INTERNATIONAL ORGANIZATION FOR STANDARDIZATION

ORGANISATION INTERNATIONALE DE NORMALISATION

ISO/IEC JTC1/SC29/WG11

CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC1/SC29/WG11

MPEG/N12126

Torino, IT, July 2011

Title	**Text of ISO/IEC 14496-12:2008 | 15444-12:2008 (3rd edition) / Amendment 4: MP4 files as a playlist, and large metadata support**
Status	**PDAM**
Source	**MPEG-4 Systems**
Editors	**Noboru Harada (NTT), Miska M. Hannuksela (Nokia), David Singer (Apple)**

Document type: International Standard
Document subtype: Amendment
Document stage: (30) Committee
Document language: E

STD Version 2.1c2

ISO/IEC JTC 1/SC 29 N

Date: 2011-09-16

ISO/IEC 14496-12:2008/PDAM 4 & ISO/IEC 15444-12:2008/PDAM 4

ISO/IEC JTC 1/SC 29/WG 11

Secretariat:

Information technology — Coding of audio-visual objects — Part 12: ISO base media file format, AMENDMENT 4: MP4 files as a playlist, and large metadata support

Élément introductif — Élément central — Partie 12: Titre de la partie

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

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

Amendment 4 to ISO/IEC 14496-12:2008 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Information technology — Coding of audio-visual objects — Part 12: ISO base media file format, AMENDMENT 4: MP4 files as a playlist, and large metadata support

<<ed: This amendment needs re-organizing into amendment style.>>

1 Introduction

1.1 General

This amendment covers a number of independent areas, presented separately here before the re-organization into amendment style, for ease of reading.

1.2 File Tracks

This Technology under Consideration document introduces a file track feature that enables using ISO Base Media Files as flexible playlists. Among other things, file tracks enable DASH (ISO/IEC 23009-1) clients record received segments into file(s) in a straightforward manner (see 2.5).

When compared to using playlist formats or presentation languages (e.g. SMIL), the file track feature may provide the following advantages:

- It uses the same parser for both the containing file and the contained files. Hence, when the playback switches from one contained file to another, it is less likely that the playback suffers from interrupts or glitches.
- It provides powerful mapping of contained files onto the presentation timeline of the containing file by using edit lists.

1.3 Handling large numbers of meta-data items

ISO/IEC 23000-6 Professional Archival Application Format (PA-AF) is one of the MPEG-A standards. It has been used in several archiving system in Japan. For example, a Japanese record label applied the PA-AF in their preservation system.

It was found that the number of files can be archived in a PA-AF file was limited to 65536 and it was not enough for some applications. This limitation was caused by the current ISO base media file format standard specification because the number of bits assigned for item_ID and item_count are 16 bits.

In order to enable users to archive larger number of files exceeding 65536 files in a PA-AF package, the number of bits assigned for item_ID and item_count should be updated to 32 bits.

1.4 Movie Fragment Related

It is recognized that clients could use the Composition to Decode box (see ISO/IEC 14496-12:2008/Amd.1) to find out the required decoded picture buffering delay. However, the use of the Composition to Decode box with movie fragments is unspecified in Amendment 1.

It is also realized that the alternative startup sequence sample grouping (see ISO/IEC 14496-12:2008/Amd.1) could be used to even out differences in the decoded picture buffering delay between Representations. If the buffering requirements of the switch-to Representation are greater than those of the switch-from Representation, the use of alternative startup sequence comes with the cost of unequal video frame rate for a short while after switching, while the audio playback continues without interruptions. However, the initial startup delay when the media presentation is started can be lower than that indicated by the empty edits in the Edit List boxes.

As a result, this amendment proposes the following clarifications and improvements:

1. A Track Extension Properties (‘trep’) is proposed to be optionally included in the Movie Extends box. The ‘trep’ box can be used to document or summarize characteristics of the track in the subsequent movie fragments. It may contain any number of child boxes.
2. It is proposed that the Composition to Decode box may also appear in the Track Extension Properties box and then documents all the following movie fragments of the track.
3. Amendment 3 allows Sample Group Description boxes in Track Fragment boxes. Consequently, a client can no longer rely on the sample group description entries in the Movie box to describe the sample grouping of entire presentation including movie fragments. Alternative Startup Sequence Properties (‘assp’) box is therefore proposed to be included in Track Extension Properties box to summarize the characteristics of sample group description entries that may be expected to appear within track fragments for alternative startups sequence sample groups.

1.5 Incomplete Track support

Tracks that are incomplete may result, for example, when subsegments are received partially according to level assignments and `padding_flag` in the Level Assignment Box indicates that the data in a Media Data box that is not received can be replaced by zeros. Consequently, sample data for non-accessed levels is not present but has been replaced by zero bytes, and care should be taken not to attempt to process such samples. While incomplete tracks remain correct syntactically, it would be good to let players know upfront in the Movie box which tracks are incomplete in order to assist players in selection of alternative tracks.

It is proposed to add a new section into the ISOBMFF for indication of incomplete tracks. In order to indicate incomplete tracks explicitly, they are proposed to have specific sample entry formats similar to those of protected streams (encv, enca, etc.). Tracks having any of these specific sample entry formats should usually not be played, if another track in the same alternate group is available. With this support, readers can detect incomplete tracks from sample entries and avoid processing incomplete tracks. The sample data of incomplete tracks may, however, be included into samples of other tracks by reference, and hence an incomplete track should not be removed as long as any track reference points to the incomplete track.

More information on receiving subsegments partially according to level assignments and storing the partially received subsegments into a conforming ISOBMFF file can be obtained from Clause 2.2 of the Working Draft of DASH Implementation Guidelines (MPEG N11752).

1.6 M2TS compression as file format hint track

MPEG-2 TS Reception Hint Track provides the possibility of storing MPEG-2 TS encapsulated in ISO BMFF for a variety of purposes such as storage and playback in DVR-type applications.

The traditional MPEG-2 TS streams, e.g. intended for broadcast purposes are CBR coded, resulting in significant stuffing at codec layers and/or encapsulation layer. This stuffing is redundant as long as the file is stored locally. Hence a mechanism to reversibly remove this redundant data is very desirable.

2 File Track

2.1 File Track

A new track type, called file track, is specified in the following. The samples of the file track conform to ISO Base Media File Format themselves. In other words, if a sample of a file track is stored as an independent file, a conventional file player conforming to one of the brands indicated in the ftyp box of the sample is able to play such independent file.

A new track type requires definition of handler_type value ‘file’ for the handler (‘hdlr’) box, which indicates that the respective track contains samples formatted according to and complying with the ISO Base Media File Format. A media information header box for file tracks (‘fmhd’) may be specified or a null media header (‘nmhd’) may be used instead.

A file sample entry is specified for the ‘file’ handler type as follows:

```
class FileSampleEntry(codingname) extends SampleEntry (codingname){
   Box[] any_box;
}
```

2.2 Storing ISO base media file format files in file tracks

2.2.1 Introduction

There are various ways to store ISO base media file format files as samples in another file. The first Subclause, below, has each sample in a file track being a ‘whole file’ (whole media presentation). This is certainly simple, but it has disadvantages. The most notable are:

- the samples are ‘large’ and somewhat, as it were, indigestible; for example, some MP4 players expect to be able to load and buffer at least a sample at a time;
- if edits are applied, no matter how small the time-range selected, a systems tool cannot do more than copy the entire sample and mark the desired time range in an edit list.

2.2.2 Entire ISO base media file format files as samples

A file sample entry for files that conform to this part of ISO/IEC 14496 are identified by codingname ‘bmff’. The timescale of the file track should match the movie timescale of the included file, and the respective sample entry is specified as follows:

```
class ISOBaseMediaFileSampleEntry(‘bmff’) extends FileSampleEntry (codingname){
   FileTypeBox file_type;
   Box[] any_box;
}
```

file_type shall be a copy of the File Type box from the reference file.

Since every sample of a file track is a separately playable file, every sample is a sync sample and the sync sample table should be absent from the enclosing file track.

An ISO base media compatible file containing a file track is referred to as level-2 container file, whereas a file not containing any file tracks is referred to as level-1 container file.

2.2.3 Samples of ISO base media file format files as samples, inline initialization

An alternative design would leverage the work in DASH and permit a segment to be a sample. Note that a file divided into segments is the entire file when the sample count is 1. In this design, a sync sample would be a self-initializing media segment, using DASH terminology (or a media segment prepended with the initialization segment).

This also works around one of the issues in the current design – the interpretation of absolute file offsets in the embedded file. If the samples are all relatively-addressed segments (for MP4 family files), the issue of absolute offsets does not arise.

The file format does not allow zero-duration samples (as it makes time-to-sample mapping ambiguous) so pure initialization segments, which would have zero duration, would not be allowed.

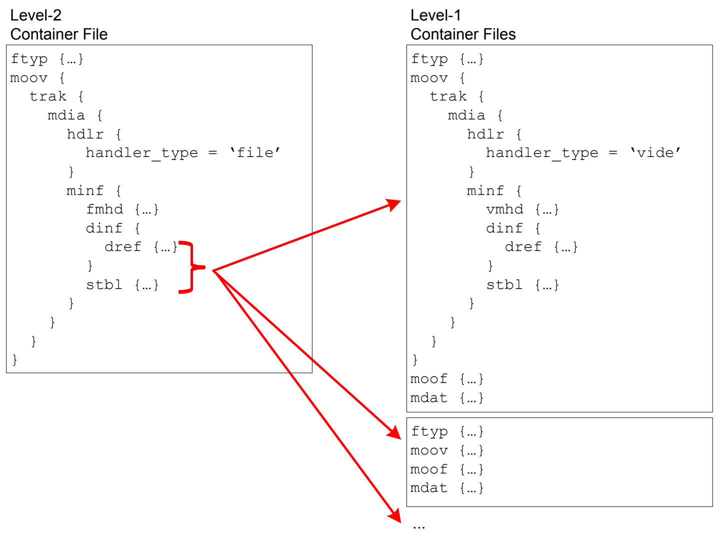
2.2.4 Samples of ISO base media file format files as samples, sample-entry initialization

It is also possible that we could store the entire initialization segment (the ‘empty movie box’, for MP4 family files) in the sample entry. That makes the segments essentially self-contained; we can now use the sync-sample marking to mark segments that ‘contain RAP’ or ‘start with RAP’.

2.3 Level-1 Container Files

2.3.1 Overview

Level-1 container files may be stored as separate files as illustrated in Figure AMD4.1. The level-2 container file then contains a track box for the file track, which in turn includes a data reference box[1] containing one data reference entry per each level-1 container file. Other boxes contained in the track box of the file track are generated as follows: The sample description (‘stsd’) box, which is contained in the sample table (‘stbl’) box, contains sample entries, each of which refers to one data reference entry. The sample to chunk box, which is also contained in the sample table (‘stbl’) box, contains information to associate each sample to a sample description index, hence also indicating the file containing the sample.


Level-2
Container File

```
ftyp {…}
moov {
  trak {
    mdia {
      hdlr {
        handler_type = ‘file’
      }
      minf {
        fmhd {…}
        dinf {
          dref {…}
        }
        stbl {…}
      }
    }
  }
}
```

Level-1
Container Files

```
ftyp {…}
moov {
  trak {
    mdia {
      hdlr {
        handler_type = ‘vide’
      }
      minf {
        vmhd {…}
        dinf {
          dref {…}
        }
        stbl {…}
      }
    }
  }
}
moof {…}
mdat {…}
```

```
ftyp {…}
moov {…}
moof {…}
mdat {…}
```

…

Figure AMD4.1 — Level-2 container file including a file track referring to self-standing level-1 container files.

2.3.2 Example of Data Referencing

In this and the following paragraphs, we give a more detailed example of the data referencing illustrated in general level in Figure AMD4.1. In the example, there are five level-1 container files stored with filename

[1] The data reference (‘dref’) box contains a table of data reference entries that declare the location(s) of the media data used within the associated track. A data reference entry may indicate that the location of the media data is the same file that contains the data reference box itself. Alternatively, a data reference entry may be a URN or a URL. The data reference index in the sample description ties data reference entries to the samples in the track. The media data of a track may be split over several sources in this way.

“sample_n.mp4” where n is an integer from 1 to 5, inclusive. The level-2 container file and all the level-1 container files are stored in the same directory. The data reference box for the file track in the level-2 container file contains the following information (in pseudo-code):

```
FullBox(‘dref’, version = 0, flags = 0) {
  entry_count = 5;
  FullBox(‘url ’, version = 0, flags = 0) {
    location = “file://sample1.mp4”;
  }
  FullBox(‘url ’, version = 0, flags = 0) {
    location = “file://sample2.mp4”;
  }
  FullBox(‘url ’, version = 0, flags = 0) {
    location = “file://sample3.mp4”;
  }
  FullBox(‘url ’, version = 0, flags = 0) {
    location = “file://sample4.mp4”;
  }
  FullBox(‘url ’, version = 0, flags = 0) {
    location = “file://sample5.mp4”;
  }
}
```

Continuing the same example, the sample description (‘stsd’) box within the sample table box for the file track is composed as follows:

```
FullBox(‘stsd’, version = 0, flags = 0) {
  entry_count = 5;
  FileSampleEntry {
    const unsigned int(8)[6] reserved = 0;
    data_reference_index = 1;
    // potentially other syntax elements for sample entry of a file track
  }
  FileSampleEntry {
    const unsigned int(8)[6] reserved = 0;
    data_reference_index = 2;
    // potentially other syntax elements for sample entry of a file track
  }
  FileSampleEntry {
    const unsigned int(8)[6] reserved = 0;
    data_reference_index = 3;
    // potentially other syntax elements for sample entry of a file track
  }
  FileSampleEntry {
    const unsigned int(8)[6] reserved = 0;
    data_reference_index = 4;
    // potentially other syntax elements for sample entry of a file track
  }
  FileSampleEntry {
    const unsigned int(8)[6] reserved = 0;
    data_reference_index = 5;
    // potentially other syntax elements for sample entry of a file track
  }
}
```

Continuing the same example, the file track of the level-2 container file contains five samples, referring to level-1 container files “sample_n.mp4” in increasing order of n = 1 .. 5. Each sample in the file track is dedicated a chunk, because the sample description index changes per each sample. The sample to chunk (‘stsc’) box in the sample table (‘stbl’) box comprises the following information:

```
FullBox('stsc', version = 0, flags = 0) {
  entry_count = 5; // number of chunks
  {
    first_chunk = 1;
    samples_per_chunk = 1;
    sample_description_index = 1;
  }
  {
    first_chunk = 2;
    samples_per_chunk = 1;
    sample_description_index = 2;
  }
  {
    first_chunk = 3;
    samples_per_chunk = 1;
    sample_description_index = 3;
  }
  {
    first_chunk = 4;
    samples_per_chunk = 1;
    sample_description_index = 4;
  }
  {
    first_chunk = 5;
    samples_per_chunk = 1;
    sample_description_index = 5;
  }
}
```

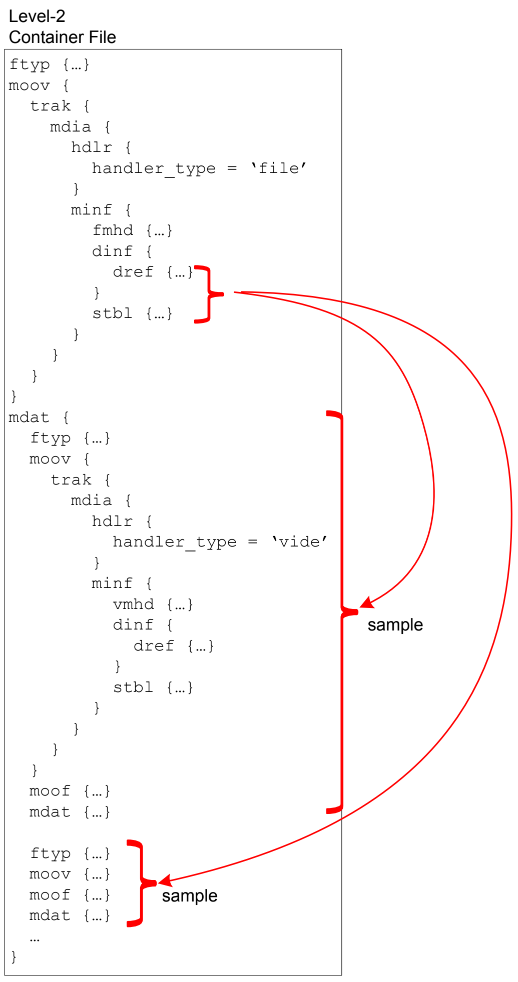
Continuing the same example, the chunk offset box within the sample table box of the file track contains the file offsets for all chunks. In this example, the offsets are 0 (point to the beginning of the files “sample_n.mp4”). Furthermore, the sample size box within the sample table box of the file track contains the sizes of samples in terms of bytes. In this example, the sample size box contains the file sizes of files “sample_n.mp4” in one-byte accuracy.

2.4 Embedded Level-1 Container Files

2.4.1 Overview

The content of level-1 container files may be stored in the mdat box of the level-2 container file as illustrated in Figure AMD4.2. The level-2 container file then contains a track box for the file track, which in turn includes a data reference box containing one data reference entry having the “self-contained flag” set. Other boxes contained in the track box of the file track are generated as follows: The sample description (‘stsd’) box, which is contained in the sample table (‘stbl’) box, contains a sample entry, which refers to the data reference entry. The sample to chunk box, which is also contained in the sample table (‘stbl’) box, contains information to associate each sample to a sample description index, hence also indicating that the level-2 container file contains the samples of the file track.

When the sample data of a file track is stored referring to a data reference entry with “self-contained flag” set, that sample data should be interpreted as if it were a standalone file (e.g. chunk offsets within this included file are within the included file, not the including file).



```
ftyp {…}
moov {
  trak {
    mdia {
      hdlr {
        handler_type = 'file'
      }
      minf {
        fmhd {…}
        dinf {
          dref {…}
        }
        stbl {…}
      }
    }
  }
}
mdat {
  ftyp {…}
  moov {
    trak {
      mdia {
        hdlr {
          handler_type = 'vide'
        }
        minf {
          vmhd {…}
          dinf {
            dref {…}
          }
          stbl {…}
        }
      }
    }
  }
  moof {…}
  mdat {…}

  ftyp {…}
  moov {…}
  moof {…}
  mdat {…}
  …
}
```

Figure AMD4.2 — Level-2 container file including a file track referring to the content of level-1 container files embedded in the mdat box

2.4.2 Example of Data Referencing

In this and the following paragraphs, we give a more detailed example of the data referencing illustrated in general level in Figure AMD4.2. In the example, there are five level-1 container files are embedded in the mdat box of the level-2 container file. The data reference box for the file track in the level-2 container file contains the following information (in pseudo-code):

```
FullBox('dref', version = 0, flags = 0) {
  entry_count = 1;
  FullBox('url ', version = 0, flags = 1) { // self-contained flag is set
  }
}
```

Continuing the same example, the sample description (‘stsd’) box within the sample table box for the file track is composed as follows:

```
FullBox(‘stsd’, version = 0, flags = 0) {
  entry_count = 1;
  FileSampleEntry {
    const unsigned int(8)[6] reserved = 0;
    data_reference_index = 1;
    // potentially other syntax elements for sample entry of a file track
  }
}
```

Continuing the same example, the file track of the level-2 container file contains five samples, which (in this example) are stored contiguously in the mdat box of the level-2 container file. As the samples are contiguous and share the same sample description index, they are included in the same chunk. The sample to chunk (‘stsc’) box in the sample table (‘stbl’) box comprises the following information:

```
FullBox(‘stsc’, version = 0, flags = 0) {
  entry_count = 1; // number of chunks
  {
    first_chunk = 1;
    samples_per_chunk = 5;
    sample_description_index = 1;
  }
}
```

Continuing the same example, the chunk offset box within the sample table box of the file track contains the file offset for the chunk specified in the sample to chunk box. In this example, the offset is set to the byte offset (from the beginning of the level-2 container file) of the first byte of the first sample. Furthermore, the sample size box within the sample table box of the file track contains the sizes of samples in terms of bytes. In this example, the sample size box contains the file sizes of embedded level-1 container files.

2.5 Recording of Received Streams in DASH Client

2.5.1 Description

At any time, A DASH client receives either one representation from group 0, if present, or the combination of at most one representation from each non-zero group. In addition, a DASH client may acknowledge the provided subset information and select the non-zero groups being processed according to a provided subset.

When recording received segments, the DASH client may handle the first received period as follows. The DASH client initializes (i.e., opens a file handle for creating) a level-2 container file. The DASH client determines the groups being received (based on the information in the MPD) and creates one file track per each received group.

A sample for a file track is created from received segments as follows. A sample of a file track contains one initialization segment (i.e., one movie box). Consecutive media segments sharing the same initialization segment are enclosed in the same sample of a file track. In addition to combining an initialization segment and consecutive media segments to a sample, the DASH client may have to create an edit list box as described in the next paragraph.

A sample of a file track is decoded and played identically to playing the file corresponding to the sample. This includes processing of the edit list boxes, if any, and hence recovering the presentation timeline for the sample of a file track. For continuous playout of a file track, each of sample of it should therefore have a presentation timeline, where media playout starts at time zero. The DASH client may generate an entry or entries to an edit list box to force media playout starting at time zero regardless of the composition times indicated in the media segments. Furthermore, when representation switching appears and a switch-to representation is received starting from a segment or movie fragment that does not start from a random access point, the DASH client may generate an entry or entries to an edit list box to indicate that the media playout of the respective sample in a file track starts from the random access point, which is mapped to the time zero of the presentation timeline.

When the reception of a subsequent period is started, the DASH client should continue to record into the same file tracks as used in the previous period. The DASH client should associate the groups of the new period with those of the previous period, e.g., the primary video group of the new period with that of the previous period, and continue to use the file tracks consistently according to these associations. If the number of received groups increases when the reception of a period is stated or such a group is introduced that does not have an associated group earlier, the DASH client should create a new file track. An edit list box should be created for the new file track to map it correctly to the presentation timeline of the level-2 container file.

It is noted that a level-2 container file may contain movie fragments. File recording with movie fragments enables simultaneous recording and playback of the same file and may also be helpful to cope with the risk of unexpected disruptions of the DASH client operation e.g. due to power outage.

2.5.2 Example

In this Subclause, we present an example of a DASH session, the operation of a DASH client, and the file(s) created by the DASH client. The example is illustrated in Figure AMD4.3.

The MPD for the DASH session contains two groups consistently over the periods, one “primary video” group and one “primary audio” group. The primary video group contains two representations coded for different bitrates, for instance. The primary audio group contains only one representation for the purpose of keeping the example simple. The MPD may contain multiple periods, out of which two are illustrated in Figure AMD4.3. Media segments have a constant duration and random access points are not aligned with the start of media segments. In order to keep the example simple, it is assumed that a media segment contains only one movie fragment (or that the DASH client is not able to perform switching at the precision of movie fragments). Each representation has its own initialization segment.

In the example, the DASH client chooses to receive representation 1 of the primary video group at the beginning (instead of representation 2). During the reception of segment m of period 1, the DASH client makes a conclusion that the received video bitrate has to be changed. It therefore requests the initialization segment of representation 2 and media segments of representation 2 starting from media segment m. The decoding and playback of media segment m of representation 2 can be started from the random access point indicated by a dashed vertical line in Figure AMD4.3. Simultaneously to or interleaved with the reception of primary video media segments, the DASH client also receives primary audio segments. When period 2 starts, the DASH client decides again to switch to representation 1 of the primary video group.

The DASH client creates a level-2 container file for the received segments as follows. A file track is created for each group being received – hence, two file tracks are created, one for the primary video group and another one for the primary audio group. The initialization segment of a representation and consecutive received media segments for the representation are concatenated to form a sample of the respective file track. As there is overlapping data in the received media segments m for representations 1 and 2, the DASH client creates an edit list box into the samples of the file track. The edit list box of the first sample of the primary video file track limits the duration of the sample such that it lasts until the mentioned random access point. The edit list box of the second sample of the primary video file track shifts the media timeline onto the presentation timeline of the sample so that the random access point appears at time zero on the presentation timeline. The sample durations of the file track (as coded in the decoding time to sample box and the sample duration fields of track fragments, if any) are set to match the duration of the sample (as governed by its edit list).

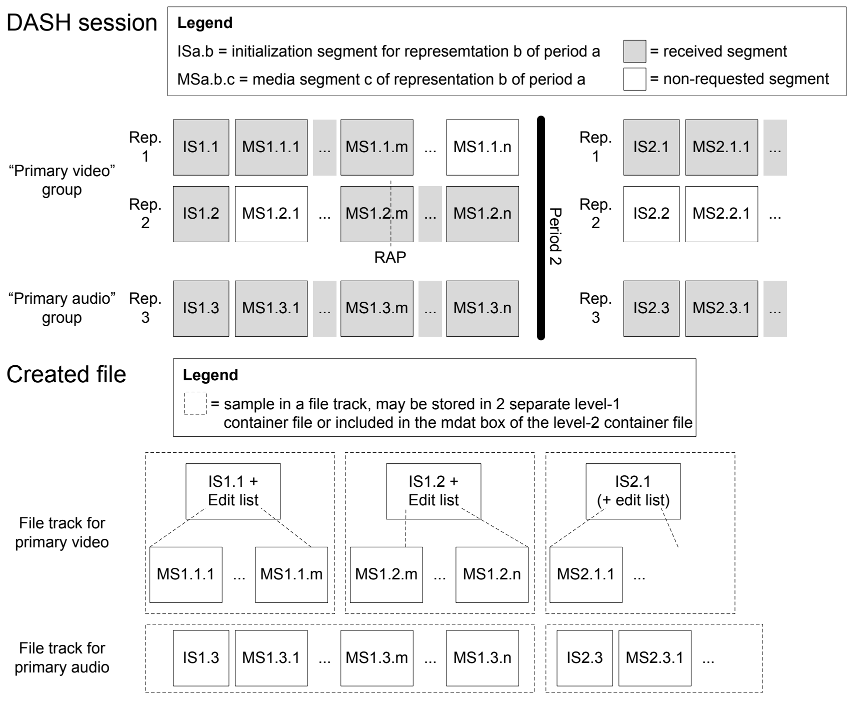


Figure AMD4.3 — Example of DASH session and the recorded file(s)

3 Large Item counts

In 8.11.3.1 Definition, replace:

The box starts with three or four values, specifying the size in bytes of the `offset` field, `length` field, `base_offset` field, and, in version 1 of this box, the `extent_index` fields, respectively. These values must be from the set {0, 4, 8}.

with:

The box starts with three or four values, specifying the size in bytes of the `offset` field, `length` field, `base_offset` field, and, in versions 1 and 2 of this box, the `extent_index` fields, respectively. These values must be from the set {0, 4, 8}.

Replace:

For maximum compatibility, version 0 of this box should be used in preference to version 1 with construction_method==0, when possible.

with:

For maximum compatibility, version 0 of this box should be used in preference to version 1 with construction_method==0, or version 2 when possible. Version 2 of this box should be used to support large item_ID values exceeding 65536.

In 8.11.3.2 Syntax, replace a definition of "aligned(8) class ItemLocationBox" with:

```
aligned(8) class ItemLocationBox extends FullBox('iloc', version, 0) {
   unsigned int(4)   offset_size;
   unsigned int(4)   length_size;
   unsigned int(4)   base_offset_size;
   if ((version == 1) || (version == 2)) {
      unsigned int(4)   index_size;
   } else {
      unsigned int(4)   reserved;
   }
   if (version == 2) {
      unsigned int(32)  item_count;
   } else {
      unsigned int(16)  item_count;
   }
   for (i=0; i<item_count; i++) {
      if (version == 2) {
         unsigned int(32)  item_ID;
      } else {
         unsigned int(16)  item_ID;
      }
      if ((version == 1) || (version == 2)) {
         unsigned int(12)  reserved = 0;
         unsigned int(4)   construction_method;
      }
      unsigned int(16)  data_reference_index;
      unsigned int(base_offset_size*8)  base_offset;
      unsigned int(16)     extent_count;
      for (j=0; j<extent_count; j++) {
         if (((version == 1) || (version == 2)) && (index_size > 0)) {
            unsigned int(index_size*8)  extent_index;
         }
         unsigned int(offset_size*8) extent_offset;
         unsigned int(length_size*8) extent_length;
      }
   }
}
```

In 8.11.4.2, replace a definition of "aligned(8) class PrimaryItemBox" with:

```
aligned(8) class PrimaryItemBox
      extends FullBox('pitm', version, 0) {
   if (version == 1) {
      unsigned int(32)  item_ID;
   } else {
      unsigned int(16)  item_ID;
   }
}
```

In 8.11.4.3, replace:

`item_ID` is the identifier of the primary item

with:

`item_ID` is the identifier of the primary item. Version 1 should be used for large item_ID values exceeding 65536.

In 8.11.6.1, replace:

Three versions of the item info entry are defined. Version 1 includes additional information to version 0 as specified by an extension type. For instance, it shall be used with extension type `'fdel'` for items that are referenced by the file partition box (`'fpar'`), which is defined for source file partitionings and applies to file delivery transmissions. Version 2 provides an alternative structure in which metadata item types are indicated by a 32-bit (typically 4-character) registered or defined code; two of these codes are defined to indicate a MIME type or metadata typed by a URI.

If no extension is desired, the box may terminate without the `extension_type` field and the extension; if, in addition, `content_encoding` is not desired, that field also may be absent and the box terminate before it. If an extension is desired without an explicit `content_encoding`, a single null byte, signifying the empty string, must be supplied for the `content_encoding`, before the indication of `extension_type`.

If file delivery item information is needed and a version 2 ItemInfoEntry is used, then the file delivery information is stored (a) as a separate item of type 'fdel') (b) linked by an item reference from the item, to the file delivery information, of type 'fdel'. There must be exactly one such reference if file delivery information is needed.

It is possible that there are valid URI forms for MPEG-7 metadata (e.g. a schema URI with a fragment identifying a particular element), and it may be possible that these structures could be used for MPEG-7. However, there is explicit support for MPEG-7 in ISO base media file format family files, and this explicit support is preferred as it allows, among other things:

a) incremental update of the metadata (logically, I/P coding, in video terms) whereas this draft is 'I-frame only';

b) binarization and thus compaction;

c) the use of multiple schemas.

Therefore, the use of these structures for MPEG-7 is deprecated (and undocumented).

Information on URI forms for some metadata systems can be found in Annex G.

with:

Four versions of the item info entry are defined. Version 1 includes additional information to version 0 as specified by an extension type. For instance, it shall be used with extension type `'fdel'` for items that are referenced by the file partition box (`'fpar'`), which is defined for source file partitionings and applies to file delivery transmissions. Version 2 provides an alternative structure in which metadata item types are indicated by a 32-bit (typically 4-character) registered or defined code; two of these codes are defined to indicate a MIME type or metadata typed by a URI. Version 3 is defined to support large item_ID values.

If no extension is desired, the box may terminate without the `extension_type` field and the extension; if, in addition, `content_encoding` is not desired, that field also may be absent and the box terminate before it. If an extension is desired without an explicit `content_encoding`, a single null byte, signifying the empty string, must be supplied for the `content_encoding`, before the indication of `extension_type`.

If file delivery item information is needed and a version 2 ItemInfoEntry is used, then the file delivery information is stored (a) as a separate item of type 'fdel') (b) linked by an item reference from the item, to the file delivery information, of type ‘fdel’. There must be exactly one such reference if file delivery information is needed.

It is possible that there are valid URI forms for MPEG-7 metadata (e.g. a schema URI with a fragment identifying a particular element), and it may be possible that these structures could be used for MPEG-7. However, there is explicit support for MPEG-7 in ISO base media file format family files, and this explicit support is preferred as it allows, among other things:

a) incremental update of the metadata (logically, I/P coding, in video terms) whereas this draft is ‘I-frame only’;

b) binarization and thus compaction;

c) the use of multiple schemas.

Therefore, the use of these structures for MPEG-7 is deprecated (and undocumented).

Information on URI forms for some metadata systems can be found in Annex G.

Version 1 of ItemInfoBox should be used for larger number of itemInfoEntries exceeding 65535.

In 8.11.6.2, replace definitions of "aligned(8) class ItemInfoEntry" and "aligned(8) class ItemInfoBox" with:

```
aligned(8) class ItemInfoEntry
      extends FullBox(‘infe’, version, 0) {
   if ((version == 0) || (version == 1)) {
      unsigned int(16)  item_ID;
      unsigned int(16)  item_protection_index
      string            item_name;
      string            content_type;
      string            content_encoding;   //optional
   }
   if (version == 1) {
      unsigned int(32)  extension_type;     //optional
      ItemInfoExtension(extension_type);    //optional
   }
   if ((version == 2) || (version == 3)) {
      if (version == 3) {
         unsigned int(32)  item_ID;
      } else {
         unsigned int(16)  item_ID;
      }
      unsigned int(16)  item_protection_index;
      unsigned int(32)  item_type;

      string            item_name;
      if (item_type==’mime’) {
         string         content_type;
         string         content_encoding;   //optional
      } else if (item_type == ‘uri ‘) {
         string         item_uri_type;
      }
   }
}
aligned(8) class ItemInfoBox
      extends FullBox(‘iinf’, version, 0) {
   if (version == 1) {
      unsigned int(32)  entry_count;
   } else {
      unsigned int(16) entry_count;
   }
   ItemInfoEntry[ entry_count ]      item_infos;
}
```

In 8.11.12.1 Definition, replace:

The item reference box allows the linking of one item to others via typed references. All the references for one item of a specific type are collected into a single item type reference box, whose type is the reference type, and which has a ‘from item ID’ field indicating which item is linked. The items linked to are then represented by an array of ‘to item ID’s. All these single item type reference boxes are then collected into the item reference box. The reference types defined for the track reference box defined in 8.3.3 may be used here if appropriate, or other registered reference types.

with:

The item reference box allows the linking of one item to others via typed references. All the references for one item of a specific type are collected into a single item type reference box, whose type is the reference type, and which has a ‘from item ID’ field indicating which item is linked. The items linked to are then represented by an array of ‘to item ID’s. All these single item type reference boxes are then collected into the item reference box. The reference types defined for the track reference box defined in 8.3.3 may be used here if appropriate, or other registered reference types. Version 1 of ItemReferenceBox with SingleItemReferenceBoxLarge should be used for large from_item_ID and to_item_ID values exceeding 65535.

Replace 8.11.12.2 Syntax with:

```
aligned(8) class SingleItemTypeReferenceBox(referenceType) extends
Box(referenceType) {
   unsigned int(16) from_item_ID;
   unsigned int(16) reference_count;
   for (j=0; j<reference_count; j++) {
      unsigned int(16) to_item_ID;
   }
}
aligned(8) class SingleItemTypeReferenceBoxLarge(referenceType) extends
Box(referenceType) {
   unsigned int(32) from_item_ID;
   unsigned int(16) reference_count;
   for (j=0; j<reference_count; j++) {
      unsigned int(32) to_item_ID;
   }
}

aligned(8) class ItemReferenceBox extends FullBox(‘iref’, version, 0) {
   if (version==0) {
      SingleItemTypeReferenceBox    references[];
   } else if (version==1) {
      SingleItemTypeReferenceBoxLarge   references[];
   }
}
```

In 8.13.3.1, replace:

The File Partition box identifies the source file and provides a partitioning of that file into source blocks and symbols. Further information about the source file, e.g., filename, content location and group IDs, is contained in the Item Information box (`'iinf'`), where the Item Information entry corresponding to the item ID of the source file is of version 1 and includes a File Delivery Item Information Extension (`'fdel'`).

with:

The File Partition box identifies the source file and provides a partitioning of that file into source blocks and symbols. Further information about the source file, e.g., filename, content location and group IDs, is contained in the Item Information box (`'iinf'`), where the Item Information entry corresponding to the item ID of the source file is of version 1 and includes a File Delivery Item Information Extension (`'fdel'`). Version 1 of FilePartitionBox should be used to support large item_ID values and entry_count exceeding 65535.

In 8.13.3.2, replace a definition of "aligned(8) class FilePartitionBox" with:

```
aligned(8) class FilePartitionBox
      extends FullBox('fpar', version, 0) {
   if (version == 1) {
      unsigned int(32)  item_ID;
   } else {
      unsigned int(16)  item_ID;
   }
   unsigned int(16)  packet_payload_size;
   unsigned int(8)   reserved = 0;
   unsigned int(8)   FEC_encoding_ID;
   unsigned int(16)  FEC_instance_ID;
   unsigned int(16)  max_source_block_length;
   unsigned int(16)  encoding_symbol_length;
   unsigned int(16)  max_number_of_encoding_symbols;
   string            scheme_specific_info;
   if (version == 1) {
      unsigned int(32)  entry_count;
   } else {
      unsigned int(16)  entry_count;
   }
   for (i=1; i <= entry_count; i++) {
      unsigned int(16)  block_count;
      unsigned int(32)  block_size;
   }
}
```

In 8.13.3.1, replace:

The FEC reservoir box associates the source file identified in the file partition box (`'fpar'`) with FEC reservoirs stored as additional items. It contains a list that starts with the first FEC reservoir associated with the first source block of the source file and continues sequentially through the source blocks of the source file.

with:

The FEC reservoir box associates the source file identified in the file partition box (`'fpar'`) with FEC reservoirs stored as additional items. It contains a list that starts with the first FEC reservoir associated with the first source block of the source file and continues sequentially through the source blocks of the source file. Version 1 of FECReservoirBox should be used to support large item_ID values and entry_count exceeding 65535.

In 8.13.4.2, replace a definition of "aligned(8) class FECReservoirBox" with:

```
aligned(8) class FECReservoirBox
      extends FullBox('fecr', version, 0) {
   if (version == 1) {
      unsigned int(32)  entry_count;
   } else {
      unsigned int(16)  entry_count;
   }
   for (i=1; i <= entry_count; i++) {
      if (version == 1) {
         unsigned int(32)  item_ID;
      } else {
         unsigned int(16)  item_ID;
      }
      unsigned int(32)  symbol_count;
   }
}
```

In 8.13.7.1, replace:

The File reservoir box associates the source file identified in the file partition box ('fpar') with File reservoirs stored as additional items. It contains a list that starts with the first File reservoir associated with the first source block of the source file and continues sequentially through the source blocks of the source file.

with:

The File reservoir box associates the source file identified in the file partition box ('fpar') with File reservoirs stored as additional items. It contains a list that starts with the first File reservoir associated with the first source block of the source file and continues sequentially through the source blocks of the source file. Version 1 of FileReservourBox should be used to support large item_ID values and entry_count exceeding 65535.

In 8.13.7.2, replace a definition of "aligned(8) class FileReservoirBox" with:

```
aligned(8) class FileReservoirBox
      extends FullBox('fire', version, 0) {
   if (version == 1) {
      unsigned int(32)  entry_count;
   } else {
      unsigned int(16)  entry_count;
   }
   for (i=1; i <= entry_count; i++) {
      if (version == 1) {
         unsigned int(32)  item_ID;
      } else {
         unsigned int(16)  item_ID;
      }
      unsigned int(32)  symbol_count;
   }
}
```

In 9.2.4.5, between definitions of "aligned(8) class FDitemconstructor" and "aligned(8) class FDxmlboxconstructor", add:

```
aligned(8) class FDitemconstructorLarge extends FDconstructor(3)
{
   unsigned int(32)  item_ID;
   unsigned int(32)  extent_index;
   unsigned int(64)  data_offset;    //offset in byte within extent
   unsigned int(24)  data_length;    //non-zero length in byte within extent or
                                     //if (data_length==0) rest of extent
}
```

Noboru Harada 9/14/2011 9:56 AM
Comment [1]: Is this OK? Or should we use different value for the construction?

harada 7/23/2011 12:59 AM
Comment [2]: Check if this update from 16bits to 32bits is needed

4 Movie Fragment Related

*In 8.6.1.4.1, add Track Extension Properties Box (`'trep'`) as a container box as follows:*Box Type: `'cslg'`
Container: Sample Table Box (`'stbl'`) or Track Extension Properties Box (`'trep'`)
Mandatory: No
Quantity: Zero or one

Add the following sentence after the sentence starting with "When the Composition to Decode Box is included in the Sample Table Box":

When the Composition to Decode Box is included in the Track Extension Properties Box, it documents the composition and decoding time relations of the samples in all movie fragments following the Movie Box.

In 8.8, add the following Subclauses:

8.8.12 Track Extension Properties Box

8.8.12.1 Definition

Box Type: `'trep'`
Container: Movie Extends Box (`'mvex'`)
Mandatory: No
Quantity: Zero or more. (Zero or one per track)

This box can be used to document or summarize characteristics of the track in the subsequent movie fragments. It may contain any number of child boxes.

8.8.12.2 Syntax

```
class TrackExtensionPropertiesBox extends FullBox('trep', 0, 0) {
   unsigned int(32) track_id;
   // Any number of boxes may follow
}
```

8.8.12.3 Semantics

`track_id` indicates the track for which the track extension properties are provided in this box.

Add the following Subclauses:

8.8.13 Alternative Startup Sequence Properties Box

8.8.13.1 Definition

Box Type: `'assp'`
Container: Track Extension Properties Box (`'trep'`)
Mandatory: No
Quantity: Zero or one

This box indicates the properties of alternative startup sequence sample groups in the subsequent track fragments of the track indicated in the containing Track Extension Properties box.

Version 0 of the Alternative Startup Sequence Properties box shall be used if version 0 of the Sample to Group box is used for the alternative startup sequence sample grouping. Version 1 of the Alternative Startup Sequence Properties box shall be used if version 1 of the Sample to Group box is used for the alternative startup sequence sample grouping.

8.8.13.2 Syntax

```
class AlternativeStartupSequencePropertiesBox extends FullBox('assp', version, 0)
{
   if (version == 0) {
      signed int(32)    min_initial_alt_startup_offset;
   }
   else if (version == 1) {
      unsigned int(32)  num_entries;
      for (j=1; j <= num_entries; j++) {
         unsigned int(32)  grouping_type_parameter;
         signed int(32)    min_initial_alt_startup_offset;
      }
   }
}
```

8.8.13.3 Semantics

`min_initial_alt_startup_offset`: No value of sample_offset[1] of the referred sample group description entries of the alternative startup sequence sample grouping shall be smaller than min_initial_alt_startup_offset. In version 0 of this box, the alternative startup sequence sample grouping using version 0 of the Sample to Group box is referred to. In version 1 of this box, the alternative startup sequence sample grouping using version 1 of the Sample to Group box is referred to as further constrained by `grouping_type_parameter`.

`num_entries` indicates the number of alternative startup sequence sample groupings documented in this box.

`grouping_type_parameter` indicates which one of the alternative sample groupings this loop entry applies to.

5 Incomplete Track support

Replace 8.12.2 introduction and syntax as follows:

Box Types: `'frma'`
Container: Protection Scheme Information Box (`'sinf'`)
or Complete Track Information Box (`'cinf'`)
Mandatory: Yes when used in a protected sample entry.
Yes when used in a sample entry for an incomplete track.
Quantity: Exactly one in Protection Scheme Information Box.
Exactly one in Complete Track Information Box.

The Original Format Box `'frma'` contains the four-character-code of the original un-transformed sample description:

```
aligned(8) class OriginalFormatBox(codingname) extends Box ('frma') {
   unsigned int(32)  data_format = codingname;
                     // format of the un-transformed sample entry
}
```

In Clause 8, add the following Subclauses:

8.17 Support for Incomplete Tracks

8.17.1 General

This Subclause documents the sample entry formats for tracks that are incomplete. Incomplete tracks may contain samples that are marked empty or not received using the sample format.

Incomplete tracks may result, for example, when subsegments are received partially according to level assignments and `padding_flag` in the Level Assignment box indicates that the data in a Media Data box that is not received can be replaced by zeros. Consequently, sample data assigned to non-accessed levels is not present, and care should be taken not to attempt to process such samples. However, in partially received subsegments some tracks might remain complete in content while other tracks might be incomplete and only contain data that is included by reference into the complete tracks.

This Subclause specifies support for sample entry formats for incomplete tracks. With this support, readers can detect incomplete tracks from their sample entries and avoid processing such tracks or take the possibility of empty or not received samples into account when processing such tracks.

The support for incomplete tracks is similar to the content protection transformation where sample entries are hidden behind generic sample entries, such as `'encv'` and `'enca'`. Because the format of a sample entry varies with media-type, a different encapsulating four-character-code is used for incomplete tracks of each media type (audio, video, text etc.). They are:

Stream (Track) Type	**Sample-Entry Code**
Video	`icpv`
Audio	`icpa`
Text	`icpt`
System	`icps`
Hint	`icph`
Timed Metadata	`icpm`

Sample data of incomplete tracks may be included into samples of other tracks by reference, and hence an incomplete track should not be removed as long as any track reference points to it.

8.17.2 Transformation

The sample entry for a track that becomes incomplete e.g. through partial reception, should be modified as follows:

1) The four-character-code of the sample entry, e.g. `'avc1'`, is replaced by a new sample entry code `'icpv'` meaning an incomplete track.

2) A Complete Track Information box is added to the sample description, leaving all other boxes unmodified.

3) The original sample entry type, e.g. `'avc1'`, is stored within an Original Format box contained in the Complete Track Information box.

After transformation, an example sample entry will look like:

```
class IncompleteAVCSampleEntry() extends VisualSampleEntry ('icpv'){
   CompleteTrackInfoBox();
   AVCConfigurationBox config;
   MPEG4BitRateBox (); // optional
   MPEG4ExtensionDescriptorsBox (); // optional
}
```

8.17.3 Complete Track Information Box

8.17.3.1 Definition

Box Types:	`'cinf'`
Container:	Sample Entry for an Incomplete Track
Mandatory:	Yes
Quantity:	Exactly one

The Complete Track Information Box contains, within the Original Format Box, the sample entry format of the complete track that was transformed to the present incomplete track. It may contain optional boxes for example including information required to process samples of the present incomplete track. The Complete Track Information Box is a container box. It is mandatory in a sample entry that uses a code indicating an incomplete track.

8.17.3.2 Syntax

```
aligned(8) class CompleteTrackInfoBox(fmt) extends Box('cinf') {
   OriginalFormatBox(fmt)  original_format;
}
```

6 M2TS compression as file format hint track

In 9.3.4.1 before the line "// Packet format", add the following:

```
aligned(8) class MPEG2TSReplicatedFFConstructor
   extends MPEF2TSConstructor(3) {
   uint(8) replicatedcount;
}
```

Replace the syntax of `MPEG2TSPacketRepresentation` and `MPEG2TSSample` in 9.3.4.1 with the following:

```
// Packet format
aligned(8) class MPEG2TSPacketRepresentation {
   uint(8)  precedingbytes[precedingbyteslen];
   uint(8)  sync_byte;
   if (sync_byte == 0x47) {
      uint(8)     packet[187];
   } else if (sync_byte == 0x00 || sync_byte == 0x01) {
      uint(8)     headerdatalen;
      uint(4)     reserved;
      uint(4)     num_constructors;
      bit(1)      transport_error_indicator;
      bit(1)      payload_unit_start_indicator;
      bit(1)      transport_priority;
      bit(13)     PID;
      bit(2)      transport_scrambling_control;
      bit(2)      adaptation_field_control;
      bit(4)      continuity_counter;
      if (sync_byte == 0x00 && (adaptation_field_control == ´10´ ||
          adaptation_field_control == ´11´)) {
         uint(8)     adaptation_field[headerdatalen-3];
      }
      MPEG2TSConstructor   constructors[num_constructors];
   } else if (sync_byte == 0xFF) {
      // implicit null packet that has been removed
   }
   uint(8)   trailingbytes[trailingbyteslen];
}
// Sample format
aligned(8) class MPEG2TSSample {
   MPEG2TSPacketRepresentation   sample[];
}
```

in 9.3.4.2, replace the semantics of sync_byte with the following:

`sync_byte`: if this value is 0x47, then the sample is a transport stream packet (a precomputed reception hint track sample), with the remaining bytes following in the field `packet`. The values 0x00 and 0x01 are used for constructed sample(s). If MPEG2TSSampleConstructor is used to construct sample(s), it points to a track indexed by `trackrefindex` in the track reference box with reference type 'hint'. If this value is 0xFF, it implies that a null packet has been removed at this position. All other values are currently reserved. When the packet data is actually put into a streaming channel, the value shall always be set to 0x47.

In 9.3.4.2, replace the semantics of the constructors array with the following:

The `constructors` array is a collection of one or more constructor entries, to allow for multiple access units in one transport stream packet or to combine different types of constructors to construct a transport stream packet. An `MPEG2TSImmediateConstructor` can contain, amongst others, the PES header. An `MPEG2TSSampleConstructor` references data in the associated media track. An `MPEG2TSReplicatedFFConstructor` is used to efficiently represent a sequence of 0xFF byte values. If `MPEG2TSReplicatedFFConstructor` is not used, the sum of `headerdatalen` and the `datalen` fields of all constructors of an `MPEG2TSPacket` must be equal to the length of the transport stream packet being constructed, minus 1 byte, which is 187.

In 9.3.4.2 between the semantics of trailingbytes and samplenumber, add the following semantics of replicatedcount:

`replicatedcount`: the number of times that the 0xFF byte is repeated.