containing global information about the AV objects contained, followed by an arbitrary number of segments containing the AV object and BIFS data. This data is carried in the form of AL PDUs, per the MPEG-4 FlexMux specification.

The AL PDUs are interspersed within "file segments." Each segment contains a header describing the AL PDUs located within that segment. The MPEG-4 file thus contains AL PDUs multiplexed and indexed such that random access of individual objects (encapsulated in AL PDUs) is possible. To stream a file for playback, the index information (object tables) is removed and AL PDUs are delivered. The proposed format builds a layer on top of the adaptation sub-layer of the FlexMux layer to index the AL PDUs by object number. In the absence of this indexing information, random access becomes practically impossible. Note that a segment can contain part of an AL PDU, an entire AL PDU, or even more than one AL PDU. Furthermore, a segment may have unused space inbetween AL PDUs, or even be completely empty. This helps during on-line editing of large files.

File Header

The first byte of the File Header contains the file type definition. This byte describes the contents of the file according to the following table.

Bit 0 – If set indicates an streaming file (intended to denote a constrained syntax – TBD - that facilitates streaming)
Bit 1 – If set indicates that there are Physical AV Objects present in the stream
Bit 2 – If set indicates that there are Logical AV Objects present in the stream
Bit 3:7 – Reserved

Following the file type field is a 1 byte extension indicator (followed by possible extension data), and a 1 byte code describing the profile/level of the entire stream. This allows a decoder to determine if it is capable of handling the data in the file. After the file profile field is the ID used for the BIFS data. Ids are used to uniquely identify the various AV object data, including the BIFS information.

The next portion is the optional Physical Object Table, which catalogs a description of all the objects in the file that are physically present or contained in the file. The file header next contains an optional Logical Object Table, which catalogs the location of all file objects that are not physically present in the file, but are referenced via URLs. The URLs are coded as strings (without a terminating nul '\0' character), prepended by their length (using 8 bits)

Physical Object Table

The Physical Object Table is optional, and when present it is contained in the File Header. It consists of a 2 byte AV Object count, indicating the number of AV Objects in the file, followed by a 2 byte ID and 1 byte profile/level descriptions for each AV Object present in the file. Each AV Object description also contains an additional 8 bytes to indicate the offset (from the beginning of the file) to the segment in which the AV Object or BIFS information first occurs in the stream.

Logical Object Table

The Logical Object Table is also optional, and when present it is contained in the File Header. It consists of a 2 byte AV Object count indicating the AV Objects that are part of the session, but not physically present in the file. This is followed by a 2 byte AV Object ID, a 1 byte object location string length, and the string indicating the location (URL) of each AV Object in the table. The file pointed by the URL shall also be in the MPEG-4 file format (it is up to the author to ensure that the ID used exists in the remote file and is not duplicated in the local file). The concept of logical objects facilitates the use of a set of files to store an MPEG-4 presentation.

File Segment

The file is composed of one or more segments, uniquely identified by a 32 bit start code (0x000001B9). A special code denotes the end of the file (0x000001FF).

Following the segment start code is an AL PDU table. It contains a 2-byte count field, indicating how many AL PDUs are contained in the segment. For each such PDU, an 8-byte structure is used to describe the object contained. The first 2 bytes are the AV Object ID, the next 4 bytes indicate the offset to the starting point of that AL PDU in the segment. The next two bits are the 'continuity flag', and have the following meaning:

> 00 - complete PDU
> 01 - 1st segment of a split PDU; next segment follows; look in the segment tables
> 10 - Last segment of a split PDU;
> 11 - intermediate segment of a split PDU; look in the PDU table to locate the next PDU segment.

The remaining 14 bits give the size (in bytes) of the part of the AL PDU contained.

Following the AL table, we have a 4 byte segment size field which denotes the number of bytes until the beginning of the next segment start code or end-of-data code.

Random Access

Accessing an AV object at random by object number involves looking up the AL PDU table of a file segment for the object ID. If the ID is found, the corresponding AL PDU is retrieved. Since an access unit can span more than one AL PDU, it is possible that the requested object is encapsulated in more than one AL PDU. So to retrieve all the PDUs that make up the requested object, all the PDUs with the requested object ID should be examined and retrieved until a PDU with the first bit set is found. The first bit of an AL PDU indicates the beginning of an access unit.

If the ID is not found, the AL PDU table in the next segment is examined.

All AL PDU segments are listed in the AL PDU table. This also allows more than one object (instance) with the same ID to be present in the same stream segment. It is assumed that AL PDUs of the same object ID are placed in the file in their natural time (or playout) order.

Size Limitations

The file format as specified here is limited to 64K local objects and 64K remote objects. Furthermore, segments are limited to a size of 4GB. The offsets to individual objects in the physical and logical object tables limit the total size of the file to a 64-bit address space.

As shown in Figure 2, an adapter can be employed to strip the editing information from the file and indicate it as file intended for streaming.

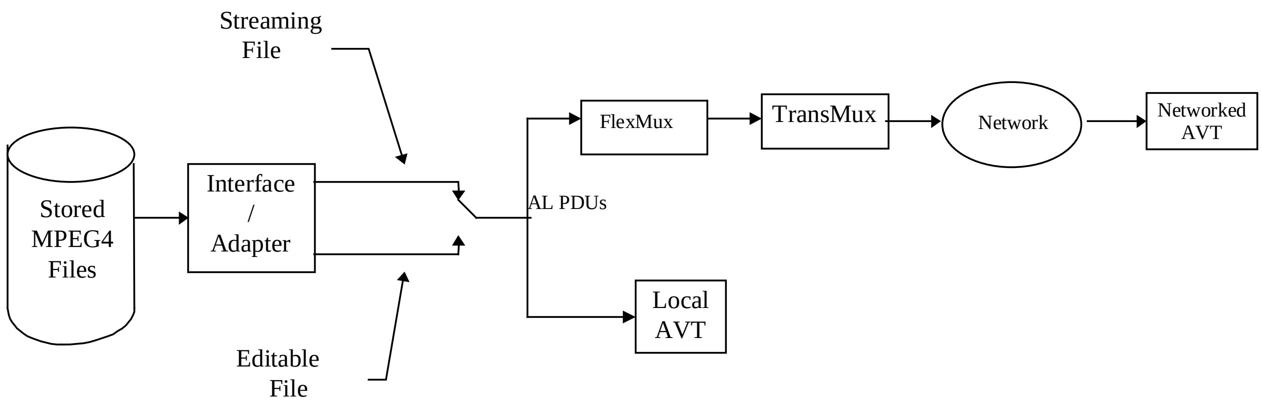

Figure 2: Serving Stored objects to Audio Visual Terminals (AVT)

3. MPEG-4 File Format Syntax

```
/*
 * $Id: mpg4file.fl 2.0 1997/07/15 21:05:17 eleft Exp eleft $
 *
 * MPEG-4 Stored File Format
 *
 * A. Eleftheriadis <eleft@ee.columbia.edu>
 * H. Kalva <hari@ee.columbia.edu>
 * A. Puri <apuri@research.att.com>
 * R. Schmidt <rls@research.att.com>
 */

// object segment start code
const int SEG_CODE = 0x000001B9;

// end-of-data code
const int EOD_CODE = 0x000001FF;

// info for physical objects (contained in file)
// one object takes one Elementary Stream (ES)
class PhysicalObjectInfo {
      bit(16) ID;          // ID (one per object - no other semantics)
      bit(4) profile; // profile/level for this object/ES only
      bit(4) level;
      unsigned int(64) firstSeg; // offset to segment where
                                 // it first appears counting
                                  //  from  the  beginning  of
the file
}

class PhysicalObjectTable {
      // how many objects contained in table
      unsigned int(16) count;
      if (count>0)
            PhysicalObjectInfo info[count];
}

// info for logical objects (not contained in file
// but referenced via URL)
```

```
class LogicalObjectInfo {
     bit(16) ID;         // ID (one per object - no other semantics)
     unsigned int(16) URLlen;
     char(8) URL[URLlen];
}

class LogicalObjectTable {
     // how many objects contained in table
     unsigned int(16) count;
     if (count>0)
          LogicalObjectInfo info[count];
}

class MPG4FileHeader {
     // magic number (the string 'MPEG4')
     char(8) magic[[0]]='M';
     char(8) magic[[1]]='P';
     char(8) magic[[2]]='E';
     char(8) magic[[3]]='G';
     char(8) magic[[4]]='4';

     // version number of file format
     char(8) version = 0;

     // general attributes (1 byte if no extension present)

     bit(1) isEditable;   // intended to signal a constrained
                          // syntax (TBD) that makes the file
                          // easy to stream
     bit(1) logicalAVO;  // set if a logical AVO is present
                          // (logical = externally referred to)
     bit(1) physicalAVO; // set if a physical AVO is present

     bit(5) reserved;

     // header extension processing
     unsigned int(8) extBytes;
     while (extBytes>0) {
          char(8) data[extBytes];
          unsigned int(8) extBytes;
     }

     // profile/level of the file's content
     unsigned int(4) profile;
     unsigned int(4) level;

     // BIFS ID (can be local or remote)
     bit(16) BIFS_ID;

     // Physical (or local) Object Table -- objects contained in file
     if (physicalAVO)
          PhysicalObjectTable pot;

     // Logical (or remote) Object Table -- objects referenced in
file
     if (logicalAVO)
          LogicalObjectTable lot;
```

```
}

// info about AL-PDU contained in a segment
class ALInfo {

      bit(16) ID;
      unsigned int(32) offset; // offset within segment
      bit(2) continuity;
            // 00 - complete PDU
            // 01 - 1st segment of a PDU
            // 10 - last segment of a PDU
            // 11 - intermediate segment of a PDU

      unsigned int(14) size; // size of PDU in bytes
}


// table of ALInfo's in a file segment
class ALTable {
      unsigned int(16) count;
      if (count>0)
            ALInfo info[count];
}


class Segment {
      // table of AL-PDUs contained in segment
      ALTable alt;

      // segment size in bytes, includes only
      // AL-PDU data space (no segment headers)
      unsigned int(32) size;

      // AL-PDU data contained (may have holes)
      char(8) data[size];
}


class MPG4File {
      // file header
      MPG4FileHeader hdr;

      // list of file segments
      bit(32) code;
      while (code!=EOD_CODE) {
            if (code==SEG_CODE) {
                  Segment seg;
            }
            bit(32) code;
      }
}
```

4. MPEG-4 File Format Software

C++ software implementing the above format can be found in the anonymous ftp site ftp://quino.ctr.columbia.edu/mpeg4/mpg4file.