

**ISO/IEC JTC 1/SC 29/WG 11**

**Coding of moving pictures and audio**

**Convenorship: UNI (Italy)**

**ISO/IEC JTC 1/SC 29/WG 11 N19257**

**Document type: Approved WG 11 document**

**Title: Technologies under Consideration for ISOBMFile Format**

**Status: Approved**

**Date of document: 2020-04-28**

**Source: Systems**

**Expected action: -**

**No. of pages: 40**

**Email of convenor: leonardo@chiariglione.org**

**Committee URL: http://mpeg.chiariglione.org**

**INTERNATIONAL ORGANISATION FOR STANDARDISATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 11**

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC 1/SC 29/WG 11 N19257**

**e-meeting – April 2020**

|  |  |
| --- | --- |
| **Source:** | **Systems** |
| **Editors** | David Singer, Miska M. Hannuksela, Frédéric Mazé |
| **Title:** | **Technologies under Consideration for ISOBMFile Format** |

**Abstract**

The document contains following technologies under consideration for the ISO base media file format (ISO/IEC 14496-12):

[1 Compact full box 3](#_Toc38982533)

[2 New set of composition modes for video tracks 3](#_Toc38982534)

[3 'loss', 'null' and 'crpt' sample entries 5](#_Toc38982535)

[4 Edit lists and movie fragments 6](#_Toc38982536)

[5 Multiplexed timed metadata tracks 6](#_Toc38982537)

[6 Embedded Metadata Signaling 12](#_Toc38982538)

[7 On MIME type parameters 13](#_Toc38982539)

[8 On relation of entity groups and sample groups 14](#_Toc38982540)

[9 Improved sub-sample information 14](#_Toc38982541)

[10 Non-independent tracks 17](#_Toc38982542)

[11 On just-in-time delivery of items 17](#_Toc38982543)

[12 Movie Fragment Compaction considerations 18](#_Toc38982544)

[13 Stronger defaulting in Track Runs 21](#_Toc38982545)

[14 Sample reordering in Track Runs 31](#_Toc38982546)

[15 Extending Segment Index 36](#_Toc38982547)

[16 Segment Index and Level Assignment 38](#_Toc38982548)

[17 Haptics support 40](#_Toc38982549)

# Compact full box

### Description

This primarily benefits scenarios where a certain stream contains many track runs, such as:

* Low-latency scenarios, where very short fragments (up to a single frame per fragment) result in very short track runs.
* Scenarios with a relatively high number of I frames. Given that the CompactTrackFragmentRunBox allows signalling a different sample size for the first sample in a ctrn, it can be more efficient to start a new ctrn box for each GOP.

### Syntax

aligned(8) class CompactFullBox(unsigned int(32) boxtype, unsigned int(2) v, bit(6) f) extends Box(boxtype) {  
 unsigned int(2) version = v;  
 bit(6) flags = f;  
}

# New set of composition modes for video tracks

[Ed. (FM): This section results from the following decision regarding m40401 in MPEG#118 file format minutes (The contribution was related to the carriage of alpha compositing metadata):

*“Disposition: noted*

We think there are two design directions here:

* + - 1. for compositing that is not time-dependent, and for which the track matrix can do positioning, then the matrix, track layer, and video media header graphicsmode could supply the structure for this. This contribution seems to propose considering new values for graphicsmode (we currently only have 'copy').
      2. for time-based actions e.g. cross-fades, dissolves, or for more complex transformations (e.g. de-warping) we could use the structures of the TuC”.

“We suggest putting this table as a proposed new set of graphicsmodes in the TuC, along with the explicit layering provided by the layer field and the transforms of the matrix field.”]

As illustrated in annex A.6 of ISO/IEC 14496-12 [1], the composition of video tracks can be performed “by following their layer number (from back to front), and their composition mode. In addition, each track may be transformed by means of a matrix, and also the overall movie transformed by **matrix**. This permits both simple operations (e.g. pixel doubling, correction of 90º rotation) as well as more complex operations (shearing, arbitrary rotation, for example)”.

Currently, only one composition mode ‘copy’ is defined that copies the video track over the existing image. To support more complex operations (e.g. alpha compositing), new composition modes are proposed in table below.

In the table, term ‘Source’ and term ‘Destination’ are used interchangeably for the front/top layer and the back/bottom layer or the backdrop, respectively. The parameter value of ‘layer’ in TrackHeaderBox of each tracks specifies the front-to-back ordering of visual track.

Such table and the associated algorithms with default parameters may be defined in a separate document, e.g. ISO/IEC 23001-8 or W3C recommendation [2].

1. *“ISO/IEC Part 12 – ISO Base Media File Format”,* Fifth Edition*,* February 2015.
2. *“W3C: Composing and Blending 1.0”,* W3C Candidate Recommendation, January 2015.

**New proposed set of composition mode (graphicsmode) values**

|  |  |  |
| --- | --- | --- |
| **Value** | **Compositing mode** | **Description** |
| 1 | Clear | No regions are enabled. |
| 2 | Source | Only the source will be present. |
| 3 | Destination | Only the destination will be present. |
| 4 | Source Over | Source is placed over the destination. |
| 5 | Destination Over | Destination is placed over the source. |
| 6 | Source In | Source that overlaps the destination, replaces the destination. |
| 7 | Destination In | Destination which overlaps the source, replaces the source. |
| 8 | Source Out | Source is placed, where it falls outside of the destination. |
| 9 | Destination Out | Destination is placed, where it falls outside of the source. |
| 10 | Source Atop | Source which overlaps the destination, replaces the destination. Destination is placed elsewhere. |
| 11 | Destination Atop | Destination which overlaps the source replaces the source. Source is placed elsewhere. |
| 12 | XOR | The non-overlapping regions of source and destination are combined. |
| 13 ~16 | Reserved |  |
| 17 | Dissolve | Takes random pixels from both layers. With high opacity, most pixels are taken from the source layer. With low opacity most pixels are taken from the destination layer. |
| 18 | Plus | Display the sum of the source image and destination image. |
| 19 | Multiply | The source color is multiplied by the destination color and replaces the destination. |
| 20 | Screen | The values of the pixels in the two layers are inverted, multiplied, and then inverted again |
| 21 | Overlay | Overlay combines Multiply and Screen blend modes. |
| 22 | Darken | Selects the darker of the destination and source colors. |
| 23 | Lighten | Selects the lighter of the destination and source colors |
| 24 | Color\_dodge | Brightens the destination color to reflect the source color |
| 25 | Color\_burn | Darkens the destination color to reflect the source color |
| 26 | Hard\_light | Multiplies or screens the colors, depending on the source color value. |
| 27 | Soft\_light | Darkens or lightens the colors, depending on the source color value. |
| 28 | Difference | Subtracts the darker of the two constituent colors from the lighter color |
| 29 | Exclusion | Produces an effect similar to that of the Difference mode but lower in contrast. |
| 30 ~ 255 | Reserved | For new modes or non-separable blending modes. |

# 'loss', 'null' and 'crpt' sample entries

[Ed. (MH): This section results from the following decisions in MPEG#116 and MPEG#117: In the MPEG#117 file format minutes regarding M39936: "The ‘null’ and ‘crpt’ sample entries should be in the TuC for part 12." In the MPEG#116 file format minutes regarding M39333: "We add a section for now to the Partial file WD, but that would go into Part12, to document 'null' and 'crpt' samples (lost being a special case)."]

[Ed. (FM): In the MPEG#118 file format minutes regarding m40528: “In this section, we document the three sample entries above; but the receiver might like to know the original format for material that is lost or corrupted. Should these be seen as transformations of the sample entry (like encryption, or restricted) rather than independent? They then would vary an original format box etc.

This, of course, raises the question of whether you always know what the type of the lost samples was.”]

The sample entry type 'loss' indicates that the file writer does not know what media should be here, if anything. The sample entry type 'null' indicates the same as an empty edit. When the sample entry type is 'loss' or 'null', the sample size shall be equal to 0. [Ed. (MH): From M38920 with editorial changes.]

A corrupted media frame is media frame that cannot be correctly decoded, due to being partially received or due to missing data in the media frames that are in the inter frame prediction chain. If a decoder tries to decode a corrupted media frame, an unexpected behavior, which can be as bad as decoder crash, may occur.

For indication of corrupted media files, the sample entry type 'crpt' is defined. This way, samples that contain media frames that are known to be corrupted can be associated with a sample entry of type 'crpt'. [Ed. (MH): 'crpt' originates from M39333 with editorial changes.]

'loss' sample entries should carry an OriginalFormatBox, documenting the format of the lost samples. 'crpt' sample entries must carry both an original format box, and also any sample entry boxes required by the original format.

# Edit lists and movie fragments

aligned(8) class EditAdjustmentBox extends FullBox('efrg', version, 0) {  
 unsigned int(6) reserved = 0;  
 unsigned int(1) initial\_adjust;  
 unsigned int(1) final\_adjust;  
 if initial\_adjust { unsigned int(32) initial\_delta; }  
 if final\_adjust { unsigned int(32) final\_delta; }  
}

EditAdjustmentBox should be placed near (after) the 'tfdt' box. If this box is absent, it is equivalent to it being present with initial\_adjust and final\_adjust set to false (0).

In an initial movie, if either there is no edit list (no 'edts' container box), or the initial track is empty (refers to no samples) and there is an edit of zero duration, it is equivalent to having final\_adjust set to false (0).

Derive the following two values:

1. insert\_start = earlest\_comp\_time + { initial\_adjust ? initial\_delta : 0}
2. insert\_end = latest\_comp\_time - { final\_adjust ? final\_delta : 0}

Then, on receipt of a fragment, if the previous fragment has no final\_adjust and this fragment has no initial\_adjust, extend the existing edit to insert\_end; otherwise, insert a new edit from insert\_start to insert\_end.

[Ed.: There is a small (well-known) problem here: the insert\_end is not quite right automatically, as it should include the composition duration of the latest sample; the snag is, we do not know it. There doesn't seem to be a good solution to this that doesn't also raise problems we are trying to avoid: in essence we need this 'edit adjustment' to talk about a 'little bit' of the future (the composition duration of the last frame) without straying into (e.g.) the next fragment. Fixes might include making the final\_adjust signed, explicitly including the frame duration, changing the editadjustment from having a final\_adjust to having a duration from the insert\_start, but none of these are ideal. Which is probably why this is still a discussion point.]

# Multiplexed timed metadata tracks

## Basic Design

The basic design is in the 7th edition Amendment 1. However, we have possible extensions on the table; inline associations, sample groups, and structurally-dependent metadata.

## Extensions

## Carrying inline associations

### General

In some circumstances, it can be useful to carry inline definitions. This allows for more flexibility, at the expense of risking that a reader will encounter metadata items it does not recognize and did not expect.

The support is fairly simple; a box in the sample entry to warm that inline carriage may occur, and then a value item box in the sample data that carries both a MetedataKeyBox (the declaration) and a value box.

Tracks without inline keys offer a few advantages:

* A client can determine the entire set of keys that may be present in the track allowing the client to ignore the entire track if no keys of interest to the client are present.
* Space is optimized as keys are carried once in the sample entry and values have only a box header to frame their data and associate them with their key. Inline key/value boxes carry a MetadataKeyBox so if multiple inline keys are present in the same access units, they do not share the key with sibling boxes or with other access units.
* If the sample group optimization technique described earlier is used, the client can determine which samples contain values for particular keys.

Tracks signaling the presence of inline keys offer a few advantages:

* If a new key is required (e.g., for new key, data type, locale, etc. combination), an inline key/value box can be written at any time. This contrasts to the non-inline key case where the set of keys must be known a priori.
* A client does not need to be able to enumerate all combinations of key properties (key, datatype, locale, etc.) to write the track.

With that said, it is recommended that tracks be created without using inline keys. Inline keys can be seen as an optional fallback facility to be used when needed.

A movie may contain a combination of metadata tracks, some using inline keys and some not using inline keys. Where the set of keys can be known a priori, non-inline key- bearing tracks can be used. Where that’s not possible, inline keys can be used.

### Sample entry

The optional MetadataInlineKeysPresentBox indicates if inline key/value boxes might occur in corresponding access units. If MetadataInlineKeysPresentBox is absent or present with a presence indicator of 0, no inline key/value boxes (value boxes with a local\_id of 0xFFFFFFFF) should occur in the access units.

The box is defined as:

aligned(8) class MetadataInlineKeysPresentBox extends Box('keyi') {   
 unsigned int(8) inlineKeyValueBoxesPresent;  
}

inlineKeyValueBoxesPresent is a Boolean integer that should be set to a non-zero value (the value 1 is preferred) if inline key/value boxes are known to be present or might be present in the access units associated with this sample entry.

If MetadataInlineKeysPresentBox is present but inlineKeyValueBoxesPresent is set to 0, access units should be treated as though no MetadataInlineKeysPresentBox is attached to the sample entry. Whether MetadataInlineKeysPresentBox is absent or inlineKeyValueBoxesPresent is set to 0, access unit metadata values having a local id of 0xFFFFFFFF should be ignored.

This approach allows a sample entry to reserve space for and include a MetadataInlineKeysPresentBox but to rewrite just the inlineKeyValueBoxesPresent field to 0 to signal there are no inline key/value boxes present.

If all sample values include inline keys, a MetadataKeyTableBox shall still be present although it may be empty (i.e., it contains no MetadataKeyBoxes). It is also possible to have a combination of some known keys signaled in the MetadataKeyTableBox and some inline key/values signaled with a MetadataInlineKeysPresentBox.

### Sample data item

If the access units associated with the BoxedMetadataSampleEntry contain inline key/value metadata, each inline item is carried in a box with a local\_key\_id of 0xFFFFFFFF and conforming to the type MetadataInlineKeyValueAUBox defined as:

aligned(8) class MetadataInlineKeyValueAUBox extends MetadataAUBox(0xFFFFFFFF) {  
 MetadataKeyBox inline\_key; // local\_key\_id of ‘1key’   
 MetadataAUBox inline\_value; // local\_key\_id of ‘1val’  
}

inline\_key is a MetadataKeyBox where local\_key\_id is set to ‘1key’ (for “one key”).

inline\_value is a MetadataAUBox where local\_key\_id is set to ‘1val’ (for “one value”).

The MetadataInlineKeyValueAUBox can be viewed as a MetadataAUBox with two differences:

* It is a container box carrying two boxes, one a MetadataKeyBox holding the key and the other a MetadataAUBox holding the value for the metadata item.
* It has a local\_key\_id (or box type) with the special value 0xFFFFFFFF. All inline key/value boxes share this special local\_key\_id of 0xFFFFFFFF regardless of the contained value’s key.

Because a MetadataInlineKeyValueAUBox carries both the key and the value using that key, this box alone is sufficient to carry what would otherwise require a MetadataAUBox and an associated BoxedMetadataSampleEntry with a MetadataKeyTableBox having the same local\_key\_id as the MetadataAUBox. This allows any non-inline key and associated value to be converted to an inline key/value box. The reverse transform (inline key/value box to non-inline value and sample entry) is possible, too.

While possible, writing a MetadataInlineKeyValueAUBox declaring a key that’s also declared within the MetadataKeyTableBox (i.e., it carries a duplicate MetadataKeyBox) is strongly discouraged. The presence of a MetadataInlineKeysPresentBox signaling the presence of inline keys defeats optimizations that are possible when all available keys are declared within the MetadataKeyTableBox and no inline keys are used. Using inline keys should be reserved for cases when the keys to be written cannot be known at the time the sample entry is constructed. Section xx also discusses the use of inline keys.

## Using sample groups to optimize key searches

### General

This section describes an optional mechanism to optimize searches for metadata track access units containing particular key/value pairs. While this mechanism’s support is not required, it allows a reader to locate access units with values for a key without having to read each access unit in the track and scan for the key’s value. This can be useful for some kinds of metadata (i.e., values that don’t occur in every metadata access unit) but for others (e.g., GPS metadata) it may not add much value. Different tracks in the same movie may choose to use or not to use this optimization.

If inline key/value boxes are used, the mechanism described here can be used to locate those access units with inline key/value boxes. This can be useful in limiting the scan for metadata items with keys only found in inline key/value boxes.

The following subsections describe the details.

### Sample group overview

A metadata track conforming to this specification may optionally make use of the SampleGroupDescriptionBox and SampleToGroupBox constructs to optimize searching for access units containing particular keys or inline keys. This can be characterized as having a ‘key search sample group.’

### Optimizing search with a sample group

In a metadata track containing one or more sample entries, the MetadataKeyTableBox() in the BoxedMetadataSampleEntry can be used to determine possible keys present in the track’s AUs. If a key is not present in the MetadataKeyTableBox(), it is known that the key doesn’t exist in any AUs. It doesn’t however indicate which samples have particular keys (and associated values). Therefore, to determine which metadata keys are present in the track requires an exhaustive search of AUs (associated with that sample entry) in the metadata track.

While it would be possible to create a track with sample entries for each combination of keys present in the track and only associate the samples with that combination with the particular sample entry, having many sample entries may not be ideal or easily done. An alternative (described here) is to define a new kind of sample group that indicates the keys present in each AU.

The new sample group consists of a SampleGroupDescriptionBox holding a new group description for each new combination of keys present in AUs. If all AUs consist of the same four keys, for example, there would be one group description with these four keys. If the set of keys varied, there need only be as many descriptions as there are different sets of keys present in AUs.

A client looking for AUs with a particular key (or keys) would first consult the sample entry (or sample entries if there are more than one) and determine if the key is present in the set of possible keys (via MetadataKeyTableBox()). If this succeeds, the client would check if the optional sample group exists, and finding this to be the case, the client would walk through the SampleToGroupBox checking if the corresponding sample group description contains the key. As these operations require only information present in the MovieBox(), direct reading and processing of AUs is unnecessary.

NOTE: While “key” is used here as being present in the sample group description, an equivalent, more compact identifier is used.

### The keysearch sample group

For this specification, an optional sample group known as a “key search sample group” is defined. It consists of SampleGroupDescriptionBox and SampleToGroupBox having the grouping type ‘keyp’.

The SampleGroupDescriptionBox will contain variable-sized SampleGroupDescriptionEntries, each of type MetadataKeySearchGroupEntry. MetadataKeySearchGroupEntry is defined as:

class MetadataKeySearchGroupEntry() extends SampleGroupDescriptionEntry (‘keyp’) {  
 unsigned int(32) entry\_count;  
 unsigned int(32) local\_key\_ids\_array[entry\_count];   
}

entry\_count is a 32-bit unsigned integer holding the number local key ids that follow in local\_key\_ids\_array[].

local\_key\_ids\_array is an array of 32-bit integers corresponding to the local\_key\_id field used in the associated MetadataKeyTableBox() and the local key ids used in associated metadata track access units. A value of 0 is reserved and can be used to mark an array entry as absent. A value of 0xFFFFFFFF indicates the associated access units all contain one or more inline key/value boxes.

Each sample group description entry signals the presence of one or more keys from the key table found in the sample entry associated with the sample(s). Access units associated with this sample group description shall have corresponding metadata values with these same keys.

Each key in use is signaled by using the 32-bit integer value of the local\_key\_id field associated with the MetadataKeyTableBox entry. This local key id is also used in access units as the type of Box holding the corresponding value.

If two samples differ in the keys present, they cannot share the same sample group description. A sample group description for each combination should be created.

NOTE 1 While not strictly required, it is recommended that the order of local\_key\_ids be the same as the order of local key ids in the MetadataKeyTableBox of the sample entry. This can be followed by 0xFFFFFFFF if there is an inline key present. This prevents group descriptions with the same set of keys but differing only in key order from creating multiple, trivially different sample group descriptions.

NOTE 2 There is no relationship between the order of keys in the MetadataKeySearchGroupEntry and the order of values for those keys in the associated access unit(s).

A version 0 SampleGroupDescriptionBox should not be used.

Finally, if a sample group spans multiple sample entries with different sets of keys, the local key ids present in the sample entries spanned must be compatible (i.e., the local\_key\_id must be present in each MetadataKeyTableBox and the corresponding key table entry must be the same). An easy way to accomplish this is not to have samples from different sample entries share the same MetadataKeySearchGroupEntry.

## Structurally dependent metadata

### General

If the metadata values in a metadata track are dependent upon another track in a way that a change in the other track may invalidate some or all metadata item values, the dependent metadata items and metadata track itself are termed structurally dependent upon the other track. Structurally dependent metadata tracks are linked to the track upon which they are structurally dependent using a track-reference of type ‘cdep’. The ‘cdep’ track-reference should be used in addition to the ‘cdsc’ track reference because the ‘cdep’ track-reference’s purpose is to indicate tracks needing attention if another track is transformed (e.g., a video track being scaled or cropped during a reencode to a new file where the metadata will also be copied). Tracks having a ‘cdsc’ track-reference without a ‘cdep’ track-reference can be passed through directly so long as there are no other conditions restricting pass-through. Metadata tracks having a ‘cdep’ track-reference may also need to have metadata items transformed or deleted due to the change in the other track (e.g., the video track).

### MetadataStructuralDependencyBox

The interpretation of a metadata value may depend upon a detail of another track. For example, a geometric value such as a region of interest may be interpreted in the coordinate system of a video track. If the other track is transformed (e.g., scaled or cropped in the case of video), the metadata value may become invalid. To signal which metadata items are structurally dependent, a MetadataStructuralDependencyBox() should be present in MetadataKeyBox() for such metadata items.

The metadata track should also have a ‘cdep’ track reference to the other track upon which values are structurally dependent. This reference is used to determine which metadata tracks might need attention if the target track is manipulated.

If the other track is transformed, currently one of three things can occur to the dependent metadata items:

* + If the values can be transformed in response to the change in the other track, metadata values can be read, transformed, and written in their corrected form. This will typically be limited to being performed in a process that reads the source movie and writes a new one. This requires that the code performing this transformation be able to understand the change in the other track and how to transform the dependent metadata values.
  + If the values cannot be transformed, the metadata item can be deleted by removing the MetadataKeyBox() from the MetadataKeyTableBox() (i.e., setting the local id to 0 and optionally removing the MetadataKeyBox()) and optionally removing metadata values from associated access units. Again, this will typically be limited to being performed in a process that reads the source movie and writes a new one. This removal should be avoided if possible but is available for cases where the transform is not understood, the metadata values are not understood, or the change cannot be applied to understood values.
  + Another option is to mark the structurally dependent value as invalid without rewriting access units or deleting the metdata item from the key table. This allows the presence of metadata items known to have become invalid. Clients reading structurally dependent but invalid items may choose to ignore these or do whatever they see fit. They should not treat them as valid.

The MetadataStructuralDependencyBox() is a Box with this definition:

aligned(8) class MetadataStructualDependencyBox extends Box('sdpd') {   
 MetadataStructuralDependencyInfoBox info;  
}

MetadataStructuralDependencyInfoBox is a FullBox having one currently defined flag value. This flag can be used to mark a key table entry as invalid under the structural dependency.

Other children boxes may be introduced in the future.

The MetadataStructuralDependencyInfoBox is a FullBox with this definition:

aligned(8) class MetadataStructualDependencyInfoBox   
 extends FullBox('sdpi', 0, flags) {  
};

flags can have the lowest order bit (i.e., 0x000001) set to indicate the structural dependency is invalid. If this bit is clear, the metadata item and associated values should be considered valid.

Newly written MetadataStructuralDependencyBox() should have this flag in their contained MetadataStructuralDependencyInfoBox() be clear. New flags may be introduced in the future.

NOTE: In the future, other children boxes of MetadataStructuralDependencyBox() may be introduced. The current thinking is that these will help clients understand under which kinds of transforms the values might remain valid. For example, a video scaling where metadata values use a normalized range of 0...1 might not need to be transformed. If the metadata values used pixels, they might however need to be transformed.

# Embedded Metadata Signaling

It would be beneficial to signal the presence of the embedded metadata at the file level.

We propose to introduce a new SampleGroupDescription box entry that signals the presence of metadata of a particular type. That sample group entry is then used with SampleToGroup boxes to signal the location of the samples that contain the related metadata. The type of the metadata is signalled using a URN scheme that uniquely identifies the metadata.

We define a new SampleGroupEntry with the following syntax and semantics:

class MetadataSampleGroupEntry ()   
 extends SampleGroupDescriptionEntry ('emmd')  
{  
 string metadata\_type;  
}

metadata\_type: a URN that uniquely identifies the type of metadata that is carried.

Additional metadata-specific information may also be provided in this sample group description entry.

[[ed: We could do with an example, to illustrate the usage. Would we allow further parameters, dependent on the URN, after the URN (whereupon we should box the URN)? There is a sample group rather like this in the timed metadata multiplex; should we harmonize? We need the 4CC defined (e.g. 'emmd' for embedded metadata?).]]

# On MIME type parameters

## Introduction

This section discusses issues and solutions for signalling of important video information, as part of the MIME type parameters, for HDR/WCG video, and video with display orientation changes. Hereafter, important video information refers to video information that may be used for content selection, e.g., selection of a video track or a part thereof for consumption.

### Video with display orientation changes

AVC and HEVC both support video content for which the decoder side should apply a transformation of rotation and/or flipping to the cropped decoded picture prior to display, indicated by the display orientation SEI message. Such video is also referred to as video with display orientation changes. Video with display orientation changes need special post-decoding rendering processing to generate desirable viewing experience.

### Signalling of HDR/WCG information in ISOBMFF

The HDR/WCG information can be signalled using the ColourInformationBox defined in clause 12.1.5 of the ISOBMFF specification, for example with the colour\_type equal to ‘nclx’, in which case the most important HDR/WCG information would be carried in the fields colour\_primaries, transfer\_characteristics, matrix\_coefficients, and full\_range\_flag.

In addition, certain HDR/WCG solutions make use of dynamic metadata conveyed in SEI messages.

### The restricted scheme design in ISOBMFF

The restricted scheme design in ISOBMFF is for handling of situations where the file author requires certain actions on the player or renderer, to enable players to simply inspect a file to find out such requirements for rendering a bitstream and stops legacy players from decoding and rendering files that require further processing. The mechanism applies to any type of video codec.

The mechanism is similar to the content protection transformation where sample entries are hidden behind generic sample entries, ‘encv’, ‘enca’, etc., indicating encrypted or encapsulated media. The analogous mechanism for restricted video uses a transformation with the generic sample entry ‘resv’. The method may be applied when the content should only be decoded by players that present it correctly.

The restricted scheme is specified in clauses 8.15.1 to 8.15.3 of the ISOBMFF specification.

## Problems and discussions

[Ed. (FM): Issues and solutions in this section were initially introduced as Items 6 (first part) and 7 from m40373 (MPEG #118 meeting)]

The following problems related to the MIME type parameters and signalling of HDR/WCG video, and video with display orientation changes were observed:

1. For video with display orientation changes, a special restricted scheme is missing.
2. There lacks a mechanism to include important video information for HDR/WCG video as part of the MIME type parameters.

## Proposal

The following methods are proposed to solve problems:

1. To solve the first problem, a new restricted scheme type, 'vdoc', is defined, which, when used, indicates that the track carries video with display orientation changes. No further information is provided, and the SchemeInformationBox may be absent in the RestrictedSchemeInfoBox.
2. To solve the second problem, a new optional MIME type parameter 'hdrinfo' is defined to contain the important information of HDR/WCG video. The format of this optional MIME type parameters is a single value or a comma-separated list of values, where each value consists of one or more dot-separated elements. A value of the 'hdrinfo' parameter contains four fields, in the form of "elment1.elment2.elment3.elment4", where the four elements 1 to 4 are the hexadecimal representations of the fields colour\_primaries, transfer\_characteristics, matrix\_coeffs, and full\_range\_flag, respectively, as defined in clause 12.1.5 of the ISOBMFF specification.

# On relation of entity groups and sample groups

It is possible to specify an entity group and a sample group that use the same four-character code as their type and mutually contribute to indicate static and dynamic information for a specific purpose. Such a pairing of an entity group and a sample group is a documented characteric of a specific four-character code. Common static information should be contained in an entity group with a particular group\_id value, and dynamic information should be contained in a sample group of the same type and with the value of grouping\_type\_parameter equal to the value of group\_id.

# Improved sub-sample information

We are concerned that the subsample information box mandatorily carries bytes that may not always be needed (6 per sample) and suggest a revision to make them optional.

Unfortunately we seem to have partially given away the flags to specific coding systems. It's not cleat if the fix here (reclaiming them) will work.

Note that the revision could be (almost) backwards-compatible if we were to make the default value of the flags (flag==0) say that the fields are present, but this would be abnormal; it also doesn't help old readers read files in which the flags are set and the fields are absent.

We note that we could compact more if we could use pattern-based compression (many samples might have the same pattern). The pattern could be subsample sizes, or the size of the subsample size field (initial SEIs might always be small and only need 1-byte sizes).

## Sub-Sample Information Box

### Definition

Box Type: 'subs'  
Container: SampleTableBox or TrackFragmentBox  
Mandatory: No  
Quantity: Zero or more

This box is designed to contain sub-sample information.

A sub-sample is a contiguous range of bytes of a sample. The specific definition of a sub-sample shall be supplied for a given coding system (e.g. for ISO/IEC 14496-10, Advanced Video Coding). In the absence of such a specific definition, this box shall not be applied to samples using that coding system.

If subsample\_count is 0 for any entry, then those samples have no subsample information and no array follows. The table is sparsely coded; the table identifies which samples have sub-sample structure by recording the difference in sample-number between each entry. The first entry in the table records the sample number of the first sample having sub-sample information.

NOTE It is possible to combine subsample\_priority and discardable such that when subsample\_priority is smaller than a certain value, discardable is set to 1. However, since different systems may use different scales of priority values, to separate them is safe to have a clean solution for discardable sub-samples.

When version is 0 or 1, then when more than one SubSampleInformationBox is present in the same container box, the value of flags shall differ in each of these SubSampleInformationBoxes. The semantics of flags, if any, shall be supplied for a given coding system. If flags have no semantics for a given coding system, the flags shall be 0.

If version is 2, the flags field is used by the box definition and not available to derived coding systems. Instead, a coding\_flags field may be present with the same semantics.

When the version is 0 or 1, the extra fields are mandatory and the size of the subsample field is indicated by the version. When the version is 2, the size of the subsample field and the presence of the other fields are indicated by flag bits:

1 when 0 defines 16-bit subsample sizes, when 1 defines 32-bit sizes

2 subsample\_priority absent when 1

4 discardable absent when 1

8 codec\_specific\_parameters absent when 1

16 coding\_flags absent when 1

### Syntax

aligned(8) class SubSampleInformationBox  
 extends FullBox('subs', version, flags) {  
 unsigned int(32) entry\_count;  
 int i,j;  
 for (i=0; i < entry\_count; i++) {  
 unsigned int(32) sample\_delta;  
 unsigned int(16) subsample\_count;  
 if (subsample\_count > 0) {  
 for (j=0; j < subsample\_count; j++) {  
 if(version == 0)  
 {  
 unsigned int(16) subsample\_size;  
 }  
 else if (version == 1)  
 {  
 unsigned int(32) subsample\_size;  
 }  
 else if (version==2)  
 {  
 if ((flags & 1)==0)  
 unsigned int(16) subsample\_size;  
 else  
 unsigned int(32) subsample\_size;  
 }  
 if (version < 2) {  
 unsigned int(8) subsample\_priority;  
 unsigned int(8) discardable;  
 unsigned int(32) codec\_specific\_parameters;  
 }   
 else // version>=2  
 {  
 if ((flags & 2)==0) unsigned int(8) subsample\_priority;  
 if ((flags & 4)==0) unsigned int(8) discardable;  
 if ((flags & 8)==0)   
 unsigned int(32) codec\_specific\_parameters;  
 if ((flags & 16)==0)   
 unsigned int(24) coding\_flags;  
 }  
 }  
 }  
 }  
}

### Semantics

version is an integer that specifies the version of this box (0 or 1 in this specification)

entry\_count is an integer that gives the number of entries in the following table.

sample\_delta is an integer that indicates the sample having sub‐sample structure. It is coded as the difference, in decoding order, between the desired sample number, and the sample number indicated in the previous entry. If the current entry is the first entry in the track, the value indicates the sample number of the first sample having sub-sample information, that is, the value is the difference between the sample number and zero (0). If the current entry is the first entry in a track fragment with preceding non-empty track fragments, the value indicates the difference between the sample number of the first sample having sub-sample information and the sample number of the last sample in the previous track fragment. If the current entry is the first entry in a track fragment without any preceding track fragments, the value indicates the sample number of the first sample having sub-sample information, that is, the value is the difference between the sample number and zero (0). This implies that the sample\_delta for the first entry describing the first sample in the track or in the track fragment is always 1.

subsample\_count is an integer that specifies the number of sub-sample for the current sample. If there is no sub-sample structure, then this field takes the value 0.

subsample\_size is an integer that specifies the size, in bytes, of the current sub-sample.

subsample\_priority is an integer specifying the degradation priority for each sub-sample. Higher values of subsample\_priority, indicate sub-samples which are important to, and have a greater impact on, the decoded quality.

discardable equal to 0 means that the sub-sample is required to decode the current sample, while equal to 1 means the sub-sample is not required to decode the current sample but may be used for enhancements, e.g., the sub-sample consists of supplemental enhancement information (SEI) messages.

coding\_flags provides the same value, when version is 2, as the flags field in the box when version is 0 or 1

codec\_specific\_parameters is defined by the codec in use. If no such definition is available, this field shall be set to 0.

# Non-independent tracks

## Comment

OMAF clause 7.1.4 specifies indication of a track not intended to be presented alone as a generic ISOBMFF extension. Bit 4 of the flags (with bit 0 being the least significant bit) of the TrackHeaderBox is used to indicate whether a track is not intended to be presented alone, e.g., due to that the track represents only a small portion of a videos scene.

## Proposed change

Include in the semantics of flags of TrackHeaderBox (8.3.2.3) the following text that is copied from OMAF clause 7.1.4:

track\_not\_intended\_for\_presentation\_alone: Indicates that the track is not intended to be presented alone without other tracks. Flag value is 0x000010. The flag not being set (i.e., flags & 0x000010 == 0) indicates that the track may or may not be intended to be presented alone without other tracks.

## Discussion

The not-alone flag seems hard to document; what exactly is "with something else"? And who is this flag for (not old readers, obviously)? If my viewport into an omni video needs only this track, am I violating this flag or not?

# On just-in-time delivery of items

[Ed. (MH): This section was authored on the basis of the meeting notes on m42584 in MPEG #122.]

## Introduction

It can be desirable to deliver image items in a "just-in-time manner" (e.g. in DASH media segments or CMAF Segments) at the time when they are referenced by an HTML page in the MetaBox (non-timed). The following sub-sections include options.

## Usage of URL fragment identifier

The following URL or URL fragment identifier schemes are under consideration to be used within the web resource tracks for concluding that images referenced by the URL are to be loaded "just-in-time":

1. Referencing an independent image file.

http://example.com/images/image1.jpg

1. Usage of the item\_name fragment identifier as specified in Annex L of ISO/IEC 14496-12.

Example: http://example.com/segments/segment23.mp4#item\_name=image1.jpg

1. Specifying a DASH MPD specific fragment identifier. The segment URL by indexing through the MPD.

Example: http://example.com/videos/video.mpd#presentation\_time=23s&item\_name=image1.jpg

## Bundle item information box

[Ed. (MH): Meeting notes on m42584 in MPEG #122: The bundle seems like an optimization for a specific set of cases; this assumes that the metabox declares all items, but they are referenced possibly by data references or their offsets are later in the file. This is a loading hint for readers that do lazy loading; it suggests that resources will be used at the same time.]

### Definition

Box Type: 'binf'  
Container: MetaBox  
Mandatory: No  
Quantity: Zero or more

The BundleInfoBox provides an array of bundle item information entry box of the items used as a bundle. It means that all items in a bundle are used when all extent files of the bundle items are delivered. Note that extent files of the bundle may be delivered in different movie fragments. For example, a resource type of primary item is ‘text/html’ or ‘application/xhtml+xml’ and the primary item, i.e. an entry page, is specified in the BundleInfoBox, the entry page is loaded when all the other resources in the BundleInfoBox are delivered.

### Syntax

aligned(8) class BundleItemInfoEntry extends FullBox(‘bife’, version=0, 0) {  
 unsigned int(16) item\_ID;  
}

aligned(8) class BundleInfoBox extends FullBox(‘binf’, version=0, 0) {  
 unsigned int(32) entry\_count;  
 BudleItemInfoEntry[ entry\_count ] bundle\_item\_infos;  
}

### Semantics

item\_ID is the identifier of the item in a bundle.

# Movie Fragment Compaction considerations

## Inheritance within a track run

The addition of zero or more RepeatStruct structures to the end of the CompactTrackRunBox is proposed. The number of RepeatStruct structures is determined by a file writer. The RepeatStruct is summarized as follows:

1. The box contains another RepeatStruct structure if the end of the box has not been reached yet. The function EndOfBox( ) returns 0, if the end of the box has not been reached yet, and returns 0 otherwise.
2. Each RepeatStruct contains:
   1. The number of times a pattern is repeated (repeat\_count\_minus1 + 1)
   2. A starting sample index within the track run (repeat\_start)
   3. The length of the pattern (repeat\_period\_minus1 + 1)
3. The values of syntax elements are either copied from the sample in the pattern or is present, as controlled in RepeatStruct. For example, the structure can only have sample sizes present and inherent all other syntax element values. Size index fields are present for each syntax element (sample duration, sample size, sample flags, and composition time offset). When size index field is equal to 0, the respective syntax element value is copied from the indicated pattern. When the syntax element is present in the structure, the syntax element length can be indicated to be 8, 16, or 32 bits with the size index field equal to 1, 2, or 3, respectively.

## Proposed syntax

aligned(8) class CompactTrackRunBox  
 extends FullBox('ctrn', version, tr\_flags) {  
  
... // add here the CompactTrackRunBox as in PDAM  
  
 while (!EndOfBox())  
 RepeatStruct();  
 }  
}

aligned(8) class RepeatStruct() {  
 unsigned int(8) repeat\_count\_minus1;  
 if(sample\_count < 256) rs\_len = 8;  
 else if (sample\_count < 65536) rs\_len = 16;  
 else rs\_len = 32;  
 unsigned int(rs\_len) repeat\_start;  
 unsigned int(7) repeat\_period\_minus1;  
 unsigned int(1) exp\_size\_idx\_flag;  
 if( exp\_size\_idx\_flag ) {  
 unsigned int(2) dur\_size\_idx;  
 unsigned int(2) siz\_size\_idx;  
 unsigned int(2) fgs\_size\_inx;  
 unsigned int(2) cto\_size\_idx;  
 }  
 else {  
 // values inferred  
 dur\_size\_idx = 0;  
 siz\_size\_idx = sample\_size\_index;  
 fgs\_size\_inx = 0;  
 cto\_size\_idx = 0;  
 }  
 for (cnt = 0; cnt <= repeat\_count\_minus1; cnt++){  
 for (i = 0; i <= repeat\_period\_minus1; i++ ) {  
 // function f() the same as in TuC  
 unsigned int(f(dur\_size\_idx)) exp\_sample\_duration;  
 unsigned int(f(siz\_size\_idx)) exp\_sample\_size;  
 unsigned int(f(fgs\_size\_idx)) exp\_sample\_flags;  
 if (version == 0)   
 unsigned int(f(cto\_size\_idx)) exp\_sample\_duration;  
 else  
 signed int(f(cto\_size\_idx)) exp\_sample\_duration;  
 }  
 sample\_count += repeat\_period\_minus1 + 1;  
 }  
}

## General Compaction

Should we provide for simple GZIPping (as is done in the QuickTime cmov atom) of the contents of the Movie Box and Movie Fragment box?

Such work would need to take into account the question of what offsets outside the gzipped content mean (e.g. segment index boxes). It's clear that transport level compression means excess overhead since it would compress the mdat boxes, which are typically 'dense'. But the complexities of compressing even only top-level boxes, when some of them may be segment indexes, becomes tricky, especially when considering backwards-compatibility, and the fact that indexes may be out-of-line, in separate files.

Perhaps we can do two-pass: compress the moof boxes, calculate offsets and use them in segment indexes, and then optionally compress the sidx box. (We note that sidx boxes lend themselves to compression since they can get large.)

consider

ftyp ftyp

– cmap

sidx *csid*

meta meta

moof *cmof*

mdat mdat

moof *cmof*

mdat mdat

do the offsets in the meta change? in the sidx? We note that the meta problem is soluble using identified media data boxes.

## Compaction under encryption

There seems to be a need to optimize first the sample encryption information (see m45164).

# Stronger defaulting in Track Runs

## Discussion

Many file writers operate by parsing the high-level syntax of a given input video bitstream and generating the file format metadata from the information of the bitstream. A backward-compatible approach could be achieved by letting advanced clients

* omit the downloading of MovieFragmentBoxes, and
* create the MovieFragmentBoxes in the client side by parsing the high-level syntax of the received MediaDataBoxes.

At the same time, the transmitted track run data is reduced to 0 bytes, i.e. ultimate compression of MovieFragmentBox metadata is achieved.

## Overview

A set of indications is proposed based on which a player can conclude that it is able to process the media data without the MovieFragmentBox. Thus, even though the MovieFragmentBox is available, the player does not need to fetch and process it. Hence, this is a backward compatible approach for avoiding the delivery of MovieFragmentBox.

The following indications are proposed:

* Either of the following indications is included in the ISOBMFF:
  + Indicate with a box flag in a data reference box that all MediaDataBoxes that are referenced through the data reference entry are "tigthly packed", i.e. contain samples for a single track only in decoding order without unused bytes, sample auxiliary information, metadata, or any other information that does not belong to the sample format. See Section ‎13.4.
  + Indicate with a box flag of the SegmentIndexBox that the media data box(es) carrying the data for the described segment are "tightly packed". See Section ‎13.6.
* The following indication is included in the NAL unit file format (ISO/IEC 14496-15):
  + Indicate with a box in the sample entry how picture timing can be derived. The box includes a multiplication factor that applies to convert picture order count differences to composition times. For a picture that starts a new coded video sequence, the box indicates the composition time difference from the last picture of the previous coded video sequence. See Section ‎13.5.
* For usage with DASH, either of the following approaches can be used:
  + Extensions of the SegmentIndexBox indicate the referenced metadata size (mainly MovieFragmentBoxes) or offset(s) to the media data and can be used to conclude the byte ranges of MediaDataBoxes. See alternative approaches in Sections ‎13.6 and ‎13.7.
  + The following indications are included in the DASH MPD: Indicate the byte ranges or URLs for requesting the MediaDataBoxes only with a new MPD attribute. See Section ‎13.8.

A player can operate as follows:

* Conclude from the flag indicating "tightly packed" media data boxes that reception of MovieFragmentBoxes is not necessary.
* Use the indicated byte ranges or URLs for requesting MediaDataBoxes only.
* Generate the MovieFragmentBoxes based on parsing the high-level syntax of the bitstream in the received MediaDataBoxes or directly decodes and plays the bitstream without deriving the file format metadata. In this operation, the information of the TrackRunBox is concluded based on the received MediaDataBox for a movie fragment as follows:
  + By carrying out the access unit boundary determination as specified in AVC or HEVC, the sample sizes and the sample count can be derived.
  + Picture composition timing may be provided in the bitstream (picture timing SEI message) or it is concluded that composition times are proportional to picture order count. Values of sample\_duration are derived accordingly.
  + VCL NAL unit type can be used to determine sample flags or the sample flags can be set to indicate an unknown status.

## Analysis

### Responses to comments at MPEG#127

In the following, we copy the comments from the File Format minutes of MPEG#127 (labeled Cx) and provide our answers to them (Ax).

C1. It seems that in the case that the URLs (e.g. in an MPD) resolve to "pure media data" one would need a new MIME type (not the one for an ISO segment). In that case, is it really in scope for the file format? (See bullet below).

A1. The scheme is primarily intended for on-demand streaming (e.g. ISO base media file format on-demand profile of DASH) where all SegmentIndexBox(es) are placed before any MovieFragmentBox. No changes in the segment formats are proposed and hence no new MIME types are needed either. The media data is selectively fetched using HTTP GET requests with byte ranges that are concluded from the SegmentIndexBox(es) with extensions specified in the proposed scheme.

C2. The 'pain' is not the file format overhead, but the implementation complexity (and edge cases). We need to evaluate this.

A2. We implemented options for providing media data offsets in MP4Box and tested the reader compatibility of the options with several readers. See Section ‎13.3.2 for details.

C3. This relies on getting somehow getting a segment index (either in-band, e.g. after the moov box, or out of band) at the client:

* because the representation is ftyp-moov-mdat-mdat-mdat…
* because the representation is ftyp-moov-moof+mdat-moof+mdat-moof+mdat… and the sidx tells you the byte-range requests in each segment to omit the moof box

A3. Right, we assume that sidx(es) are placed before any moof, which is required e.g. in ISO base media file format on-demand profile of DASH. Thus, the file structure would be ftyp-moov-sidx(es)-moof+mdat-moof+dat-moof+dat…

### Tests on reader compatibility on extended SegmentIndexBox and SegmentMediaOffsetBox

This section provides results of the reader compatibility tests of the options in the TuC.

Tests were carried out by segmenting a video clip with one AVC media track by differently modified MP4Box programs. Files were then served to players from an HTTP server.

Three different MP4Box versions were used:

* Unmodified MP4Box
* Modified MP4Box which adds SegmentMediaOffsetBox ('smof') after SegmentIndexBox
* Modified MP4Box which sets flags to 1 in SegmentIndexBox and adds media\_data\_offset fields to the end of the SegmentIndexBox.

As seen in Table 1, extended SegmentIndexBox did not introduce any degraded functionality in any of tested players, compared to the unmodified input files.

In browsers, Dash.js reference client relies on browser-side Media Source Extensions (MSE) to handle segment data parsing. Both Firefox and Chromium browsers use MSE implementations which stop segment processing with an error, if they encounter an unknown root-level box. This prevents playback of files which contain SegmentMediaOffsetBox.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Media player | Unmodified | Added 'smof' box | Extended 'sidx' box | Notes |
| Vlc (3.0.8) | OK | OK | OK |  |
| MP4Client (GPAC 0.8.0) | OK | OK | OK | Log messages about extra bytes in 'sidx' box. |
| ffplay (ffmpeg 4.1.3) | OK | OK | OK |  |
| Android Exoplayer (2.10.4) | OK | OK | OK |  |
| Dash.js reference client (3.0.0) running in Firefox browser (69.0.1) | OK | Not playing. Error because of unknown root-level box. | OK |  |
| Dash.js reference client (3.0.0) running in Chromium browser (76.0.3809.100) | OK | Not playing. Error because of unknown root-level box. | OK |  |
| Dash.js reference client (3.0.0) running in Safari browser (13.0.1) | OK | OK | OK |  |

Table 1: Playback test results

## Flag in data reference box

The following is proposed to be added into clause 8.7.2.1 of ISOBMFF:

When (flags & 0x000002) is greater than 0 in a data reference box, all MediaDataBoxes that are referenced through the data reference box are "tigthly packed", i.e.:

* + MediaDataBoxes contain samples for a single track only.
  + The samples are in decoding order within a MediaDataBox.
  + MediaDataBoxes contain no unused bytes, sample auxiliary information, metadata, or any other information that does not belong to the sample format.

## Box in 14496-15

*Add the following clause 4.13:*

**4.13 Indicating composition times directly proportional to picture order counts**

**4.13.1 Definition**

Box Type: 'reti'  
Container: Sample Entry   
Mandatory: No  
Quantity: Zero or one

The presence of this box specifies that the composition time difference of any two consecutive pictures in output order in the same coded video sequence is directionally proportional their picture order count difference.

**4.13.2 Syntax**

class RelativeTimingBox extends FullBox ('reti', version=0, flags=0) {  
 unsigned int(32) poc\_unit\_duration;  
 unsigned int(32) cvs\_start\_interval;  
}

**4.13.3 Semantics**

poc\_unit\_duration specifies the composition time difference that corresponds to a picture order count difference equal to 1.

cvs\_start\_interval specifies the composition time difference of the first picture of each coded video sequence, in output order, relative to the last picture of the previous coded video sequence, in output order.

NOTE: In practice, cvs\_start\_interval is the sample duration of the last picture of each coded video sequence.

## Extension of the segment index box

### Overview

Figure 1 below illustrates new versions of the extended segment index box ‘sidx’. In these new versions of the segment index box, when indexing fragments (i.e. reference\_type=0), two indexes can be stored per fragment (instead of a single one currently: referenced\_size).



Figure : New version of ‘sidx’

As illustrated in the new ‘sidx’ syntax below, the first index is associated with the actual data of the considered fragment while the second index is associated with the metadata of this fragment.

*Add the following at the end of clause 8.16.3.1 (definition of SegmentIndexBox):*

The flags field has the following semantics:

(flags & 1) equal to 1 specifies that the referenced segments are constrained as follows:

* + There is a single referenced MediaDataBox or IdentifiedMediaDataBox for references with reference\_type equal to 0.
  + The referenced MediaDataBox or IdentifiedMediaDataBox contains samples for a single track only.
  + The samples are in decoding order within the referenced MediaDataBox or IdentifiedMediaDataBox.
  + The referenced MediaDataBox or IdentifiedMediaDataBox contains no unused bytes, sample auxiliary information, metadata, or any other information that does not belong to the sample format.

NOTE 1: Since encryption requires extra data to be stored with samples, it is not possible to use (flags & 1) equal to 1 with encrypted media data.

NOTE 2: When (flags & 1) is equal to 1, and media samples are either self-framing or of constant size (indicated by flags & 4), and the sample times are either predictable (indicated by flags & 2) or calculable, it is possible to process the referenced MediaDataBox or IdentifiedMediaDataBox without the MovieFragmentBox. Clause J.2.5 provides background and rationale for using flags and contains an example of a file structure.

(flags & 2) equal to 2 specifies that the referenced segments are constrained so that default\_sample\_duration of TrackExtendsBox applies to each sample and that sample\_composition\_time\_offset is equal to 0 for each sample.

(flags & 4) equal to 4 specifies that the referenced segments are constrained so that default\_sample\_size of TrackExtendsBox applies to each sample.

*Change the syntax of the* SegmentIndexBox *in clause 8.16.3.2 to the following (i.e., replacing* 0 *with* flags *in the box header, and adding the parts conditioned by the value of* flags*):*

aligned(8) class SegmentIndexBox extends FullBox('sidx', version, flags) {  
 unsigned int(32) reference\_ID;  
 unsigned int(32) timescale;  
 if (version==0) {  
 unsigned int(32) earliest\_presentation\_time;  
 unsigned int(32) first\_offset;  
 }  
 else {  
 unsigned int(64) earliest\_presentation\_time;  
 unsigned int(64) first\_offset;  
 }  
 unsigned int(16) reserved = 0;  
 unsigned int(16) reference\_count;  
 for(i=1; i <= reference\_count; i++)  
 {  
 bit (1) reference\_type;  
 unsigned int(31) referenced\_size;  
 unsigned int(32) subsegment\_duration;  
 bit(1) starts\_with\_SAP;  
 unsigned int(3) SAP\_type;  
 unsigned int(28) SAP\_delta\_time;  
 }  
 if (flags & 1)  
 for(i=1; i <= reference\_count; i++)  
 if (reference\_type == 0) // reference\_type of the same i value  
 unsigned int(32) media\_data\_offset;  
}

*Add the following to the end of clause 8.16.3.3:*

media\_data\_offset specifies the offset to the start of the referenced MediaDataBox or the IdentifiedMediaDataBox of a subsegment from the start of the subsegment.

## Alternative improvements to the SegmentIndexBox

### Option 1

To support the different indexing modes, the semantics of reference\_type is extended as follows (highlighted in yellow):

* when set to 1 indicates that the reference is to a SegmentIndexBox; otherwise the reference is to media content as follows:
* when set to 0 indicates content including both metadata and media data (e.g., in the case of files based on this document, to a MovieFragmentBox);
* when set to 2 indicates content including metadata only (e.g., in the case of files based on this document, one or more MovieFragmentBox);
* when set to 3 indicates content including media data only (e.g., in the case of files based on this document, one or more MediaDataBox or IdentifiedMediaDataBox);

if a separate index segment is used, then entries with reference type 1 or 2 are in the index segment, and entries with reference type 0 or 3 are in the media file;

In this option a new version of the segment index box requires two bits for the representation of the reference\_type as illustrated below. The referenced\_size field in the new version is interpreted according to the following values of the reference\_type:

* When set to 0, the referenced\_size is the distance in bytes from the first byte of the referenced index to the first byte of the next referenced index (moof) item.
* When set to 1, the referenced\_size is the distance in bytes from the first byte of the referenced index to the first byte of the next referenced index (sidx) item.
* When set to 2, referenced\_size is the distance in bytes from the first byte of the referenced metadata item to the first byte of the next referenced index metadata item, or in the case of the last entry, the end of the referenced index metadata item.
* When set to 3, referenced\_size is the distance in bytes from the first byte of the referenced data item to the first byte of the next referenced index data item, or in the case of the last entry, the end of the referenced index data item .

The value of subsegment\_duration of each entry with reference\_type equal to 2 or 3 corresponds to the duration of the indexed sub-segment. When the reference\_type is set to 1, the semantics of the subsegment\_duration is the same as in ISOBMFF Table J.3.

### Option 2

#### Syntax

aligned(8) class SegmentIndexBox extends FullBox('sidx', version, flags) {  
 unsigned int(32) reference\_ID;  
 unsigned int(32) timescale;  
 if (version==0 || new\_version) {  
 unsigned int(32) earliest\_presentation\_time;  
 unsigned int(32) first\_offset;  
 } else { //version =1 || new\_version  
 unsigned int(64) earliest\_presentation\_time;  
 unsigned int(64) first\_offset;  
 }  
 unsigned int(16) reserved = 0;  
 unsigned int(16) reference\_count;  
 if (new version)  
 unsigned int(16) subpart\_count;  
 for(i=1; i <= reference\_count; i++) {  
 bit (1) reference\_type;   
 unsigned int(31) referenced\_size;  
 if (new\_version) {  
 for (j=1; j<=subpart\_count;j++) {  
 unsigned int(32) data\_reference\_offset;   
 // may be controlled by a flags value   
 unsigned int(32) referenced\_data\_size;   
 }  
 }  
 unsigned int(32) subsegment\_duration;   
 bit(1) starts\_with\_SAP;  
 unsigned int(3) SAP\_type;  
 unsigned int(28) SAP\_delta\_time;  
 }  
}

#### Semantics (for the new fields, the other remaining unchanged)

Data\_reference\_offset indicates in bytes from where, in a file or in a segment file, the indexed data start. The offset corresponds to the first byte of the file or to the first byte of the considered segment file.

referenced\_data\_size indicates a size in bytes for a contiguous byte range in the data part of the referenced fragment

subpart\_count indicates the number of data blocks (for example tiles) for the current subsegment.

This extended ‘*sidx*’ box can also be combined with ‘*sidx*’ boxes of the current version, for example as in the hierarchical or daisy-chain schemes defined in ISO/IEC 14496-12

## Attributes in DASH MPD

The following is proposed to be added into the SegmentBase element (clause 5.3.9.2.2 of DASH).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | @mediaOnlyRange | O | specifies the byte range that consists only of the media data, such as MediaDataBox, applicable to all Media Segments of the Representation. When used with ISOBMFF Media Segments, the indicated byte range shall start with a box.  The byte range shall be expressed and formatted as a byte-range-spec as defined in RFC 7233, Clause 2.1.. It is restricted to a single expression identifying a contiguous range of bytes. |

The following is proposed to be added into the SegmentList.SegmentURL element (clause 5.3.9.3.2 of DASH).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | @mediaOnlyRange | O | specifies the byte range within the resource identified by the @media that consists only of the media data, such as MediaDataBox. When used with ISOBMFF Media Segments, the indicated byte range shall start with a box.  The byte range shall be expressed and formatted as a byte-range-spec as defined in RFC 7233, Clause 2.1.. It is restricted to a single expression identifying a contiguous range of bytes. |

The following is proposed to be added into the SegmentTemplate element (clause 5.4.9.4.2 of DASH).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | @mediaOnly | O | specifies the template to create the Media Segment List where Media Segments only consist of the media data, such as MediaDataBoxes. |

## Example usage of the segment index box

*Add the following clause J.2.5:*

**J.2.5 Simple one-level indexing of "tightly packed" media**

When the flags field of the SegmentIndexBox is set so that (flags & 1) is equal to 1, the media data is "tigthly packed", i.e. a single MediaDataBox or IdentifiedMediaDataBox contain samples for a single track only in decoding order without unused bytes, sample auxiliary information, metadata, or any other information that does not belong to the sample format. A legacy client simply omits the flags field and the SegmentIndexBox syntax conditional on the values of the flags field. A client taking advantage of the "tightly packed" media could operate as follows:

* Conclude that since the SegmentIndexBox has (flags & 1) equal to 1, the reception of MovieFragmentBoxes is not necessary.
* Omit the downloading of MovieFragmentBoxes and only download the media data by deriving a byte range from the media\_data\_offset given in the SegmentIndexBox. Thus, a bitrate saving in the transmitted data is achieved.
* Create the MovieFragmentBoxes in the client side either by parsing the high-level syntax of the received media data or, when (flags & 2) and (flags & 4) are set, use the default values given in of TrackExtendsBox. For example, in case of AVC or HEVC, the information in the TrackRunBox could be concluded as follows:
  + Deriving the sample sizes and the sample size by carrying out the access unit boundary determination as specified in AVC or HEVC.
  + Deriving sample composition times from picture timing SEI messages present in the bitstream or concluding that composition times are proportional to picture order count.
  + Deriving sample flags from the VCL NAL unit types or setting sample flags to indicate an unknown status.

Figure J.2 shows an example that is aligned with the structure presented in Figure J.1. All entries of the top level SegmentIndexBox point to segments comprising one or more movie fragments, i.e. reference\_type is equal to 0. The values of referenced\_size and subsegment\_duration of each entry are calculated as in Table J.1 above. The dashed double-ended arrows in Figure J.2 indicate the values of media\_data\_offset.



Figure J.2. Simple segment index including media\_data\_offset values (dashed arrows).

# Sample reordering in Track Runs

## Introduction

Samples in TRUN are contiguous in byte range and stored in decoding order. As discussed previously, we identified use cases where changing the sample ordering could be beneficial:

- Usage of partially received segments (HTTP streaming, ISOBMFF-based broadcast)

- More efficient unequal FEC protection of the segment, with moof and base sublayer in a single, more protected zone

In these use cases, the movie fragment is incomplete and some samples are not available. With the current design, it is very likely that samples from a temporal sublayer are received while depended-on samples of lower layers are not. This implies that additionally to the lost samples, we end up with unusable yet completely received samples. By ordering samples according to their temporal sublayers, the usage of partially received segments can be improved.

As discussed in m44768, there are several options to do this:

- Option1: split the temporal layers as one per track

This is straightforward and works for codecs having a multi-track ISOBMFF encapsulation defined (yet not all of them do). However, this is quite costly in terms of ISOBMFF structures, each sublayer requiring a dedicated track hence duplication of traf, tfhd and trun boxes for each track. The complexity is also high since samples have to be properly re-interleaved when "merging" these tracks. This also requires multi-track segments which is not very friendly in HAS delivery or ISOBMFF-based broadcast. It finally requires splitting sample grouping and CENC information per track, which is both complex and costly (additional sample to group boxes, subsamples boxes, sample group descriptions if not in moov, saio/saiz/senc boxes etc ...)

- Option 2: split the temporal layers as one per TRUN

Since samples have to be in decoding order within the fragment, the only possibility to store data per temporal sublayer is to have a new trun whenever a we have a change of temporal sublayer between non contiguous samples:

*Example:*

For a classical I0P0B1B2B3 layout at 25 fps, with BN having no dependencies on BN+1 a potential 1s GOP structure could be:

I10P250 B51 B32 B23 B43 B91 B72 B63 B83B131 B112 B103 B123 B171 B152 B143 B163 B211 B192 B183 B203 B232 B223 B243

Our expected layout would be:

I10P250 B51 B91 B131 B171 B211 B32 B72 B112 B152 B192 B232 B23 B43 B63 B83B103 B123 B143 B163 B183 B203 B223 B243

To achieve this with the current ‘trun’ design, we would need:

*TRUN* I10P250 B51 *TRUN* B32 *TRUN* B23 B43 *TRUN* B91 *TRUN* B72 *TRUN* B63 B83*TRUN* B131 *TRUN* B112 *TRUN* B103 B123 *TRUN* B171 *TRUN* B152 *TRUN* B143 B163 *TRUN* B211 *TRUN* B192 *TRUN* B183 B203 *TRUN* B232 *TRUN* B223 B243

Hence 17 trun instead of 1! With a base TRUN size of 20 (12 for full box + 8 for sample count+data offset) or 16 for ctrn (12 for full box + 8 for sample count+data offset assuming offset can be less than 65k), we end up with at least 320 bytes (trun) or 256 bytes (ctrn) of overhead.

This gets even worse if we start increasing the GOP size or the frame rate.

The proposal in m44768 to overcome this was to use a single trun with a sample layout index allowing custom sample layouts (more details below).

## Further discussion

### Context

We investigated how to reuse an existing ISOBMFF HAS packaging (single file or segmented) to provide a low frame rate version of the content without duplicating the files. Our ultimate goal is to have as few byte range requests to issue as possible for a given media segment, to reduce complexity.

The level assignment box ‘leva’ seemed to be designed for this, as illustrated in DASH 4th edition (Figure 6):

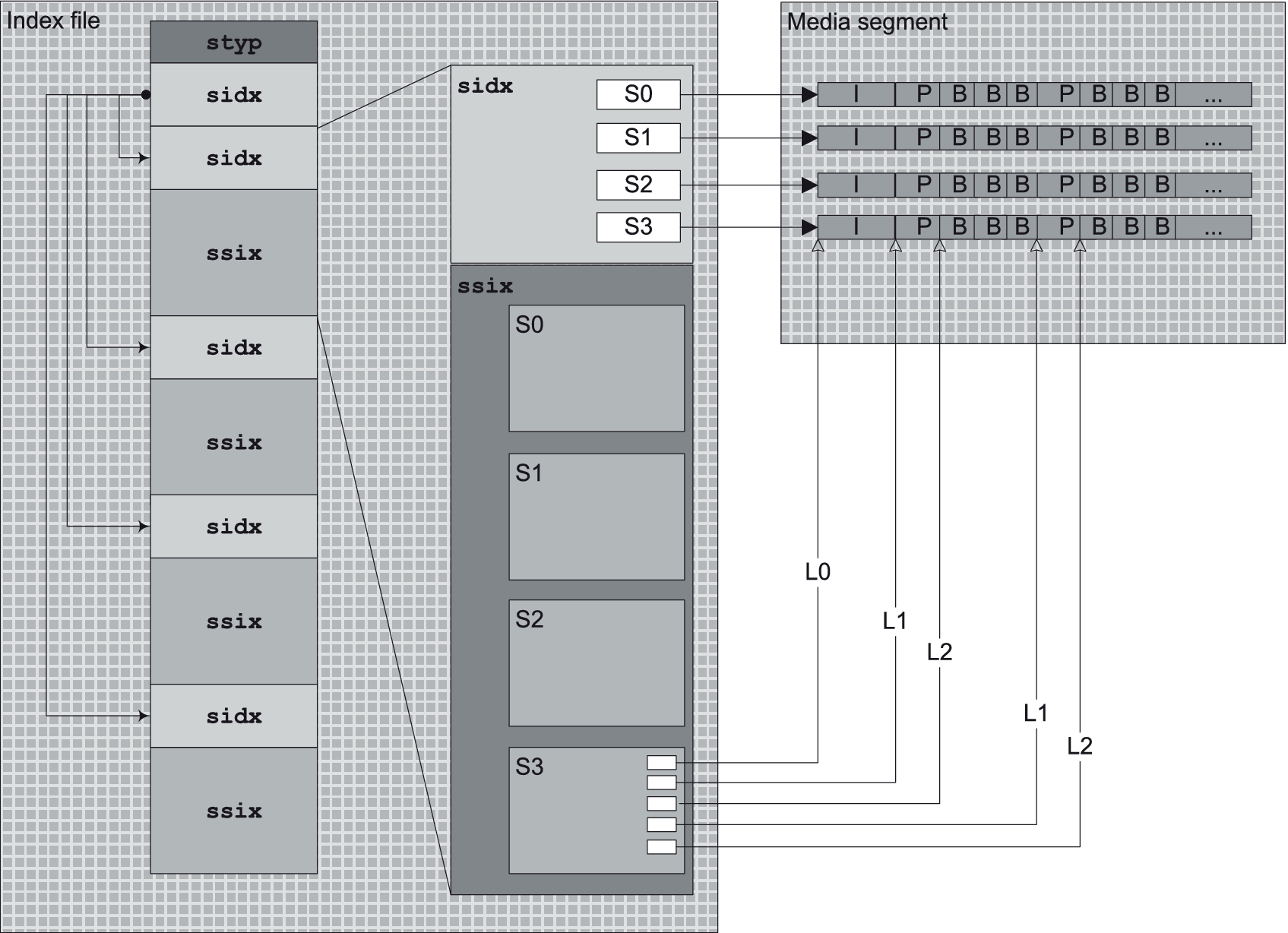


Figure : Mapping temporal sub-layers to Sub-Segment Index Box for trick mode in DASH

This box describes the assignment of one level per temporal sublayer and each level is further described in terms of byte range in an ‘ssix’ box.

### leva and ssix clarifications

See the Defect Report for 14496-12.

### Sample reordering using leva and ssix

Assuming our previous interpretation is correct, if we want to have samples organized per temporal dependencies, we have two possibilities:

1. Separate each sublayer in a dedicated traf, which we would want to avoid as we explained in section 2.
2. When a single track is used, the common (if not only) usage is to map the moof and first IDR (possibly the immediately following P frame if no leading pictures) in the first level, and the rest to a second level. Trying to map all P frames in the segment to the same level will not be possible (disjoint byte ranges), unless using the above trick (section 2) of splitting the ‘trun’ into multiple ‘trun’ to reorder the samples, with the size increase it induces.

One possibility would be to relax the leva/ssix constraint on byte range continuity per level, and allow multiple occurrences of a level in an ‘ssix’ box (see discussion in previous section). While this works, this has the following drawbacks:

- the ssix box becomes quite big: for our previous example, 17 entries instead of 4 (one per sublayer), each entry counting for 32 bits

- it seems to break the philosophy of ssix

- multiple byte ranges will be required for a given level

- samples are still in decoding order in the ‘mdat’ (not compatible with the identified use cases above)

We therefore propose to introduce a sample ordering index at the ‘trun’ level to enable:

- Single trun, single track

- Single byte-range request for a given sublayer (or level)

- Unmodified 'ssix' / 'leva', except specification clarifications as discussed in previous section

- Reusability of partially downloaded segments

- Friendliness for unequal FEC protection schemes

## Proposal

The proposed syntax below could be added either directly in the 'trun' or 'ctrn', or as a companion box (with mandatory processing) of the 'trun'. We propose here our preferred solution which is an extension of 'ctrn'.

### Sample interleave index in (Compact) Trun

The initial proposal in m44768 proposed to use the trun or compact trun currently under investigation to provide sample interleaving (or reordering) information. The principles of trun are kept (still describes contiguous set of samples in decoding order in file), only the locations of the samples in the mdat within the run change. This make sure that one reader fetching one trun has still all the data for these samples.

As noted in m44768, using a data offset per sample to provide the sample\_interleave\_index would not be very efficient since we already have the ‘trun’ base offset and each sample size. We therefore need to indicate the sample\_interleave\_index in the track run in a compact way.

The initial proposal from m44768 cost one index per sample, the sample\_interleave\_index using the same coding trick (1, 2 or 4 bytes) as the other fields in ‘ctrn’. We propose to deduce the number of bits to use for the sample\_interleave\_index from the one for sample count, since the sample\_interleave\_index shall be given for each sample.

We then need:

* 1 bit flags to indicate presence/absence of reordering/interleaving index
* 1 bit flags for the sample\_count\_index\_size

We propose to add the following flags value in ‘ctrn’ flags (see m50571 on ctrn tests):

0xTO\_BE\_DEFINED sample\_interleave\_bit: when set, indicates the samples in the trun may be in an order different from the decoding order, and that a map of sample index in decoding order will be given at the end of the trun.

Add in ctrn syntax section:

unsigned int(8) function indexToBitSize(sample\_count) {  
 if (sample\_count<256) return 8;  
 else return 16;  
}

Add at the end of ctrn box syntax:

if (tr\_flags & sample\_interleave\_bit) {  
 unsigned int(indexToBitSize(sample\_count))   
 sample\_interleave\_index [ sample\_count];  
}

Add to semantics:

sample\_interleave\_index: indicates the order of sample interleaving in the trun. A value of 0 indicates that the sample data start at the trun data offset. A value of K>0 indicates that the sample data start at the trun data offset plus the sum of the size of all samples with an interleaving index strictly less than K. The index shall range between 0 and sample\_count-1 inclusive. There shall not be two samples with the same interleaving index in the same trun.

We also propose to clarify the content of a compact track run in its definition section:

“A track run documents a contiguous set of samples in decoding order for a track. However, the actual storage of samples in mdat within the track run may be interleaved according to an optional index sample\_interleave\_index”

### Sample count in Compact Trun

We note that sample\_count is hardcoded to 16 bits in ctrn, but it is quite common for fragments to have less than 256 samples. Was this made on purpose?

We suggest using one remaining flag in ctrn to signal the size of the sample\_count field (see m50571 on ctrn tests results), as follows:

In Section 8.8.8.2.2 of AMD4, add the following:

0xTO\_BE\_DEFINED sample\_count16bits that when set indicates that sample\_count is coded on 16 bits. When not set, sample\_count is coded on 8 bits.

and in Section 8.8.8.2.2 of AMD4, replace in the syntax for CompactTrackRunBox:

unsigned int(16) sample\_count;

with

if (tr\_flags & sample\_count16bits)  
 unsigned int(16) sample\_count;  
 else  
 unsigned int(8) sample\_count;

## Conclusion

We recommend experts to include these improvements to track run description into the on-going AMD4 on Part-12.

# Extending Segment Index

## Introduction

Following discussions on ‚ssix‘, this proposal suggests extending the ‚sidx‘ to carry additional information. It basically proposes to add the sap size to the sidx, to enable efficient download of SAP when using MPEG DASH.

## Motivation

There are various ways to provide fast forward and fast rewind (a.k.a. trick modes, video scrubbing…) using http streaming. Some approaches may use a dedicated I-frame/low frame rate stream, while some other may use jumps between keyframes in a regular stream.

Using a dedicated stream increases the complexity of content generation and delivery, and it may be beneficial to provide trick modes based on „regular“ streams.

Assuming each segment begins by an I-frame, some implementations use „download and cancel“ approach: The download of the segment is started, then cancelled once the I-frame has been successfully downloaded and displayed. In this case, the ‚sidx‘ may be sufficient. However, using http 1.1, it requires closing the http session after each I-frame, which is not efficient.

‚ssix‘ may provide more precise indexing information (byte-ranges to portions of sub-segments) so that only the required amount of data is downloaded. It can be done by describing stream access points (sap) or temporal levels associated to stream samples, using sample groups. This avoids closing the http session, as precise byte-ranges are provided.

However, it comes with a significant increase in complexity, regarding content generation and player processing, because ‚ssix‘ requires many additional boxes to be present:

* ‚ssix‘, providing a mapping between levels and byte-ranges for a list of sub-segments
* ‚leva‘, providing a mapping between levels and the associated meaning. In our example, this meaning would typically be provided by a sample group, providing the sap type or the temporal level for each sample
* sample group description ‚sgpd‘ and sample to group box ‚sbgp‘ describing the criterion used for sample grouping and the group that each sample belongs to.

Further, when using MPEG DASH, the sample group information may be missing once the segment index has been downloaded, as it is partially or totally contained in the movie fragment header. This could thus require an additional download per subsegment to fill the missing information.

## Proposal

We would like a simple way to get the size of the SAP starting each subsegment.

We propose to extend the segment index information, to provide a range of bytes sufficient to download the SAP. This would allow downloading just the right amount of bytes to get the entire SAP, while keeping the http session open, and would enable simple seek/trick modes support.

aligned(8) class SegmentIndexBox extends FullBox('sidx', version, 0) {  
 unsigned int(32) reference\_ID;  
 unsigned int(32) timescale;  
 if (version==0) {  
 unsigned int(32) earliest\_presentation\_time;  
 unsigned int(32) first\_offset;  
 }  
 else {  
 unsigned int(64) earliest\_presentation\_time;  
 unsigned int(64) first\_offset;  
 }  
 unsigned int(16) reserved = 0;  
 unsigned int(16) reference\_count;  
 for(i=1; i <= reference\_count; i++)  
 {  
 bit (1) reference\_type;  
 unsigned int(31) referenced\_size;  
 unsigned int(32) subsegment\_duration;  
 bit(1) starts\_with\_SAP;  
 unsigned int(3) SAP\_type;  
 unsigned int(28) SAP\_delta\_time;  
 if (version>=2) {   
 if (starts\_with\_SAP) {  
 unsigned int(32) SAP\_range;  
 }  
 }  
 }  
}

SAP\_range provides a range of bytes, starting from the beginning of the subsegment sufficient to download the SAP (if any) associated to this subsegment.

*Note: could be made more constrained/explicit using SAP\_size instead of SAP\_range*

## Conclusion

We propose to consider the modification of the ‚sidx‘ syntax, to carry additional useful information, which may go beyond the modification proposed here.

While being flexible, the ‚ssix‘ design comes with a significant added complexty and is quite confusing. We believe the ‚sidx‘ could and should be sufficient to accomodate simple use cases.

## Notes from the minutes (Brussels)

Putting this inside the loop breaks backward compatibility; perhaps it would be safer in a separate loop? Would a flag be better than a version (unclear)? And should it be conditional on the reference\_type as well as starts\_with\_SAP and the feature being present? Do we need specific provision for "I don't know" (0?)?

Putting it all together, something like this

if (flags & mask) {  
 for(i=1; i <= reference\_count; i++)  
 {  
 if ((reference\_type[i] == 0) && (starts\_with\_SAP[i] == 1))   
 {   
 unsigned int(32) SAP\_end\_offset;  
 }  
 }  
 }

But can we achieve a similar effect by somehow telling the reader that level 0 in the ssix is, in fact, the SAP? Then no reading of the level assignment box, or sample groups, is needed. Also one could include two byte ranges for level 0 (the initial SAP, and enough bytes to include the second SAP).

We could do this with a version change to the ssix ("for version 1, the level assignment box is optional, and whether present or not, level 0 means the SAPs").

# Segment Index and Level Assignment

This is a follow-up on m50898 and on the discussions on the reflector regarding leva/ssix interpretation.

See also the Defect Report and issue *https://github.com/MPEGGroup/FileFormat/issues/12*

## Context

As identified in [m50898](http://wg11.sc29.org/doc_end_user/documents/128_Geneva/wg11/m50898-v1-m50898.zip) (also attached), section 3, we believe there is currently a contradiction between DASH and ISOBMFF interpretations of ssix and leva boxes.

* ISOBMFF seems to forbid multiple byte ranges in ssix box for a given level, due to constraints on level appearance order in a fraction (see ISOBMFF 8.8.13.1 and 8.16.4.1)
* DASH seems to expects/allow multiple byte ranges for a given level, as indicated in DASH 4th edition Figure 6 (section 6.4.6.3)

After digging into the history of leva and ssix, it appears:

* leva/ssix was proposed in m19471 as a single byte range into ISOBMFF for subsegments, and integrated as WD of AMD in w11726
* concerns were raised on usage of SSIX in MPEG-2 TS cases, where sample reordering is not possible (see m19975, m20087), and a new design on multiple byte range was added in ssix in m20218. The semantics for leva and ssix with ISOBMFF were not modified by the new design.

We therefore have two different design for ssix:

* when used in ISOBMFF, "leva" must be present and all bytes of a given level shall be contiguous
* when used outside ISOBMFF, level assignments is not in scope and multiple byte ranges for a given level are possible (depending on the segment format specification).

The constraints on ISOBMFF typically invalidate MPEG-DASH 4th edition Figure 6 (section 6.4.6.3), but only for ISOBMFF segments.

We therefore have two possibilities:

* do not change the semantics of leva and clarify ssix
* change semantics of leva and clarify ssix accordingly.

We note that the second approach might require some signaling (brand or "leva" version) to indicate that byte range for a fraction are no longer contiguous.

We don't have a strong opinion on either approaches and let the group decide which approach should be kept.

*From the minutes (Brussels): We prefer to change the versions of leva and ssix to relax the constraints.*

## Proposal

### SSIX-only clarifications

In 8.16.4.1 (ssix definition), replace the first 2 bullet points with (additions in yellow highlight, removals in red strike-through):

"

* Each level shall be assigned to exactly one partial subsegment, i.e. byte ranges for one level shall be contiguous and there shall not be two range entries for the same level.
* Levels of partial subsegments shall be assigned by increasing numbers within a subsegment, as mandated by the level assignment box~~. i.e., samples of a partial subsegment may depend on any samples of preceding partial subsegments in the same subsegment, but not the other way around. For example, each partial subsegment contains samples having an identical temporal level and partial subsegments appear in increasing temporal level order within the subsegment.~~

"

### leva and SSIX clarifications

If mutliple byte range per level is desired, we suggest supporting it through a version of leva:

In 8.8.13.2, replace

aligned(8) class LevelAssignmentBox extends FullBox('leva', 0, 0) {

with

aligned(8) class LevelAssignmentBox extends FullBox('leva', version, 0) {

In 8.8.13.1, replace

"Within a fraction, data for each level shall appear contiguously. Data for levels within a fraction shall appear in increasing order of level value. All data in a fraction shall be assigned to levels. "

with

"All data in a fraction shall be assigned to levels.

* For a level assignment box using version 0, data for each level shall appear contiguously within a fraction. Data for levels within a fraction shall appear in increasing order of level value.
* For a level assignment box using version 1, data for each level may appear non-contiguously within a fraction, and may appear in non-increasing order of level value.

"

In 8.16.4.1 (ssix definition), replace the first 2 bullet points with (additions in yellow highlight, removals in red strike-through):

"

* When version 0 of the level assignment box is used,
  + Each level shall be assigned to exactly one partial subsegment, i.e. byte ranges for one level shall be contiguous and there shall not be two range entries for the same level.
  + Levels of partial subsegments shall be assigned by increasing numbers within a subsegment, as mandated by the level assignment box~~. i.e., samples of a partial subsegment may depend on any samples of preceding partial subsegments in the same subsegment, but not the other way around. For example, each partial subsegment contains samples having an identical temporal level and partial subsegments appear in increasing temporal level order within the subsegment.~~
* When version 0 of the level assignment box is not used, byte ranges for a one level may be non-contiguous, resulting in two or more range entries with the same level.

"

# Haptics support

## Context

This section proposes adding a new media type for Timed Haptics to the ISO base media file format, and also proposes some boxes for 'common use' among haptics codecs.

The following section would be added as a new subsection of Chapter 12.

## Haptic Media

### Media Handler

Haptic media uses the 'hapt' handler type in the HandlerBox of the MediaBox, as defined in 8.4.3.

### Haptic Media Header

Haptics tracks use the NullMediaHeaderBox in the MediaInformationBox as defined in 8.4.5.

### Sample Entry

#### Definition

Haptic tracks use HapticSampleEntry.

The HapticSampleEntry extends the SampleEntry class and holds haptic configuration information. This configuration information may be general to more than one codec or be codec-specific defined as boxes in the codec-specific extensions to HapticSampleEntry.

The arrays of VibeGeneralConfigBox and KinestheticGeneralConfigBox boxes within the sample entry indicate whether the content in this track requires vibrator actuator(s), kinesthetic actuator(s), or both.

This provides information in the sample entry on the types of actuators the content in this track is designed to be played on. For example, if the content is intended for single vibrator actuator devices (e.g., basic phones, laptops, tablets), then the sample entry will have a single VibeGeneralConfigBox (array of one). If the track is intended for multiple vibrator actuators, the sample entry will have an array of VibeGeneralConfigBoxes, each of different frequency and motor type. A similar structure can be used for kinesthetic actuators.

NOTE it is possible to create a haptic track that can be played on both vibrator and kinesthetic actuators. The sample entry for such a track will have non-zero arrays of both VibeGeneralConfigBox and KinestheticGeneralConfigBox boxes.

#### Syntax

aligned(8) class HapticSampleEntry(codingname)extends SampleEntry(codingname) {  
 VibeGeneralConfigBox[] vibes;   
 // array of vibrator actuator configurations  
 KinestheticGeneralConfigBox[] kines;   
 // array of kinesthetic actuator configurations  
 Box()[] otherboxes;  
}

### Vibrator General Configuration

#### Definition

VibeGeneralConfigBox is a general-purpose configuration box in HapticSampleEntry for specific vibrator-based haptics systems. If the content is designed to be (ideally) played on a device with one or more vibrator actuators, there would be an array of one or more VibeGeneralConfigBoxes in the HapticSampleEntry.

#### Syntax

aligned(8) class VibeGeneralConfigBox extends FullBox(‘vibh’) {   
 template unsigned int actuatorID(4);   
 // arbitrary value uniquely identifying this actuator configuration  
 template unsigned int actuatorFrequency(16);   
 template unsigned int actuatorConfiguration(4);  
 template unsigned int endpointConfiguration(4);  
 unsigned int(4) reserved = 0; // to pad to 8 bits  
}

#### Semantics

actuatorID is an arbitrary value that identifies this actuator and must be unique among the VibeGeneralConfigBoxes in a given sample entry; there are two reserved values, 0 and 0xF, which shall not be used.

actuatorFrequency is a 16-bit value representing the actual frequency of the actuator used. Currently known actuators typically use 200 Hz, 1000 Hz, or 8000 Hz. Newer actuators could use other frequencies.

endpointConfiguration is a 4-bit value representing the waveform data format to be used by the codec to interpret the haptic media.

The actuatorConfiguration and endpointConfiguration are described below:

Table 1: Actuator Configuration

|  |  |
| --- | --- |
| **Actuator Configuration Index** | **Description** |
| 0x0 | Eccentric Rotating Mass (ERM) |
| 0x1 | Narrowband Linear Resonant Actuator (LRA) |
| 0x2 | High-definition (HD) Motor |
| 0x3 – 0xF | Reserved |

Table 2: Endpoint Configuration

|  |  |
| --- | --- |
| **Endpoint Configuration** | **Description** |
| 0x0 | Amplitude Modulation |
| 0x1 | Unipolar Waveform |
| 0x2 | Bipolar Waveform |
| 0x3-0xF | Reserved |

### Kinesthetic General Configuration

#### Definition

KinestheticGeneralConfigBox is a general-purpose configuration box in HapticSampleEntry for specific kinesthetic haptics systems. If the content is designed to be (ideally) played on a device with one or more kinesthetic actuators, there would be an array of one or more KinestheticGeneralConfigBoxes in the HapticSampleEntry.

#### Syntax

aligned(8) class KinestheticGeneralConfigBox extends FullBox(‘kinh’) {  
 template unsigned int actuatorID(4);   
 // arbitrary value uniquely identifying this actuator configuration  
 template unsigned int controlFrequency(4);  
 template unsigned int endpointConfiguration(4);  
 unsigned int(4) reserved = 0; // to pad to 8 bits  
}

#### Semantics

actuatorID is an arbitrary value that identifies this actuator and must be unique among the KinestheticGeneralConfigBoxes in a given sample entry; there are two reserved values, 0 and 0xF, which shall not be used.

endpointConfiguration is a 4-bit value representing the waveform data format to be used by the codec to interpret the haptic media, as described in Table 2 above.

controlFrequency is a 4-bit field specifying the control frequency of the kinesthetic feedback.

### Codec-specific Sample Entry Extensions

Each haptic codec may extend HapticSampleEntry and contain codec-specific information.

Each codec-specific sample entry extension may also specify the maximum number of actuators it needs for rendering the haptics track, using VibeGeneralConfigBoxes or KinestheticGeneralConfigBoxes.

### Haptics Sample Format and Sync Samples

The haptics coding format defines the format of a haptics sample. It also defines whether the coding format is all sync-sample, and if not, defines what a sync sample is.