Text

Description automatically generatedISO/IEC JTC 1/SC 29/WG 03 N1197

**ISO/IEC JTC 1/SC 29/WG 03  
MPEG Systems   
Convenorship: KATS (Korea, Republic of)**

**Document type:** Output Document

**Title:** Technologies under Consideration for ISO/IEC 14496-12

**Status:** Approved

**Date of document:** 2024-07-17

**Source:** ISO/IEC JTC 1/SC 29/WG 03

**No. of pages:** 88 (with cover page)

**Email of Convenor:** young.L @ samsung . com

**Committee URL:** <https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg3>

**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 03 MPEG SYSTEMS**

**ISO/IEC JTC 1/SC 29/WG 03 N1197**

**May 2024, Rennes, FR**

|  |  |
| --- | --- |
| **Title** | **Technologies under Consideration for ISO/IEC 14496-12** |
| **Source** | **WG 03, MPEG Systems** |
| **Status** | **Approved** |
| **Serial Number** | **23807** |

**Abstract**

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

[1 Handling lost or corrupted samples using a sample group for corrupted samples 2](#_Toc171988951)

[2 Edit lists and movie fragments 4](#_Toc171988952)

[3 Multiplexed timed metadata tracks 5](#_Toc171988953)

[4 Embedded Metadata Signaling 10](#_Toc171988954)

[5 On MIME type parameters 11](#_Toc171988955)

[6 On relation of entity groups and sample groups 12](#_Toc171988956)

[7 Stronger defaulting in Track Runs 13](#_Toc171988957)

[8 Sample reordering in Track Runs 21](#_Toc171988958)

[9 Segment Index and Level Assignment 25](#_Toc171988959)

[Definition 34](#_Toc171988960)

[Syntax 34](#_Toc171988961)

[Semantics 35](#_Toc171988962)

[10 Generic sub-picture track grouping extensions 37](#_Toc171988963)

[11 Signaling of Multi-Layer Picture Compositing Information in the VVC File Format 40](#_Toc171988964)

[12 Integrating new codecs 46](#_Toc171988965)

[13 MovieFragmentHeaderBox update 48](#_Toc171988966)

[14 Sample Run Sample Group 49](#_Toc171988967)

[15 Improvements to movie fragments and dynamic tracks 52](#_Toc171988968)

[16 Support for Haptics 64](#_Toc171988969)

[17 Asynchronous media signalling for event-based haptics 65](#_Toc171988970)

[18 Use of Preselection for signaling Picture in Picture in ISOBMFF 66](#_Toc171988971)

[19 Tracks in Multiple files 71](#_Toc171988972)

[20 Extensions for sample auxiliary information 78](#_Toc171988973)

[21 On media rating 80](#_Toc171988974)

[22 Entity References 81](#_Toc171988975)

[23 Undetermined mdat size 83](#_Toc171988976)

[24 Condensed file type box 84](#_Toc171988977)

[25 MetaBox compatibility with QT 84](#_Toc171988978)

[26 Use of template, reserved and pre\_defined 85](#_Toc171988979)

[27 Update to the Segment Index Box 87](#_Toc171988980)

[28 Signaling poisoned data for AI training 89](#_Toc171988981)

[29 Empty Track References 90](#_Toc171988982)

[30 Configuration for sample auxiliary information 91](#_Toc171988983)

# 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 MetadataKeyBox (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.

# 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.

# 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 7.4.
  + 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 7.6.
* 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 7.5.
* 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 7.6 and 7.7.
  + 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 7.8.

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 7.3.2 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 "tightly 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 1: 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):

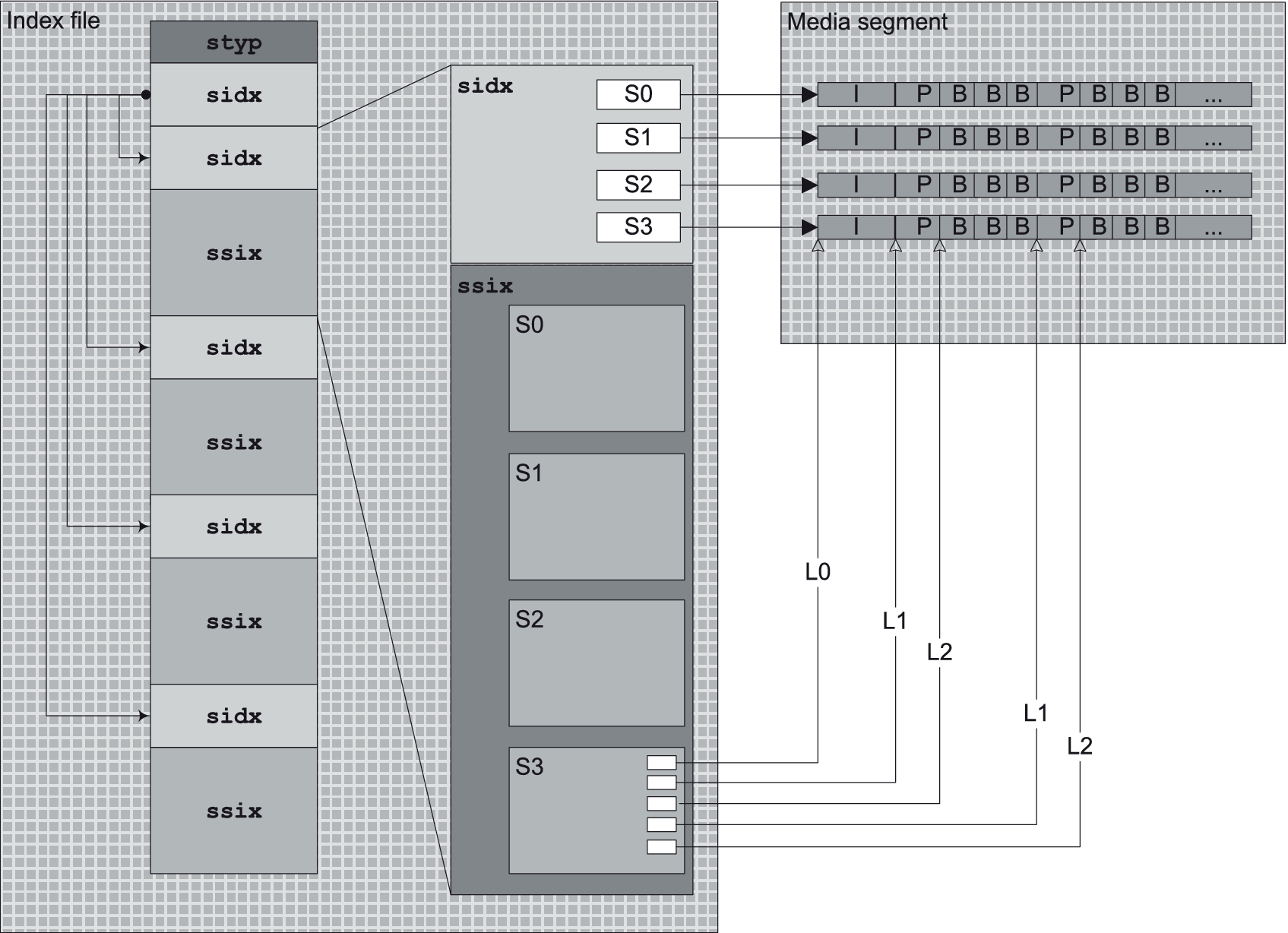


Figure 3: Mapping temporal sub-layers to Sub-Segment Index Box for trick mode in DASH

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;

# Segment Index and Level Assignment

Issues: [*http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/123*](http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/123) *and* [*http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/42*](http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/42)

See also [*https://github.com/MPEGGroup/FileFormat/issues/12*](https://github.com/MPEGGroup/FileFormat/issues/12)

## Discussion

*https://github.com/MPEGGroup/FileFormat/issues/12*

The proposal is based on the following observations:

1. signaling of IDR byte-range is very similar to the range concepts of ‘ssix’
2. avoid modifying ‘sidx’ box, both for backward compatibility issues (comment from TuC, section 15) and because we index subsegments
3. signaling multiple byte-ranges for the same level in ‘ssix’ seems reasonable (for example, two IDRs in a subsegment)
4. usage of ‘ssix’ with ‘leva’ is not always desirable, especially since:

* level assignment may be dependent on sample group description and sample to group mapping, which is not always available (‘moof’ not yet fetched).
* ‘leva’ requires level to be present in increasing order in the ‘mdat’, which does not allow multiple byte-ranges for a given level.
* ‘leva’ cannot be updated on the fly (present in 'mvex'), all levels to describe have to be known at the start of the session

We therefore would like to introduce a new design of ‘ssix’, fixing the above shortcomings. Moreover, while redesigning ‘ssix’, we also considered the use case of spatial indexing for tile tracks in a file.

## Proposal

The proposal defines a way to use multiple byte ranges per level in ‘ssix’/’leva’, and multiple byte ranges with ‘ssix’ without ‘leva’ through predefined level assignments.

Yellow-highlighted corresponds to text (Part-12) move

Green comes from above proposal

Blue are changes as proposed in TuC.

*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*

"When version 0 of the LevelAssignmentBox is used, within a fraction, data for each level shall appear contiguously, and data for levels shall appear in increasing order of level value. All data in a fraction shall be assigned to levels.

When version 1 or more of the LevelAssignmentBox is used, data for each level need not be stored contiguously and data for levels may be stored in random order of level value. Some data in a fraction may have no level assigned, in which case the level is unknow but is not a level from the levels defined by the LevelAssignmentBox."

*In 8.8.13.1 remove*

“

When padding\_flag is equal to 1 this indicates that a conforming fraction can be formed by concatenating any positive integer number of levels within a fraction and padding the last MediaDataBox by zero bytes up to the full size that is indicated in the header of the last MediaDataBox. The use of padding\_flag is deprecated.

”

*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 8.8.13.3 replace*

“padding\_flag equal to 1 indicates that a conforming fraction can be formed by concatenating any positive integer number of levels within a fraction and padding the last MediaDataBox by zero bytes up to the full size that is indicated in the header of the last MediaDataBox. When

padding\_flag is equal to 0 this is not assured.

”

with

“

padding\_flag deprecated, should be set to 0.

”

*Replace 8.16.4.1 with*

“

The SubsegmentIndexBox provides a mapping from levels (as specified by the LevelAssignmentBox or as indicated in the box itself) to byte ranges of the indexed subsegment. In other words, this box provides a compact index for how the data in a subsegment is ordered according to levels into partial subsegments. It enables a client to easily access data for partial subsegments by downloading ranges of data in the subsegment.

~~Each byte in the subsegment shall be explicitly assigned to a level, and hence the range count shall be 2 or greater.~~ ~~If the range is not associated with any information in the level assignment, then any level that is not included in the level assignment may be used.~~

There shall be 0 or 1 SubsegmentIndexBoxes per each SegmentIndexBox that indexes only leaf subsegments, i.e. that only indexes subsegments but no segment indexes. A SubsegmentIndexBox, if any, shall be the next box after the associated SegmentIndexBox. A SubsegmentIndexBox documents the subsegments that are indicated in the immediately preceding SegmentIndexBox.

In general, the media data constructed from the byte ranges is incomplete, i.e. it does not conform to the media format of the entire subsegment.

For leaf subsegments based on this document (i.e. based on movie sample tables and movie fragments):

* ~~Each level shall be assigned to exactly one partial subsegment, i.e. byte ranges for one level shall be contiguous.~~
* ~~Levels of partial subsegments shall be assigned by increasing numbers within a subsegment, 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.~~
* For version 0 of the box, each level shall be assigned to exactly one partial subsegment and in increasing order of level value, i.e. byte ranges for one level shall be contiguous and samples of a partial subsegment may depend on any samples of preceding partial subsegments in the same subsegment, but not the other way around. This implies that all data for a given level require a single range to be retrieved.
* For version 1 or more of the box, multiple byte ranges, possibly discontinuous, with the same level may be described. This implies that all data for a given level may require multiple byte ranges to be retrieved.

//editor's note: the next notes correspond to the previously existing last 2 bullets of the spec but they are informative or repetitions from leva.

Note 1: When a partial subsegment is accessed in this way, for any assignment\_type other than 3, the final MediaDataBox may be incomplete, that is, less data is accessed than the length indication of the MediaDataBox indicates is present. The length of the MediaDataBox may need adjusting, or padding used. ~~The padding\_flag in the LevelAssignmentBox indicates whether this missing data can be replaced by zeros. If not, the sample data for samples assigned to levels that are not accessed is not present, and care should be taken not to attempt to process such samples.~~

Note 2: The data ranges corresponding to partial subsegments include both MovieFragmentBoxes and MediaDataBoxes. The first partial subsegment, i.e. the lowest level, will correspond to a MovieFragmentBox as well as (parts of) MediaDataBox(es), whereas subsequent partial subsegments (higher levels) may correspond to (parts of) MediaDataBox(es) only.

For version 0 of this box, the presence of the LevelAssignmentBox in the movie is mandatory, and the LevelAssignmentBox shall have a version equal to 0.

*Editor's note:* the current v0 spec is unclear, it does not explicitly mandate leva with ssix, maybe we should keep this.

Note 3: assignment\_type equal to 0 (specified in the LevelAssignmentBox) can be used, for example, together with the temporal level sample grouping ('tele') when frames of a video bitstream are temporally ordered within subsegments; assignment\_type equal to 2 can be used, for example, when each view of a multiview video bitstream is contained in a separate track and the track fragments for all the views are contained in a single movie fragment. assignment\_type equal to 3 can be used, for example, when audio and video movie fragments (including the respective MediaDataBoxes) are interleaved. The first level can be specified to contain the audio movie fragments (including the respective MediaDataBoxes), whereas the second level can be specified to contain both audio and video movie fragments (including all MediaDataBoxes).

For version 1 of this box, the presence of the LevelAssignmentBox is only mandatory for level\_assignment\_type 0, in which case the LevelAssignmentBox shall have a version of 1.

“

*Replace 8.16.4.2 with*

1. aligned(8) class SubsegmentIndexBox extends FullBox('ssix', version, flags) {  
   if (version==0) {  
    unsigned int(32)subsegment\_count;  
    for( i=1; i <= subsegment\_count; i++) {  
    unsigned int(32)range\_count;  
    for ( j=1; j <= range\_count; j++) {  
    unsigned int(8) level;   
    unsigned int(24) range\_size;   
    }  
    }  
   } else {  
    unsigned int(16)subsegment\_count;  
    unsigned int(1)lsc;   
    unsigned int(1)incomplete;   
    unsigned int(2)lbs;   
    unsigned int(2)rbs;   
    unsigned int(2)reserved;   
    unsigned int(8)level\_assignment\_type;  
    for( i=1; i <= subsegment\_count; i++) {  
    unsigned int(lsc ? 32 : 16)range\_count;  
    for ( j=1; j <= range\_count; j++) {  
    unsigned int((lbs+1)\*8) level;   
    unsigned int((rbs+1)\*8) range\_size;   
    }  
    }  
   }  
   }

*Replace 8.16.4.3 with*

subsegment\_count is a positive integer specifying the number of subsegments for which partial subsegment information is specified in this box. subsegment\_count shall be equal to reference\_count (i.e., the number of movie fragment references) in the immediately preceding SegmentIndexBox.

lsc if set, indicates that the number of indexed ranges within a partial subsegment is coded on 32 bits, otherwise the number of indexed ranges within a partial subsegment is coded on 16 bits.

incomplete if set, indicates that the last range of a given subsegment may ~~not cover the entire~~ end before the last byte of the subsegment, in which case assignment of remaining bytes to level is unknown but the remaining bytes should not correspond to any level listed in the box.

lbs gives the number of bytes, minus 1, used in coding the level field.

rbs gives the number of bytes, minus 1, used in coding the range field.

level\_assignment\_type gives the associated semantics of the indicated level.

* 0: the level value corresponds to the level indicated in the leva box. If the range is not associated with any information in the level assignment, then any level that is not included in the level assignment may be used. This value shall only be used when the leva box version is 1 or more.
* 1: the level value corresponds to a dependency level.
  + Level 0 indicates that the byte range contains:
    - exactly one or more file-level boxes (e.g. MovieFragmentBox) other than a media data container box (e.g. MediaDataBox or IdentifiedMediaDataBox), and/or
    - zero or at most one box header (8 or 16 bytes) of a media data container box which shall correspond to the last 8 or 16 bytes of the byte range
  + Level 1 indicates same type of data as level 0 but having a dependency on the previous preceding byte range with level L0 (e.g. one single box header (8 or 16 bytes) of a media data container box, the media data container box containing data described by the preceding MovieFragmentBox)
  + Level 2 means the data is independently decodable (SAP 1, 2 or 3). Byte range with level 1 immediately preceding if specified and the first preceding byte range with level 0 are required to process the data.
  + Level N, with N>2, requires data from the preceding byte ranges with lower levels (level N-1 and below) to be processed, stopping at the previous preceding byte range with level 2 if specified, otherwise at the first byte range in the box. Each first byte range with level 0 or 1 preceding any required byte range (level 2 to N) is required to process the data.
* 2: the level value corresponds to a multitrack dependency level. In this mode, lbs shall be 1 or more (i.e., at least 16 bits to code the level). The first 8 bits of the level field give the dependency level value, with the same values and semantics as level\_assignment\_type 1. The remaining less significant bits of the level field give a trackID, which shall identify a track present in the indexed subsegment for level values other than 0 and 1, and shall be 0 if level value is 0 or 1. In this mode, each range with level N>1 consists only of data from the identified track~~, possibly with some meta-data boxes (movie fragments, etc...). The~~ and the level value only gives dependency information within the track. This allows cross-track indexation within a same level.
* 3->0xFF: ISO reserved

range\_count specifies the number of partial subsegment levels into which the media data is grouped. For version 0 of the box, this value shall be greater than or equal to 2 and each byte in the subsegment shall be explicitly assigned to a level. For version 1 or more of this box, this value may be 0 or more, and the described ranges may lead to a size smaller than the subsegment if and only if incomplete is set to 1.

range\_size indicates the size of the partial subsegment. This value shall not be 0, except for the last entry for which the value 0 may be used ~~in the last entry~~ to indicate the remaining bytes of the segment, to the end of the segment.

level specifies the level to which this partial subsegment is assigned.

For level\_assignment\_type 1 or above, byte ranges assigned to levels other than 0 or 1 shall not contain file-level box headers. Typically, the header of a media data container box (e.g. MediaDataBox or IdentifiedMediaDataBox) is in level 0 or 1 while data may be in level 2 or more.

NOTE For level\_assignment\_type 1, since level N depends only from level N-1 and below, a direct mapping of temporal sublayers to levels will not always be possible in case frames from one temporal sublayer depend on preceding frames from the same temporal sublayer in another byte range.

## Examples of use

### Single track indexing

A picture containing shape

Description automatically generated

Figure 4 - level\_assignment\_type=1 with 2 IDRs and no explicit range for moof

A picture containing shape

Description automatically generated

Figure 5 - level\_assignment\_type=1 with 2 IDRs and explicit range for moof

A picture containing text

Description automatically generated

Figure 6 - level\_assignment\_type=1 with low latency DASH segment, 2 chunks and no explicit range for 'moof

### Multi-track indexing

A picture containing text

Description automatically generated

Figure 7 - level\_assignment\_type=2, 2 moof+traf with 1 IDR each and no explicit range for moof

A picture containing shape, rectangle

Description automatically generated

Figure 8 - level\_assignment\_type=2, 1 moof with 2 trafs, 1 IDR/traf and no explicit range for moof

A picture containing rectangle

Description automatically generated

Figure 9 - level\_assignment\_type=2, 1 moof with 2 trafs, 1 IDR/traf and explicit explicit range for moof

## Alternative approach

Discussion <http://mpeg.expert/software/MPEG/Systems/FileFormat/isobmff/-/issues/218>

### Alternative approach

The SubsegmentIndexBox('ssix') provides a mapping from levels (as specified by the LevelAssignmentBox('leva')) to byte ranges of the indexed subsegment. In the current ISOBMFF specification **Error! Reference source not found.**, assignment\_type from the LevelAssignmentBox box indicates the mechanism used to specify the assignment to a level:

* 0 or 1: sample groups are used to specify levels (for value 1: a parameterized sample group (i.e. with grouping type parameter))
* 2 or 3: level assignment is by track
* 4: the respective level contains the samples for a sub-track

In order to interpret the signification of levels when sample groups are used to specify levels, a reader will have to process the MovieFragmentBox('moof') to know whether sample group descriptions are added/modified in the fragment, and to know which samples are associated to which sample group description entry. If sample group description entries are inserted in the fragment, ssix cannot map to such group description entry. Then, Levels for inserted sample group description entries cannot be described with ssix/leva..

The suggested approach in section 9.2 of the ISOBMFF TuC is the possibility to define level assignments within SubsegmentIndexBox('ssix') (with version > 0), so that no dependencies on a static LevelAssignmentBox('leva') are required. The proposed text is quite complete and keeps the same approach of level assignment type. However, if sample B depends on sample A and a file reader needs to know exact level dependencies (i.e. can B be decoded if A is not?), the samples will have to be in separate levels. This gives a potentially very complex hierarchy of samples (almost as many levels as there are samples in one GOP), and may not be sufficient for some use cases (c.f. below).

Additionally, when doing low-latency streaming, we want to push data as fast as possible without having to wait for the complete segment. However, using SubsegmentIndexBox('ssix') mandates SegmentIndexBox('sidx') (to get the number of entries), and the SegmentIndexBox('sidx') mandates a size and a duration per entry, which are not known until the end of the segment. In other words, SegmentIndexBox('sidx')/SubsegmentIndexBox('ssix') is useless in low-latency DASH/HLS.

Finally, when doing broadcast/multicast ABR, the player must decide whether a file requires repair or not. Having levels (using SubsegmentIndexBox('ssix')/LevelAssignmentBox('leva') supposing non low-latency) does help but is insufficient. Indeed, if one level is incomplete, the player still needs to understand the samples impacted by the losses to take a decision (repair loss or drop sample(s)). Some bytes missing in one level can invalidate:

* the entire byte range(s) for that level
* a subset of the samples in that level
* only the last (in decoding order) sample in that level

The repair decision will not be the same in these cases, but cannot be inferred from the SubsegmentIndexBox('ssix') description. For example, samples belonging to the same sample group description entry (hence level) may still be dependent on each other.

### proposal

Rather than establishing a (complex) hierarchy of byte ranges in SubsegmentIndexBox('ssix') which still requires the MovieFragmentBox('moof') to be downloaded to understand the meaning of the desired byte ranges (described by the mapping to sample group), we suggest properly defining sample dependencies within a track or track fragment.

The existing SampleDependencyTypeBox('sdtp') box (and equivalent fragment flags) gives a very high level on the structure of a stream, i.e. is the sample independent (IDR)?, is it depended on (IDR, P, lower Bs)? or are no other samples depending on this one (leaf B)?.

This allows for leaf samples to be simply discarded if needed. However, dependencies of intermediate samples once leaf samples are discarded is unknown:

S1d=0 S2d=1 S3d=1,2 S4d=2,3 S5d=3,4 S6d=1 S7d=1,6 S8d=6,7 S9d=7,8

*Note: SXd=y,z* reads as Sample X with dependency to samples Y and/or Z.

In this example, S5 and S9 are “leaf” samples (not depended on) and can be discarded.

If the player needs to drop other frames in the pattern, it cannot guess from the SampleDependencyTypeBox('sdtp') that S4 and S8 can be discarded as well, once S5 and S9 have been dropped.

Thus, having an explicit list of dependencies allows the file reader to know exactly which samples are impacted when not decoding one sample (by choice or due to losses in the transmission).

This has the benefit of being independent from the coding type and gives a complete description of sample dependencies.

It is important to note that dependency patterns depend on the encoder decisions (e.g. GOP structure), and can vary from GOP to GOP or can be a fixed subset. Fixed subset advocates for usage of sample groups, when dynamic variations would rather go for simple box.

Another benefit is that this proposal may be combined with ssix/leva for on-demand or non low-latency live cases, level assigned to the ‘'sdep' sample group.

### Possible syntax (updated from 65337)

Add a new section 10.13 Sample Dependency Sample Group

# Definition

Group Types: 'sdep'  
Container: Sample Group Description Box ('sgpd')  
Mandatory: No  
Quantity: Zero or more

This sample group provides explicit coding dependencies of samples towards other samples in the same track or in referenced tracks.

Dependencies are either described by a relative distance from the mapped sample (offset\_from\_reference=0) or from the last previous sample not mapped to the SampleDependencyGroupEntry(offset\_from\_reference=1).

The listed dependencies shall only contain the direct dependencies, i.e. if sample A depends on sample B which in turn depends on sample C, only sample B shall be listed as a dependency to sample A.

The version of the SampleGroupDescriptionBox for the 'sdep' sample group shall be greater than or equal to 1.

# Syntax

class SampleDependencyGroupEntry extends SampleGroupDescriptionEntry ('sdep')  
{  
 unsigned int(1) offset\_from\_reference;   
 unsigned int(1) has\_inter\_deps;  
 unsigned int(14) num\_dependencies; // intra-track

if (has\_inter\_deps)  
 unsigned int(16) num\_inter\_dependencies; // inter-track  
 else

num\_inter\_dependencies = 0;

if (num\_dependencies < 0x3FFF)

signed int(16) depended\_sample\_num\_diff[num\_dependencies];

for (i=0; i<num\_inter\_dependencies; i++){  
 unsigned int(16)track\_ref\_index;  
 signed int(16)depended\_inter\_sample\_num\_diff;  
 }  
}

# Semantics

In this subclause, the following terms are defined:

* A previous reference sample of track *A* refers to the previous sample in track *A* with no mapping to the SampleDependencyGroupEntry, meaning for example the previous IDR, BLA or CRA.
* A translated sample number is the sample number of the sample with the same decoding time (as the sample being described) in the referenced track if present, or one plus the sample number of the sample immediately preceding the decoding time.

offset\_from\_reference indicates the base sample (i.e. the sample used as reference for the sample offsets) from which is computed the depended\_sample\_num\_diff and depended\_inter\_sample\_num\_diff values. offset\_from\_reference=0 means values (for depended\_sample\_num\_diff and depended\_inter\_sample\_num\_diff) are relative to the sample number of the sample being described for dependencies in the track (also denoted mapped sample) or to the translated sample number in the referenced track for inter-track dependencies. offset\_from\_reference=1 means values (for depended\_sample\_num\_diff and depended\_inter\_sample\_num\_diff) are relative to the sample number of the previous reference sample in the track or relatively to the translated sample number in the referenced track for inter-tracks dependencies.

has\_inter\_deps indicates if the samples mapped to this entry depend from sample(s) of another track (or other tracks). Value 0 means dependency only from sample(s) of the same track. When has\_inter\_deps=0, num\_inter\_dependencies is inferred to be equal to 0. Value 1 means that samples mapped to this entry depends of sample(s) from another track (or other tracks).

num\_dependencies indicates the number of samples that the described sample depends on. Value 0 means no dependency to any other sample in the track. Value 0x3FFF (i.e. all bits set to 1) means dependencies are unknown, in which case has\_inter\_deps shall be 0. Samples that are not mapped to any sample group entry indicating sample dependencies are considered with no dependencies.

num\_inter\_dependencies indicates the number of samples that the described sample depends on in other tracks. Value 0 means no dependency to any other sample in other tracks.

depended\_sample\_num\_diff indicates the value used to locate a sample’s reference in the same track. If offset\_from\_reference=0, the value indicates the difference between the sample number of the sample being described and the sample depended on, and the value shall be strictly positive (i.e. a value of a value of 2 indicates that sample with number N depends on sample with number N-2). If offset\_from\_reference=1, the value is the difference between the previous reference sample *Pref* in the track of the sample being described and the sample number of the sample depended on, a negative value indicating a sample before *Pref*, a positive value indicating a sample after *Pref* and a value of 0 meaning *Pref*.

track\_ref\_index is the index in the track reference of type 'tdep' providing the track\_ID of the referenced track. Value 1 indicates the first entry. Value 0 is reserved.

depended\_inter\_sample\_num\_diff indicates value used to locate a sample’s reference in the referenced track. If offset\_from\_reference=0, the value indicates the difference between the translated sample number of the sample being mapped to this entry and the sample number of the sample depended on, and the value shall be strictly positive (i.e. a value of 2 indicates that sample with number N depends on sample with number N-2). If offset\_from\_reference=1, the value is the difference between the sample number of the previous reference samplet *PITref* in the referenced track and the sample number of the sample depended on, a negative value indicating a sample before *PITref*, a positive value indicating a sample after *PITref* and a value of 0 meaning *PITref*.

### Impact in TuC

#### Section 8 of TuC

We suggest to remove section 8 on Sample reordering in Track Runs of TuC.

There is no support for sample reordering and SubsegmentIndexBox('ssix')/ LevelAssignmentBox('leva') clarifications suggested in this section 8 are addressed by this contribution.

#### Section 9 of TuC:

We suggest to remove all text related to version 1 of SubsegmentIndexBox('ssix'), including figures.

However, we note that current specification does not mandate the presence of the LevelAssignmentBox('leva') when SubsegmentIndexBox('ssix') is used, and files already exist that don’t use the LevelAssignmentBox('leva') with SubsegmentIndexBox('ssix'). But in such case, the meaning of level values is unspecified.

We therefore suggest to make clear in the specification that the LevelAssignmentBox('leva') is optional, and when LevelAssignmentBox('leva') is absent, the following default level assignment is recommended:

* Level 0 indicates that the byte range contains exactly one or more file-level boxes (e.g. MovieFragmentBox) other than a media data container box (e.g. MediaDataBox or IdentifiedMediaDataBox),
* Level 1 indicates that the data is independently decodable (SAP 1, 2 or 3) and may start with a MovieFragmentBox, and only the first preceding byte range with level 0, if present, is required to process the data,
* Level N, with N > 1, indicates other data and requires data from the preceding byte ranges with lower levels (level N-1 and below) to be processed. The last occuring preceding byte range with level 0, if present, and the last occuring preceding byte range with level 1 are required to process a byte range with level N>1.

We also suggest to remove the padding\_flag that is useless in LevelAssignmentBox('leva') box.

We also suggest to specify that non-contiguous byte ranges for a same level may exist when LevelAssignmentBox('leva') is absent.

#### Section 7 of TuC

We also note that section 7 also deals with indexing for stronger defaulted fragments and we believe this has been addressed in a slightly different way in OMAF, maybe this should be removed as well.

## Questions/remarks

It is noted that the proposed 'sdep' sample group is very similar to 'refs' sample group in HEIF. It would be good to investigate the relationship between the two.

It would be interesting to check the behaviors of existing common players when files with ssix and no leva, or levels not matching leva.

# Generic sub-picture track grouping extensions

Issue:[*http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/53*](http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/53)

*And updated proposal in http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/152*

The purpose of this proposal is to allow describing 2D spatial relationship between multiple video bitstreams that relate to a same source content (characterized by a source\_id) (for instance, multiple videos representing subparts of a large panorama). This is currently defined in OMAF, but the concept is generic-enough to apply to ISOBMFF.

## Updated semantics of track\_group\_type

The semantics of track\_group\_type of the TrackReferenceBox is changed from

track\_group\_type indicates the grouping\_type and shall be set to one of the following values, or a value registered, or a value from a derived specification or registration:

'msrc' indicates that this track belongs to a multi-source presentation. Specified in 8.3.4.4.1.

'ster' indicates that this track is either the left or right view of a stereo pair suitable for playback on a stereoscopic display. Specified in 8.3.4.4.2.

The pair of track\_group\_id and track\_group\_type identifies a track group within the file. The tracks that contain a particular TrackGroupTypeBox having the same value of track\_group\_id and track\_group\_type belong to the same track group.

to

track\_group\_type indicates the grouping\_type and shall be set to one of the following values, or a value registered, or a value from a derived specification or registration:

'msrc' indicates that this track belongs to a multi-source presentation. Specified in clause 8.3.4.3 of ISO/IEC 14496-12.

'ster' indicates that this track is either the left or right view of a stereo pair suitable for playback on a stereoscopic display. Specified in clause 8.3.4.4.2 of ISO/IEC 14496-12.

'2dsr' indicates that this track belongs to a group of tracks with two dimensional spatial relationships (e.g. corresponding to spatial parts of a video source). Specified in clause (TBD) of this document.

The pair of track\_group\_id and track\_group\_type identifies a track group within the file. The tracks that contain a particular TrackGroupTypeBox having the same value of track\_group\_id and track\_group\_type belong to the same track group.

Add a new section in 12.1 Video media:

## 12.1.10 Two dimensional spatial relationships

### 12.1.10.1 Definition

A SpatialRelationship2DDescriptionBox TrackGroupTypeBox indicates that this track belongs to a group of tracks with 2D spatial relationships (e.g. corresponding to planar spatial parts of a video source). A SpatialRelationship2DDescriptionBox TrackGroupTypeBox with a given track\_group\_id implicitly defines a coordinate system with an arbitrary origin (0,0) and a maximum size defined by total\_width and total\_height; the x-axis is oriented from left to right and the y-axis from top to bottom. The tracks that have the same value of source\_id within a SpatialRelationship2DDescriptionBox TrackGroupTypeBox are mapped as being originated from the same source and their associated coordinate systems share the same origin (0,0) and the orientation of their axes. For example, a very high resolution video could have been split into sub-picture tracks. Each sub-picture track then conveys its position and sizes in the source video.

Tracks in the same track group shall declare the same source\_id, total\_width, and total\_height. Track groups with different track\_group\_id values and the same source\_id represent the same source content, possibly at different resolutions (i.e. with different values of total\_width or total\_height).

NOTE: A source can be represented by different such track groups (for instance when the same source is available at different resolutions). Each of these track groups is identified by its own identifier track\_group\_id. Since all of these track groups originate from the same source, they share the same source\_id.

There shall be one or more associated SpatialRelationship2DGroupEntry(s) in the associated track (this track possibly has a dynamic size and/or position). When every sample maps to the default indicated in the SampleGroupDescriptionBox in the MovieBox; that SampleGroupDescriptionBox can indicate a default sample group or indicate that all samples are unmapped, depending on its version by setting the static\_group\_description and static\_mapping flags.

### 12.1.10.2 Syntax

aligned(8) class SpatialRelationship2DSourceBox  
 extends FullBox('2dss', 0, 0) {  
 unsigned int(32) total\_width;  
 unsigned int(32) total\_height;  
 unsigned int(32) source\_id;  
}

aligned(8) class SpatialRelationship2DDescriptionBox extends TrackGroupTypeBox('2dsr') {  
 // track\_group\_id is inherited from TrackGroupTypeBox;  
 SpatialRelationship2DSourceBox(); // mandatory, must be first  
 // other optional boxes  
}

### 12.1.10.3 Semantics

total\_width specifies the maximum width in the coordinate system of the SpatialRelationship2DDescriptionBox track group. The value of total\_width shall be the same in all instances of SpatialRelationship2DDescriptionBox with the same value of track\_group\_id.

total\_height specifies the maximum height in the coordinate system of the SpatialRelationship2DDescriptionBox track group. The value of total\_height shall be the same in all instances of SpatialRelationship2DDescriptionBox with the same value of track\_group\_id.

source\_id parameter provides a unique identifier for the source. It implicitly defines a coordinate system associated to this source.

## 12.1.10.4 Spatial relationship 2D sample group

### Definition

The '2dsr' grouping\_type for sample grouping declares the positions and sizes of the samples from a sub-picture track in a spatial relationship track group. Version 1 of the SampleToGroupBox shall be used when grouping\_type is equal to '2dsr'. The value of grouping\_type\_parameter shall be equal to track\_group\_id of the corresponding spatial relationship track group.

### Syntax

class SpatialRelationship2DGroupEntry extends VisualSampleGroupEntry('2dsr') {  
 unsigned int(16) object\_x;  
 unsigned int(16) object\_y;  
 unsigned int(16) object\_width;  
 unsigned int(16) object\_height;   
 /\* Application specific extension here \*/  
 unsigned int(32) app\_specific\_parameters;  
}

### Semantics

object\_x specifies the horizontal position of the top-left corner of the samples in this group within the coordinate system specified by the corresponding spatial relationship track group. The position value is the value prior to applying the implicit resampling caused by the track width and height, if any, in the range of 0 to total\_width − 1, inclusive, where total\_width is included in the corresponding SpatialRelationship2DDescriptionBox.

object\_y specifies the vertical position of the top-left corner of the samples in this group within the coordinate system specified by the corresponding spatial relationship track group. The position value is the value prior to applying the implicit resampling caused by the track width and height, if any, in the range of 0 to total\_height − 1, inclusive, where total\_height is included in the corresponding SpatialRelationship2DDescriptionBox.

object\_width specifies the width of the samples in this group within the coordinate system specified by the corresponding spatial relationship track group. The width value is the value prior to applying the implicit resampling caused by the track width and height, if any, in the range of 1 to total\_width, inclusive.

object\_height specifies the height of the samples in this group within the coordinate system specified by the corresponding spatial relationship track group. The height value is the value prior to applying the implicit resampling caused by the track width and height, if any, in the range of 1 to total\_height, inclusive.

app\_specific\_parameters is a parameter that provides an extension point to define codec specific parameters.

# Integrating new codecs

Issue: [*http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/95*](http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/95)

## Introduction

New codecs (audio, video, text, …) are regularly defined within MPEG or externally, and that need to be integrated into ISOBMFF. After 20 years of evolution, the best way to integrate a new codec is not necessarily the same way it was years ago. This section contains a discussion on this topic for the purpose of improving the documentation, for example Annex B.5 of ISOBMFF.

There are 2 high level questions one can ask when integrating a codec into ISOBMFF:

* Given a codec already specified, how to best integrate it into ISOBMFF? What tools should I use for this and that?
* Given a codec under development, how to structure its high level syntax such that it integrates well with ISOBMFF and what are the pitfalls to avoid?

These questions are discussed below.

## Integrating an existing codec

The following questions should be asked when new codecs are integrated in ISOBMFF:

* Which handler should be used? Should a new handler be defined?

Although it is not documented (yet?), the underlying assumptions of a track handler are the following:

* Codecs within a handler are interchangeable, in the sense that once track data is decoded, all codecs should produce the same type of output. In other words, theoretically within a track you can have multiple sample description entries each with a different codec for that handler. (But one should not do that!!!)
* Each handler comes with a base sample entry (vide -> VisualSampleEntry, soun -> AudioSampleEntry, …). If a base sample entry contains the necessary fields for a codec (sample rate or width/height), probably the codec should use the corresponding handler (if the base sample entry is not enough, extend it using the base sample entry as a parent class). Conversely, if there is no corresponding base sample entry (or if there is no such media type yet supported), a new handler should be created.
* How to design the codec-specific Sample Entry ?

**About Decoder initialization.** It has been assumed in the past that whatever is needed for decoder initialization (including opaque sequences of bytes) should be in the sample entry. Doing so should be considered carefully as it leads to the problematic dichotomy: live vs. ondemand (avc1 vs avc3, hvc1 vs. hev1, …). In live cases, all opaque sequences of bytes for the entire session are not necessarily known upfront, and creating a new sample entry on the fly is not (yet?) possible in ISOBMFF. Usually the concern with decoder initialization is initialization latency, but often latency is due to memory allocation which for example for video can be done by knowing width, heigh, and depth. MPEG welcomes feedback on how decoder initialization is done for various codecs. One can use this public issue https://github.com/MPEGGroup/FileFormat/issues/58.

**How to compute “codecs” parameter?** In segmented media, it should be possible to compute the “codecs” parameter without having to fetch any media segment, and only the initialization segment. This should be done primarily based on the information in the sample entry. Sample group information should not be used (because that would remove freedom to put sample group descriptions in the fragments).

**Consider content splicing complexity.** Content splicing (i.e. merging 2 single-track files with the same handler and codec into 1 single-track file) is a typical operation that can be performed with ISOBMFF files. It can always be done by using 2 sample description entries in the output track. However, using multiple sample entries makes implementation more complex (and the “codecs” parameter only describes the first one). A proposed rule is to minimize the amount of data in the sample description in order to maximize the opportunity for single sample description splicing. Processing sample entries when splicing should be as simple as "doing a binary comparison of the 2 input sample entries and if they differ produce 2 output sample entries". Alternatively, having specific merge rules such that “if the fields A, B, C differ between sample entries 1 and 2, create a sample entry with max (A1,A2), max(B1,B2) max(C1,C2)"

A good way to reduce the amount of information in the sample entry is to use **sample groups**. All samples of a track can be easily made part of a common sample group, for example using the default sample group in the ‘sgpd’ box without even using an ’sbgp’ box. This should be considered in particular instead of defining a new sample entry child box type (e.g. colr, pasp, dmix, …).

## Adjusting a codec under development for better integration

In order to get integrated simply and to produce less error-prone files, codecs should have some good properties as discussed here. Designers of high level syntaxes of new codecs should try to follow these recommendations:

* **Enable shallow parsing.** The basic operation of an ISOBMFF packager is to read an elementary stream and produce an ISOBMFF file. Therefore, it should be simple for the packager to identify what will constitute a sample. It should be able to parse the stream (without decoding it fully) and to identify elements that are needed by the packager.
  + Examples of stream types following this good practice are: NALU-based streams, OBU-based streams, TLV-based streams.
  + Example of stream types not following such approach: AAC (non-ADTS) streams.
* **Separate information with different update rates**. Streams typically have information that vary over time, but at different rates. For example, in a video stream, slice-level information will vary frequently and be different for each sample. Picture Parameter Set may vary at each sample but typically varies at GoP granularity or more. Sequence Parameter Sets vary even less, e.g. only a few times in the scope of a sample entry. Profile and level values typically don’t vary within a track. Designers of new codecs should structure data in a way that the packager can easily identify data that it needs and that varies at different rates to store them at the appropriate place (sample entry, sample group, sample). This can mean creating new units (NALU, OBU, Packet) specifically for each type of data.
* **Reduce elementary stream/container file redundancy.** In today’s designs, there is often redundancy between elementary stream information and file level information. For example, width and height are stored in the elementary stream but also in the visual sample entry. The same is true for a lot of information: color (VUI and colr box), HDR static metadata (SEI and ‘mdcv’/’clli’ boxes), sample rate, frame rate, etc. Often the information is replicated at the ISOBMFF level in a codec-agnostic way and to simplify the processing at the client side. But this is a source of error when only one level is modified and not the other one. Approaches to overcome this problem include:
  + Design data units that are codec-agnostic, leveraging CICP or equivalent whenever possible.
  + Consider the possibility of the packager replacing entire data units with ISOBMFF structures when storing in ISOBMFF and the demuxer restoring these data units from the ISOBMFF structures. For example, if a stream had a color (NAL/OB)unit, the packager could avoid storing that unit, replacing it with a ‘colr’ box, and upon reading the demuxer would recreate a (NAL/OB)unit from the ‘colr’, if needed or directly communicate the ‘colr’ box to the decoder/renderer.
* **Ease identification of encryptable data**. Often when encrypting streams, payload headers are kept in the clear. However, it is not always easy to determine where those headers end without having to parse the entire header. Codec designs should consider facilitating this (adding header length, using fixed headers, …).

# MovieFragmentHeaderBox update

Issue : [*http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/97*](http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/97)

MPEG is considering defining a new version of the ‘mfhd’ box as follows:

aligned(8) class MovieFragmentHeaderBox

extends FullBox('mfhd', version, flags){

if (version == 0) {

unsigned int(32) sequence\_number;

else if (version == 1) {

unsigned int(64) sequence\_number;

}

}

With the additional semantics:

When version 1 is used, the following ‘flags’ values have the following meaning:

0 : the creator makes no statements, promises, warranties about how sequence\_number is updated

1 : the sequence\_number in this moviefragment is larger than the sequence\_number in the preceding one

3 : the sequence\_number in this moviefragment is one greater than the sequence\_number in the preceding one

# Sample Run Sample Group

Issues :

[*http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/98*](http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/98)

[*https://git.mpeg.expert/MPEG/Systems/FileFormat/isobmff/-/issues/286*](https://git.mpeg.expert/MPEG/Systems/FileFormat/isobmff/-/issues/286)

## Introduction

The overhead of file format metadata, such as the TrackRunBox(es), could be significant especially when it comes to applications using multi-track approach. The overhead of TrackRunBox(es) has been clearly established and thoroughly studied in the context of OMAF tile based streaming [1-3]. For convenience, an example from [1] is copied below.

*“Imagine a 4K video being HEVC encoded with 50 tiles, each tile measuring 384x384 pixels. In such a tile, it is not uncommon for a NAL unit containing a P or B-slice to be as small as 40 bytes, and in areas with little motion sometimes even significantly less. In such a case, the 32-bit sample\_size already results in at least a 10% TrackRunBox overhead, and that’s assuming all of the other optional TrackRunBox fields have been disabled.”*

Conventionally, file writers operate by parsing the high-level syntax of a given input video bitstream and generate the file format metadata from the information of the bitstream. Consequently, under certain constraints, the information present in TrackRunBox(es) could instead be regenerated at the client side by parsing the high-level syntax of the received bitstreams.

More details are provided in Section 14.2, below.

## Discussion

For video bitstreams encapsulated as a track, the information of the TrackRunBox could be tightly packed at a coarser granularity and additional finer-level information can be concluded in the file reader based on the received MediaDataBox for a movie fragment. Figure 19.2.1 shows the conventional encapsulation of TrackRunBox where a encoded segment has m access units, with each access unit having either VCL NAL units or non-VCL units or both. Information about each access unit is encapsulated in a sample of the track as part of the TrackRunBox.

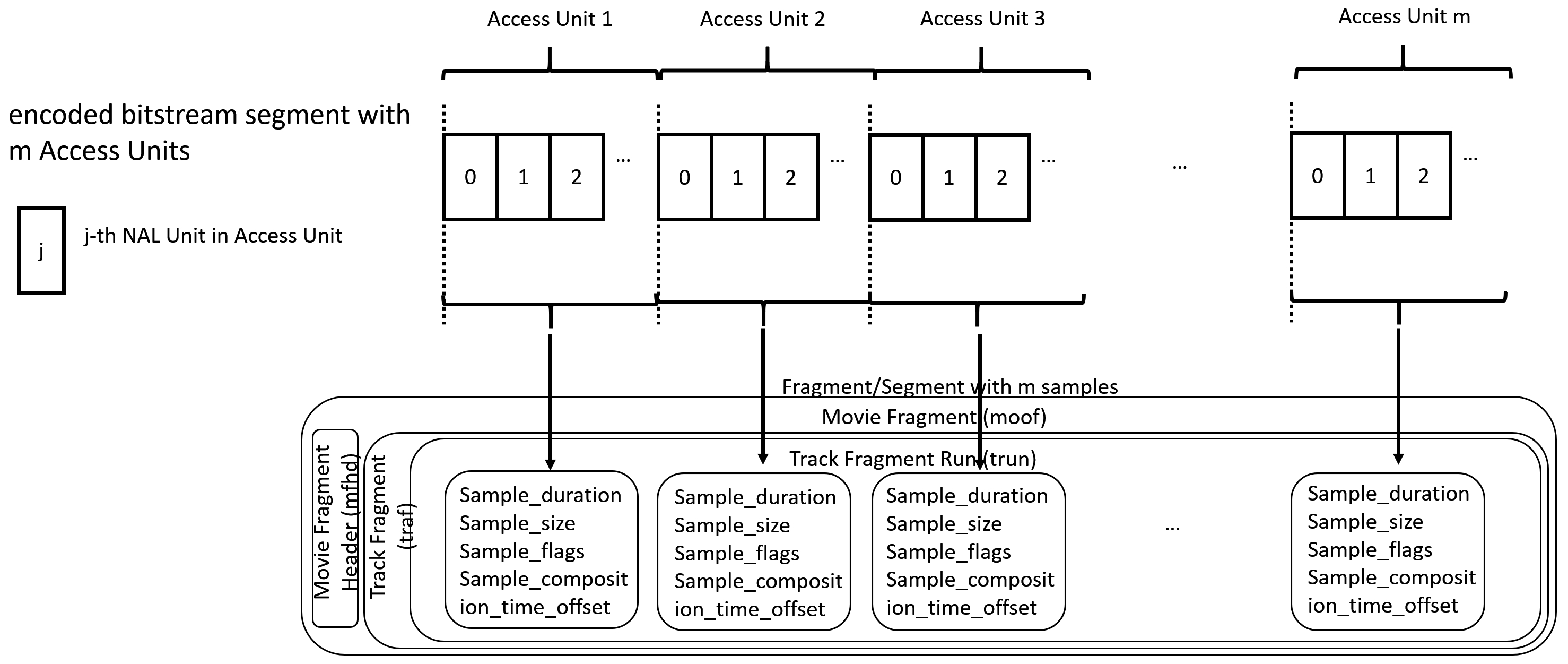


Figure 19.2.1: Conventional sample information in TrackRunBox

As shown in Figure 19.2.2, the sample-level information of the TrackRunBox (m samples) are tightly packed into a single sample information in the SampleRun sample group. Kindly note that the packing of m samples into a single sample in Figure 19.2.2 is only for demonstrating the usage of SampleRun sample group. The SampleRun sample group may contain information about n < m samples where more than one sample of the original TrackRunBox are tightly packed into the SampleRun sample group samples.

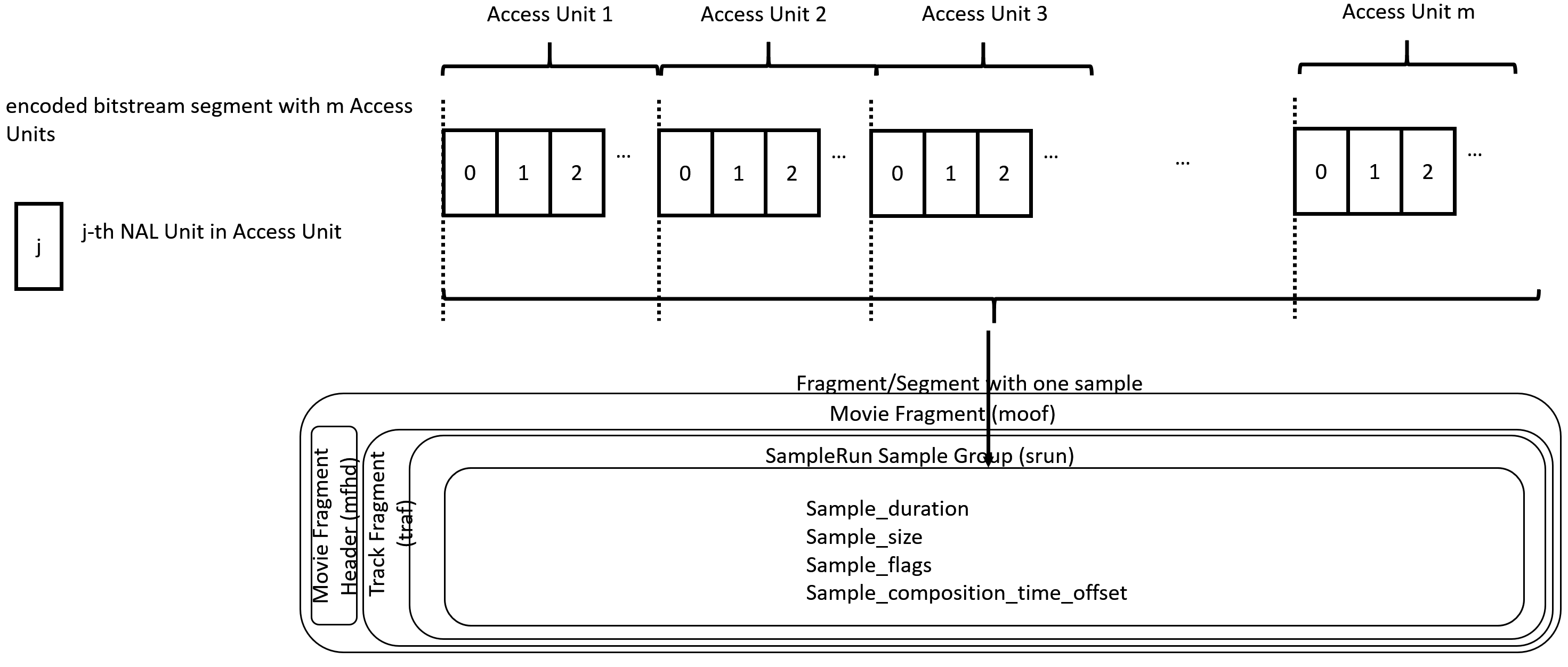


Figure 19.2.2: An example usage of SampleRun Sample group.

An approach for optimizing the delivery of MovieFragmentBox for video could be designed as follows.

* A file writer performs the following to pack a MovieFragmentBox:
  + Identifying GOP structures where pictures have the same pattern of duration, composition time offset and sample flags.
  + Generating a SampleRun sample group description entry for each identified GOP structure.
  + Merging the samples of each GOP into one sample in an 'spkt' transformed video track.
  + Mapping the samples in the 'spkt' transformed video track to the respective SampleRun sample group description entries in SampleToGroupBox(es).
* A player:
  + Performs the access unit boundary determination as specified in AVC, HEVC, VVC, or EVC for each sample in an 'spkt' transformed video track.
  + Generates the original TrackRunBox information in MovieFragmentBoxes based on the signalled SampleRun sample group and the sample sizes determined from the access unit boundaries.

## Proposal

It is proposed to support the new SampleRun sample group as follows.

### Definition

A sample run sample group may be present in a 'spkt' transformed video track. A sample run documents contiguous samples of the untransformed track where each access unit in the mapped sample of the 'spkt' transformed video track is its own sample.

Note: The sample run sample group is limited to be used in a 'spkt' transformed video track containing bitstreams which support picture boundary detection. For example, VVC, which supports signalling of picture header information and the client need not maintain a parsing context to detect picture boundaries.

### Syntax

class SampleRunEntry() extends SampleGroupDescriptionEntry('srun') {  
 unsigned int(24) flags;

if (flags & 0x000008)

unsigned int(32) default\_sample\_duration;

if (flags & 0x000010)

unsigned int(32) default\_sample\_size;

if (flags & 0x000020)

unsigned int(32) default\_sample\_flags;

unsigned int(32) sample\_count;

if (flags & 0x000004)

unsigned int(32) first\_sample\_flags;

{

if (flags & 0x000100)

unsigned int(32) sample\_duration;

if (flags & 0x000200)

unsigned int(32) sample\_size;

if (flags & 0x000400)

unsigned int(32) sample\_flags;

if (flags & 0x000800)

signed int(32) sample\_composition\_time\_offset;

}[ sample\_count ]  
}

### Semantics

flags is a map of flags

The following flags are allowed to be set in the flags:

0x000008 default-sample-duration-present

0x000010 default-sample-size-present

0x000020 default-sample-flags-present

0x000004 first-sample-flags-present; this overrides the default flags for the first sample only. This makes it possible to record a group of frames where the first is a key and the rest are difference frames, without supplying explicit flags for every sample. If this flag and field are used, sample-flags-present shall not be set.

0x000100 sample-duration-present: indicates that each sample has its own duration, otherwise the default is used.

0x000200 sample-size-present: each sample has its own size, otherwise the default is used.

0x000400 sample-flags-present; each sample has its own flags, otherwise the default is used.

0x000800 sample-composition-time-offsets-present; each sample has a composition time offset.

default\_sample\_duration: indicates the default duration of the samples in the sample run

default\_sample\_size: indicates the default size of the samples in the sample run

default\_sample\_flags: indicate the default flags values for the samples in the sample run.

sample\_count the number of samples in this sample run

first\_sample\_flags provides a set of flags for the first sample only of this sample run.

# Improvements to movie fragments and dynamic tracks

The File Format group contained in its mandate the following topic:

1. Study improvements to movie fragments especially when recording e.g. at gateways, and dynamic tracks.

The inputs below are considerations and proposals addressing this topic.

## Dependent Movie Fragments

Reference to discussions: <http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/148>

### Discussion

With low-latency delivery techniques for ISOBMFF, it is quite common that demultiplexers synchronize on the first fragment in a series (e.g., segment) but do not try to demultiplex the following fragments first, typically because the SAP frame is in the first fragment of the series. This means that a lot of information is duplicated in the following fragments due to ISOBMFF rules, but are identical with the information sent in the first segment used to synchronize. This applies to:

* sample group descriptions inserted in movie fragments
* MetaBox, UserDataBox
* Any possible extension of ISOBMFF allowing box injection in movie fragments that do not carry per-sample information

This is because ISOBMFF makes no difference between movie fragments in terms of random access for the parser, while higher delivery protocols impose different constraints on fragments (first fragment of a segment is a SAP1, must have a TFDT, etc…).

For example, a typical ‘seig’ sample group description inserted in each fragment to allow for key rolling will cost, assuming a single entry (one key) is used:

* 61 bytes if constant IV 128bits is used (cbcs)
* 46 for 128 bits IV (cenc)

At 100ms fragment duration, this leads to costs of 3.7 / 4.9 kbps.

In very low latency cases (one sample per fragment):

* for 25fps video : 9.2 / 12.2 kbps
* for AAC at 44100Hz: 16 / 21 kbps.

The signaling rate will obviously increase:

* when adding other sample group descriptions to be updated in the fragments or other boxes such as MetaBoxes
* when more complex sample descriptions are used, e.g. multiple keys per sample.

### Proposal

It is proposes to define a new version in the movie fragment header box:

aligned(8) class MovieFragmentHeaderBox extends FullBox('mfhd', version, 0){   
 unsigned int(32)sequence\_number;   
}

If version is not 0, any SampleGroupDescriptionBox, UserDataBox or MetaBox defined in the last movie fragment, or in the last TrackFragmentBox in the last movie fragment, whose MovieFragmentHeaderBox version is 0 also apply for this movie fragment, and there shall not be any SampleGroupDescriptionBox or MetaBox defined for this movie fragment.

Note that this new version of a movie fragment can still be parsed, in terms of box structure and sample information, independently of the previous fragment, however its interpretation usually re-use information from a previous movie fragment.

### Comments received at MPEG#140

* We already have a new version in the TuC (see Section 13), should we use flags instead?
* Should we pre-declare in the ’mvex’? or ’moov’?
* The use of 'dmof' instead of 'moof' as an option should be studied, so that old readers don't get puzzled when they see something that is erroneous in their opinion.
* we may not always have sample group or encryption for true low latency
* meta and udta should not be in the fragments, they are not mandatory at all therefore rotating keys based on the segments, maybe better to use the manifest for key rotation instead? (but would assume that a manifest is always present).

## On dynamic tracks in fragments

This topic comes from the discussions related to the input contribution m61140:

<http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/147>

The following comments were received at MPEG#140:

- Relying only on brands for parsers to correctly understand the features can be dangerous, and a possible rename of movie fragments boxes should be considered, for example ‘dmof’ instead of reusing ‘moof’.

- Some of the proposed functionalities can be achieved using external signaling such as DASH MPD.

### Discussion

#### Signaling cost

As noted in m58085, the proposed signaling of TrackBox in movie fragments is 4kbps on average for a 1 sec duration segment. Striping the TrackBox from non-essential box (empty sample table, data references) gives around 3.1 kbps for 1s segments.

It is proposed hereafter a further reduction of the track box for inclusion in movie fragments, with an achieved overhead of roughly 1.3 kbps for 1s segments.

#### Change tracking

In use cases such as MPEG-2 TS to fMP4 gateways, the proposed dynamic tracks could be further improved by signaling (un)changed configurations in the fragments, usually called “carousel” in broadcast terminology.

It is noted that sample can be signaled as repeated samples using dependency flags (MPEG-4 systems), but we lack support for such signaling for non-sample data:

* MetaBox
* UserDataBox in track fragment
* Sample Group Description

Obviously relying on the box version is a bad choice:

* Some boxes ('udta') do not have versions
* Other boxes have a version field but already use it (sample group description)
* It is not the intended purpose of box version, which is to indicate variation in the binary syntax and not in the payload.

One approach to deal with this problem is to perform a comparison of past and current boxes, typically through a hashing function. While this allows detecting identical configurations, it has several drawbacks:

* It is costly in client resources
* It does not allow for signaling a repeated configuration with slight variations (in the meta-data for example) that do not require reparsing of the data

If we want dynamic tracks or sample description changes in movie fragments (regardless of the method), it could be useful to introduce some change detection mechanism so we can properly identify repeated information across fragments.

#### Track Removal

In MPEG-2 TS, media streams can be removed due to a PMT update. We noticed that track fragments with duration-is-empty flag set could be used to signal no samples apply to this track. It has however some drawbacks:

* The flag cannot be used if an edit list is present for this track in the movie box,
* It cannot reliably be used as a hint that the track is no longer present; this implies that
  + resource optimization (closing decoders/buffers/etc.) cannot be fully performed.
  + Remixing to e.g. MPEG-2 TS might trigger unused PIDs in the PMT (i.e. PID declared but no packet for this PID), with unpredictable result at the demuxer side
* Moreover, support for this flag is not very good, typically triggering rebuffering in some MSE implementations (because ignored)

If we want a proper/unambiguous signaling of track removal, we need a dedicated box/field to indicated tracks no longer present.

#### File Concatenation, splicing or time-aligned track addition

MPEG-2 TS and some other delivery formats can embed splicing information, allowing a media pipeline to identify temporary service reconfigurations.

While investigating support for such signaling in ISOBMFF along with dynamic tracks, we faced an old but never resolved topic in ISOBMFF: how can we simply build files out of a collection of files, whether as a sequence (‘append a presentation to another one’), or as a set of additional tracks (‘add these tracks, time-aligned’).

In fragmented mode, addition of time-aligned tracks requires editing of the 'moov' to inject the track, and then direct concatenation can work, assuming track IDs do not conflict and timestamps origin is 0 for each “track file”. This obviously will result in one media track being stored after all other ones, but the result is a valid ISOBMFF.

Extending a fragmented presentation with another one is more problematic:

* trackIDs may need to be rewritten: if not the same in each presentation, some tracks may have to be added to the initial 'moov'
* decoder configuration may need to be updated
* timing needs to be rebuilt:
  + alignment of timestamps at boundaries
  + introduction of “gaps” in the timeline to deal with AV sync
* Signaling codec priming (e.g. AAC) is even more complex: a dedicated edit list needs to be created to remove the priming period at the beginning of the second stream.

When concatenating files to splice new content (A1->B->A2), we faced the following issues:

* If the splice is a content replacement,
  + The timing of the injected content B must be rewritten
  + There is no guarantee that the injected content B ends up exactly at the right position, in which case the timing of A2 may need to be rewritten
* If the splice is a content injection (timeline is extended),
  + The timing of the injected content B must be rewritten
  + The timing of A2 must be rewritten

We therefore thought that it would be interesting to design dynamic tracks such that file concatenation is possible without any rewrite. We designed the proposal to allow:

* “early-splicing”, where the source timeline is spliced but contiguous and the only need is to signal the splice points (e.g. MPEG-2 TS to fMP4 gateway)

“late-splicing”, where splicing is done after the fMP4 muxing stage (non-contiguous timelines)

### Proposal

#### Design

We reworked the proposal from m58085 to provide a single method for declaring new tracks and declaring new sample description(s) in an existing track, based on the previous observations.

The proposal defines a DynamicMovieBox in a movie fragment, containing

* Indication on source (timeline/splicing) and configuration changes compared to previous DynamicMovieBox
* Zero or one UserData box, Zero or one Meta box
* Zero or more DynamicTrackBox, each containing
  + a specific DynamicTrackHeaderBox containing
    - track setup (width/height/delay…)
    - indications on modification changes compared to previous DynamicTrackBox
  + some common boxes found in TrackBox.

##### Dynamic movie box

###### Definition

Box Type: 'dymv'  
Container: MovieFragmentBox  
Mandatory: No  
Quantity: Zero or one

A DynamicMovieBox completely or partially overrides the MovieBox setup (track list, user data and meta) for the current fragment.

Each DynamicMovieBox has an associated source\_id, which indicates how the movie fragment extends the initial MovieBox:

* A source\_id equal to 0 indicates that the MovieBox is modified by the movie fragment,
* A source\_id different from 0 indicates that the MovieBox is ignored (i.e. considered not present)

A change of source\_id between two consecutive movie fragments N and N-1 in a single bytes sequence (file, remote resource, etc.) indicates that tracks in N shall be considered as new tracks and tracks in N-1 shall no longer be considered present in the file. In this case, there is no guarantee that the timeline is contiguous between fragments N and N-1. How file readers handle such discontinuities is out of scope of this specification and usually driven by the processing pipeline capabilities. Implementations should however avoid introducing long playback gaps at source\_id change points.

When dynamic tracks are used, the first track fragment of each track in the parent movie fragment shall have a TrackFragmentBaseMediaDecodeTimeBox.

If two consecutive movie fragments N and N-1 have the same value for source\_id, the timeline of all tracks active in both fragments is contiguous, i.e. the constraints on TrackFragmentBaseMediaDecodeTimeBox of each track shall be respected: for a track fragment with the same trackID, the first TrackFragmentBaseMediaDecodeTimeBox in movie fragment N is equal to or greater than the first TrackFragmentBaseMediaDecodeTimeBox in movie fragment N-1 plus the sum of the sample durations in movie fragment N-1.

If several tracks need to be inserted or replaced, these tracks may be declared

* all in a single DynamicMovieBox (i.e. a single movie fragment),
* each one in its own DynamicMovieBox (i.e. one movie fragment per new track) or
* a mix of both approaches.

If a track from the MovieBox is not listed either for update or removal in a DynamicMovieBox with source\_id value of 0, it is valid, but there could be no track fragment for this track in the movie fragment, as is the case with regular movie fragments.

A DynamicMovieBox may contain zero or more DynamicTrackBox.

NOTE Usage of DynamicMovieBox with source\_id different from 0 and zero DynamicTrackBox can be used to force a discontinuity between two movie fragments. Usage of DynamicMovieBox with source\_id equal to 0 and zero DynamicTrackBox can be used to update MetaBox or UserDataBox.

Tracks declared or modified in a DynamicMovieBox may have the same configuration for several consecutive movie fragments. source\_flags allow a file parser to detect that a DynamicMovieBox is a repetition of the previous DynamicMovieBox with the same values for source\_id and bundle\_id. For a same value of source\_id, if multiple dynamic tracks or movie-related metadata (user data, meta) are modified or declared in more than one DynamicMovieBox, then each of these DynamicMovieBoxes shall use a different bundle\_id.

Presence of dynamic tracks in movie fragments shall be indicated using the brand ‘dytk’ in the ExtendedTypeBox or by using the brand ‘isod’ or higher in the FileTypeBox. The MovieBox is not mandatory when using dynamic tracks, but in that case the first movie fragment loaded shall have a FileTypeBox or ExtendedTypeBox indicating support for dynamic track.

EDITOR’S NOTE: we could make FileTypeBox optional in this case but have SegmentTypeBox mandatory.

For DynamicMovieBox the following flags are defined:

* 0x000001 source-info-present if set, indicates that source information is present; if not set, source\_id, bundle\_id and source\_flags take the value 0.
* 0x000002 in-splice if set, indicates that the tracks described in the DynamicMovieBox correspond to a content splice period and will soon move back to previous configuration. By monitoring this flag and the source\_id field, the processing media pipeline can be optimized if desired (e.g. avoid unloading/reloading decoder resources for instance). This flag shall not be set if source\_id is 0.

EDITOR’S NOTE: we could also use a flag in 'styp' to signal this, to simplify edition of files during concatenation

###### Syntax

aligned(8) class DynamicMovieBox extends FullBox('dymv', version=0, flags){   
 if (flags & 1) {  
 unsigned int(32) source\_id;  
 unsigned int(32) bundle\_id;   
 unsigned int(24) source\_flags;  
 } else {  
 source\_id=0;  
 bundle\_id=0;  
 source\_flags=0;  
 }   
 DynamicTrackBox track; // optional: zero or more  
 UserDataBox user\_data; // optional: zero or one   
 MetaBox meta; // optional: zero or one   
}

###### Semantics

source\_id identifies the origin of the fragment. The value 0 indicates that the source is the MovieBox andthe DynamicMovieBox modifies the MovieBox. Other values identify another source than the MovieBox and all tracks, UserDataBox, MetaBox and any other properties defined in the MovieBox shall be ignored.

bundle\_id provides an identifier for tracking partial configuration changes for a given source\_id.

NOTE: The bundle\_id is typically needed when two or more dynamic tracks are inserted in initial movie or during a splice period, each in their own movie fragment. This allows a file reader to detect that the changes advertised for a given bundle have already been processed in a preceding movie fragment.

source\_flags identify the modifications declared in this DynamicMovieBox compared to the previous DynamicMovieBox with the same value of source\_id and bundle\_id. The following flags are defined:

* 0x000001 if set, indicates that one or more tracks configurations have changed
* 0x000002 if set, indicates that the global (MovieBox-level) user data has changed
* 0x000004 if set, indicates that the global (MovieBox-level) meta box has changed
* 0x800000: if set, indicates that the modifications are functionally equivalent to the previous DynamicMovieBox with the same source\_id and bundle\_id. When this flag is set, a file reader may safely skip processing the DynamicMovieBox if a previously parsed DynamicMovieBox has the same source\_id and bundle\_id. Otherwise (this is the first DynamicMovieBox parsed with this source\_id and bundle\_id values), the flag may be set but shall be ignored (i.e. considered as not set) by file readers.

When source\_flags is not set (either explicitly or per the above rule) or has the value 0, the box shall not be skipped. In this case, there is no information regarding modifications of child boxes compared to previous DynamicMovieBox; and the entire content of the box must be re-evaluated.

When source\_flags is not set to 0x000001, any DynamicTrackBox present in this DynamicMovieBox shall have the 0x800000 modification\_flags set.

##### Dynamic track box

###### Definition

Box Type: 'dytk'  
Container: DynamicMovieBox  
Mandatory: No  
Quantity: Zero or more

A DynamicTrackBox declares a new track or modifies an existing track for the duration of the parent movie fragment.

Tracks declared by a DynamicTrackBox, and for which

* the associated source\_id is not 0,
* or there are no tracks with a matching track\_ID in the MovieBox

implicitly declare a TrackExtendsBox with the value default\_sample\_description\_index set to 1 and the values default\_sample\_duration, default\_sample\_size, default\_sample\_flags set to 0.

NOTE This implies that default values will likely need to be set in the TrackFragmentHeaderBox; if multiple track fragments are used for a dynamic track within one movie fragment, the default values may need to be re-coded for each track fragment.

###### Syntax

aligned(8) class DynamicTrackBox extends Box('dytk'){  
 DynamicTrackHeaderBox dyn\_tkhd;//mandatory, must be first  
 SampleDescriptionBox stsd;//conditionally mandatory  
 Box minf\_header\_info; //optional, one of vmhd, smhd, sthd, hmhd…  
 DataInformationBox data\_info; //optional  
 UserDataBox udta; //optional  
 MetaBox meta; //optional  
 TrackReferenceBox tref; //optional  
 TrackGroupBox trgr; //optional  
}

###### Semantics

data\_info if present, gives the source(s) of the samples’ data for this dynamic track. If not present, the samples’ data is present in the container.

minf\_header\_info if present, gives the media-specific header box usually found in the MediaInformationBox of a track with the same handler\_type as this dynamic track. Derived specification may mandate its presence.

Other boxes contained in the DynamicTrackBox (except DynamicTrackHeaderBox) have unchanged semantics. When present, they replace their counterpart boxes in the TrackBox (and children) of the MovieBox.

##### Dynamic track header box

###### Definition

Box Type: 'dtkh'  
Container: DynamicTrackBox  
Mandatory: Yes  
Quantity: One

A DynamicTrackBox can be used to disable an existing track from the MovieBox, override the definition of an existing track from the MovieBox or define a completely new track.

NOTE The DynamicTrackBox is a compaction of TrackHeaderBox, EditListBox and MediaHeaderBox in a single container, in order to keep the track signalling overhead low.

The following flags are defined for a dynamic track header box:

0x000001 dyn\_tk\_ignore\_track indicates, when set, that the track declared in the MovieBox or in a previous DynamicMovieBox with the same source\_id as the parent DynamicMovieBox should be ignored (treated as if not present) until a next movie fragment is received for this track.

Flag values 0x000002, 0x000004, 0x000008, 0x000010, 0x000020 and 0x000040 are used for box parsing. Other values are reserved.

If flag dyn\_tk\_ignore\_track is not set in DynamicTrackHeaderBox, the parent DynamicTrackBox overrides an existing track or declares a new track. In this case:

* If stsd is present in the parent DynamicTrackBox: if there is a track with same ID in the MovieBox, overwrite it with current track, otherwise add the new track to the presentation

NOTE1: derived specification can mandate that the handler type / timescale / width/height remain the same in this case

NOTE2: This is typically used to update a sample description for a track

* If stsd is not present in the parent DynamicTrackBox : there shall be a track with same ID in the MovieBox and all fields in DynamicTrackHeaderBox shall match their counterpart fields in the track/handler/media header boxes declared in the MovieBox. This is used to update UserDataBox, MetaBox, TrackReferenceBox, TrackGroupBox of the track. The sample descriptions of the track remain unchanged in this case.

EDITOR’S NOTE: we could also use a dedicated flag for the case where no 'stsd' is present, in order to avoid re-listing things that shall match what is in the 'moof'’s track

###### Syntax

aligned(8) class DynamicTrackHeaderBox extends FullBox('dtkh', version=0, flags){  
 unsigned int(32) track\_ID;  
 if (!(flags & 1)) {  
 if (flags & 2)   
 unsigned int(24) modification\_flags;  
 unsigned int(32) handler\_type;  
 unsigned int(32) media\_timescale;  
 if (flags & 4)   
 signed int(32) delay;  
 if (flags & 8)   
 unsigned int(24) track\_flags; //as in TrackHeaderBox  
 bit(1) lang\_3cc;  
 if (lang\_3cc) {  
 unsigned int(5)[3] language; // ISO-639-2/T language code  
 } else {  
 bit(7) reserved;  
 utf8string extended\_language

}  
 int(16) alternate\_group;  
 if (flags&16) {  
 unsigned int(32) width;  
 unsigned int(32) height;  
 int(16) layer;  
 if (flags&32) {  
 int(32)[9]matrix ;  
 }  
 }  
 if (flags&64) {  
 int(16)volume;  
 }  
 }  
}

###### Semantics

track\_ID indicates the ID for the track declaration

modification\_flags identify the modifications in the parent DynamicTrackBox compared to the previous DynamicTrackBox with the same value of track\_ID and the same value of source\_id in the parent DynamicMovieBox. These flags may be used by the reader to optimize processing of consecutive track fragments with the same track\_id and source\_id. The following flags are defined:

* 0x000001: the track configuration has changed (changes in one or more fields other than modification\_flags in DynamicTrackHeaderBox, or changes in the associated media header box (‘vmhd’, ‘smhd’, etc.) or changes in the associated data information box)
* 0x000002: the media configuration has changed (new sample description, ‘stsd’)
* 0x000004: the track UserDataBox has changed
* 0x000008: the track MetaBox has changed
* 0x000010: the track TrackReferenceBox has changed
* 0x000020: the track TrackGroupBox has changed
* 0x800000: the track modifications are functionally equivalent to the previous DynamicTrackBox with the same track\_id and source\_id. This flag may be set but shall be ignored (i.e. considered as not set) by file readers when the previous DynamicMovieBox has a different source\_id or when this is the first DynamicTrackBox parsed for this track\_id and source\_id.

When modification\_flags is not set (either explicitly or per the above rule) or has the value 0, there is no information available on possible changes of the track compared to previous DynamicTrackBox with the same track\_id and source\_id and the entire content of the DynamicTrackBox must be re-evaluated.

handler\_type same as handler\_type in HandlerBox

media\_timescale same as timescale in MediaHeaderBox

delay indicates media delay of the track, in media\_timescale. The presentation time of any sample in the track is the sum of the composition time of the sample and this value. A negative presentation time indicates that (part of) the sample data shall not be presented (media skip). If not coded, the value 0 is used.

track\_flags same as flags in TrackHeaderBox. If this field is not coded:

* If source\_id of the dynamic track is 0 and there is a matching track with the same track\_ID in the MovieBox, the flag values from the TrackHeaderBox are used,
* Otherwise, this field value is inferred to be 0x000003

language same as language in MediaHeaderBox

extended\_language same as in ExtendedLanguageBox

alternate\_group same as in TrackHeaderBox

width same as in TrackHeaderBox. If not coded, the media width, after pixel aspect ratio and clean aperture applied, is used

height same as in TrackHeaderBox If not coded, the media height, after pixel aspect ratio and clean aperture applied, is used

layer same as in TrackHeaderBox. If not coded, the layer is 0

matrix same as in TrackHeaderBox. If not coded, the identity matrix is used

volume same as in TrackHeaderBox. If not coded, full volume (1.0) is used

### Examples

### Sample description update

When a sample description changes for fragments N to K, this can be expressed as:

* Fragment N->K :
  + 1 DynamicMovieBox with source\_id=0, (bundle\_id= any), source\_flags=0x800000 (or 0x800001 as a hint).
  + 1 DynamicTrackBox with one stsd
  + DynamicTrackHeaderBox with track\_ID of the updated track and no modification\_flags or
    - modification\_flags=0x800002 if no track layout change
    - modification\_flags=0x800003 if track layout change
* Fragment K+1-> :
  + 0 DynamicMovieBox (if same config from moov can be used)

### Track addition

When **a new track appears** for fragments N to K, this can be expressed as:

* Fragment N->K :
  + 1 DynamicMovieBox with source\_id=0 (bundle\_id= any), source\_flags=0x800000 (or 0x800001 as a hint).1 DynamicTrackBox with one stsd
  + DynamicTrackHeaderBox with new track\_ID (different from all previous tracks due to source\_id=0) with
    - modification\_flags=0x800000
* Fragment K+1-> :
  + 0 DynamicMovieBox (if same config from moov can be used)

### Splicing AV contained in a single movie fragment

When splicing AV content with T tracks contained in a single movie fragment for fragments N to K, this can be expressed as:

* Fragment N->K :
  + 1 DynamicMovieBox with flags=3 (source-id-present and in-splice), source\_id>0, bundle\_id=0, source\_flags=0x800000 (or 0x800001 as a hint)
  + T DynamicTrackBox (any track\_ID) with one stsd each
    - DynamicTrackHeaderBox with modification\_flags=0x800000
* Fragment K+1-> :
  + 0 DynamicMovieBox (if same config from moov can be used)

### Splicing AV contained in separate movie fragments

When splicing AV content contained in separate movie fragments for fragments Ni to Ki, i representing each media, this can be expressed as:

* Fragment Ni->Ki :
  + 1 DynamicMovieBox with flags=3 (source-id-present and in-splice), source\_id>0, bundle\_id=***i***, source\_flags=0x800000 (or 0x800001 as a hint)
  + 1 DynamicTrackBox (any track\_ID) with one stsd
    - DynamicTrackHeaderBox with modification\_flags=0x800000
* Fragment Ki+1-> :
  + 0 DynamicMovieBox (if same config from moov can be used)

### Splicing with configuration changes in a single movie fragment

When splicing AV content contained in a single movie fragment for fragments N to K with codec configuration change happening from C1 to C2 (N < C1 < C2 < K), this can be expressed as:

* Fragment N->C1 :
  + 1 DynamicMovieBox with flags=3 (source-id-present and in-splice), source\_id=1, bundle\_id=0, source\_flags=0x800000 (or 0x800001 as a hint)
  + T DynamicTrackBox (any track\_IDs called *splice\_track\_IDs*) with one stsd each
    - DynamicTrackHeaderBox with modification\_flags=0x800000
* Fragment C1 :
  + 1 DynamicMovieBox with flags=3 (source-id-present and in-splice), source\_id=1, bundle\_id=0, source\_flags=0x000001
  + T DynamicTrackBox (*splice\_track\_IDs*) with one stsd each
    - DynamicTrackHeaderBox with modification\_flags=0x000002
* Fragment C1+1->C2 :
  + 1 DynamicMovieBox with flags=3 (source-id-present and in-splice), source\_id=1, bundle\_id=0, source\_flags=0x800000 (or 0x800001 as a hint)
  + T DynamicTrackBox (*splice\_track\_IDs*) with one stsd each
    - DynamicTrackHeaderBox with modification\_flags=0x800002
* Fragment C2+1 :
  + 1 DynamicMovieBox with flags=3 (source-id-present and in-splice), source\_id=1, bundle\_id=0, source\_flags=0x000001
  + T DynamicTrackBox (*splice\_track\_IDs*) with one stsd each
    - DynamicTrackHeaderBox with modification\_flags= 0x000002
* Fragment C2+2 -> K :
  + 1 DynamicMovieBox with flags=3 (source-id-present and in-splice), source\_id=1, bundle\_id=0, source\_flags=0x800000 (or 0x800001 as a hint)
  + T DynamicTrackBox (*splice\_track\_IDs*) with one stsd each
    - DynamicTrackHeaderBox with modification\_flags=0x800002
* Fragment K+1 :
  + 0 DynamicMovieBox (if same config from moov can be used)

### Efficiency

With this proposal, signaling a new track in simple cases costs STSD plus:

* 12 bytes for DynamicTrackHeaderBox header (cost of the FullBox)
* 17 bytes for audio (no volume), 27 bytes for video with size/layer and without track matrix (63 with matrix)
* 8 bytes for DynamicTrackBox header (cost of the Box)
* 12 bytes for DynamicMovieBox header

Overhead for 1s fragments is 0.392 kbps for audio, 0.47 kbps for video (0.76 kbps if full visual matrix is specified) hence roughly 1.3 kbps for full signaling (i.e. track and sample descriptions (stsd)) versus 4+ kbps for TrackBox injection as proposed in m58085.

Signaling removal of a track costs 12+4+8+12 bytes, 0.288 kbps

### Usage with dependent movie fragments (defined in section15.1)

The proposed definition for “dependent movie fragments” can be updated when combined with dynamic movie fragment as follows:

If version is not 0, any SampleGroupDescriptionBox, DynamicMovieBox or MetaBox defined in the last movie fragment, or in the last TrackFragment in the last movie fragment, whose MovieFragmentHeaderBox version is 0 also apply for this movie fragment, and there shall not be any SampleGroupDescriptionBox, DynamicMovieBox or MetaBox defined for this movie fragment.

# Tracks in Multiple files

This section documents approaches under investigation for handling multiple ISOBMFF files (with one or more tracks) as a unified presentation.

## External Tracks

Discussion : <http://mpeg.expert/software/MPEG/Systems/FileFormat/isobmff/-/issues/217>

Currently, an ISO Base Media file either holds the sample data within the file containing the structure data (‘trak’ or ‘traf’ box), or refers to the sample data through URL/URNs (as indicated in the DataReferenceBox). It cannot however refer to complete tracks (sample data and structure-data) in other files. If a track from one ISO Base Media file has to be referenced in another file, relying on data reference causes the structure data (track box hierarchy) to be duplicated which is:

* Memory/storage costly
* not efficient as any minor changes to the referenced file(s) shall be reflected in the referring file due to file offset changes; such offset changes may happen quite frequently in the referred file (media editing, adding a new brand, re-interleaving, fragmenting, …).

We can see several use cases for adding tracks by reference in a file:

- add a new language track (audio or subtitles) to a file without having to rewrite the video part

- simpler authoring of track groups and meta data without having to rewrite the entire file

- build a presentation from already authored (encoder output, CMAF recording) track files.

- possibility to create generic files (templates) referring to media tracks

- DASH content preparation: a single ‘moov’ can be constructed with all proper information (kind, language, dependencies, …) from external files at very low cost

Note that last point relates to the on-going exploration on DASH/CMAF/FF alignment**Error! Reference source not found.**.

Having external tracks can therefore benefit authoring, content preparation and even content delivery.

The input contribution m63099 proposes to identify tracks in files using UUID to allow a file processor identifying tracks that are part of the same presentation, but the proposal does not cover how track files are located, how dependencies across tracks are set, how track groups can be setup. It moreover requires a “controlled” environment for the generation of UUIDs, which can be tricky if files come from different providers.

**Proposal**

We propose to allow files to use external tracks in the MovieBox. Possible specification text follows.

Add new section in track structure (section 8.3)

8.3.7 External Tracks

8.3.7.1 External Track Box

8.3.7.1.1 Definition

BoxType: 'extk'   
Container: ‘moov’   
Mandatory: No  
Yes Quantity: zero or more

An ExternalTrackBox can be used to include a track from another ISO Base Media file, as defined by its TrackBox and other track-related structures. The track being referred to is called an external track. The file containing the ExternalTrackBox is hereafter called the referring file, and the file containing the external track is called the referred file. Referred files shall be ISOBMFF compliant files.

External tracks may be fragmented or not, independently of whether the referring file is fragmented or not. Derived specifications may further restrict possible combinations.

The timeline of an external track may be modified by an edit list in the referring file.

The UserDataBox and MetaBox of an external track can be overridden or augmented. UserDataBox present at movie level or MetaBox present at file or movie level in the referred files shall be ignored, and only UserDataBox present at movie level or MetaBox present at file or movie level, if any, of the referring file shall apply.

Track references and track groups of the referred files are ignored and only track references and groups (track groups or entity groups) defined in the referring file are valid.

The track\_ID of the TrackHeaderBox present in ExternalTrackBox gives the identifier of the track in the referring file and can be used to describe track references, track groups and other track relationships relying on track identifiers within the referring file. This allows defining track relations or track groups independently from the identifiers used in the referred file(s).

Additionally, the TrackHeaderBox provides the presentation information of the external track within the presentation of the referring file, such as track width/height, matrix, volumes and track flags.

The following restrictions are set on the TrackHeaderBox in the ExternalTrackBox of an external track:

* If the duration field is undefined (all 1s) and there is no edit list for this track, then the duration of the track is the duration of the referenced track.

The sample description, offsets, sizes… for an external track are the ones declared in the referred file indicated by location in the ExternalTrackLocationBox.

8.3.7.1.2 Syntax

aligned(8) class ExternalTrackBox extends Box('extk'){

ExternalTrackLocationBox extl; //shall be first

TrackHeaderBox tkhd; //shall be second

Box other[]; //any valid children of ‘trak’ except ‘mdia’ and ‘tkhd’, or ExtendedLanguageBox

}

8.3.7.1.3 Semantics

extl indicates the location of the external track. It shall be present and shall be the first box occurring in ExternalTrackBox

tkhd indicates the track header of the external track within the context of the referring file. It shall be present and shall be the second box occurring in ExternalTrackBox.

other indicate any possible box allowed as child of TrackBox except MediaBox and TrackHeaderBox, or ExtendedLanguageBox. Further restrictions on the type of children allowed in ExternalTrackBox are given by the ExternalTrackLocationBox.

8.3.7.2 External Track Location Box

8.3.7.2.1 Definition

BoxType: 'extl'  
Container: ‘extk’   
Mandatory: Yes   
Quantity: one

The ExternalTrackLocationBox allows identifying an external track by its track\_ID in a referred file.

The following values are defined for the flags field of the ExternalTrackLocationBox:

* EXTERNAL\_TRACK\_EDTS\_SKIP (flag mask is 0x000001): shall be set if any edit list present in the external track shall be ignored. If an edit list is present in the container for the edit lists in the ExternalTrackBox for this track, flag shall be set and any edit list present in the external track shall be ignored. Otherwise, (not set) there shall be no EditBox in the ExternalTrackBox, and the edit lists of the external track apply.
* EXTERNAL\_TRACK\_URN (flag mask is 0x000002): if this flag is set, it indicates that the location field is a URN string, otherwise (not set) the location string is a URL,
* EXTERNAL\_TRACK\_UDTA\_IGNORE (flag mask is 0x000004): if this flag is set, this indicates that any UserDataBox defined in the ‘trak’ box of the external track shall be ignored. Otherwise (not set), UserDataBox present in the referring track completes UserDataBox information of the external track. The resulting user data consists in the union of the user data declared in the different UserDataBoxes.
* EXTERNAL\_TRACK\_META\_IGNORE (flag mask is 0x000008): if this flag is set, this indicates that any MetaBox defined in the ‘trak’ box of the external track shall be ignored. Otherwise (not set), MetaBox present in the referring track completes MetaBox information of the external track. The resulting meta data consists in the union of the meta data declared in the different MetaBoxes.

If the indicated location is a URL, it can be an absolute or a relative URL, and the located resource shall be a compliant ISOBMF file. Relative URLs are relative to the file that contains this location.

When EXTERNAL\_TRACK\_EDTS\_SKIP is set and no edit list is present in the ExternalTrackBox, this implies that any edit present in the referred track is ignored and no edit is applied to the track.

If edits from the referred track are used, file readers may need to remap the edit list durations from the timescale of the referred movie to the timescale of the referring movie, if these timescales differ.

8.3.7.2.2 Syntax

class ExternalTrackLocationBox extends FullBox ('extl', version=0, flags)  
{  
 unsigned int(32) referenced\_track\_ID;  
 unsigned int(32) referenced\_handler\_type;   
 unsigned int(32) media\_timescale;   
 utf8string location;   
}

8.3.7.2.3 Semantics

referenced\_track\_ID indicates the identifier (track\_ID) of the external track in the referred file. If value is 0, this indicates that the referenced track is the first TrackBox present in the MovieBox of the referred file. The external track shall be declared through a TrackBox, i.e. recursively referencing external track is forbidden. The external track can use external data references or not; this can be constrained by derived specifications.

referenced\_handler\_type indicates the handler (media) type of the track, and shall be equal to the handler type of the external track in the referred file.

media\_timescale indicates the timescale used to express edit list contained in this external track. Value may be 0 when no edit list is declared in the ExternalTrackBox, or a different value expressing a preferred timescale in case of future insertion of an edit list. Otherwise (an edit list is declared in the ExternalTrackBox), value shall not be 0.

NOTE: this value may be different from the timescale in the MediaHeader in the external track.

location indicates the location of the referred file as a URN or URN, depending on the flags EXTERNAL\_TRACK\_URN.

8.3.7.3 External Track Processing Model

A file reader processes an external track as follows:

* Identify whether the referring file can be processed (brands, track handler types): this follows the same process as for files with no external tracks
* Identify whether it should take the track into consideration: this follows the same rules as for regular tracks, e.g. looking at user preferences, groups, etc …
* If an external track is selected for processing, the referred file is loaded. The external track is marked as invalid if any of the following is true:
  + the location described is invalid
  + the file and/or track cannot be processed by the reader due to brand requirements in ftyp or ttyp
  + the TrackBox corresponding to the external track cannot be found in the referred file,
  + the external track handler type does not match the handler type in ExternalTrackLocationBox
  + the ExternalTrackBox contains a TrackTypeBox with unsupported brands

If an external track is invalid, file readers may reject the file or present only a subset of the external tracks that are valid, as they would usually do for files with no external tracks,

* Otherwise (external track is valid), the processing of the external track is equivalent to processing the track using a second file reader, but using track groups and references defined in the referring file; this implies that global movie structure of the referred file, such as trex, pssh… may be required to process the external file.

‘Meta’ at file or moov level and ‘udta’ at moov level in the refered file(s) shall be ignored.

**Questions/Remarks**

Could the location be designed in such a way that changing its value would not require size changes of the moov ?

## Track Identifiers

Discussion :

<https://mpeg.expert/software/MPEG/Systems/FileFormat/isobmff/-/issues/187>

<http://mpeg.expert/software/MPEG/Systems/FileFormat/isobmff/-/issues/220>

ISOBMFF specifies encapsulation of media streams into tracks of a file. It typically assumes that all tracks of a media presentation are contained in a single file with a single movie box that documents all tracks in the file. However, some of the ISOBMFF principles are not carried forward to CMAF or DASH, because they rely on the concept of late binding, i.e., each track stored in a separate file.

In case of late binding

1. The requirement to have unique track identifiers is not carried forward.
2. Each file includes its own movie header and relationship about the tracks in different files cannot be expressed.

Without the uniqueness established between tracks (media streams), a file reader will have to deal with ambiguity in selecting the tracks for playback, and this may lead to unintended behaviour at the playback end.

In this contribution we propose to address the above concerns by introducing new MovieHeaderExtensionBox which is a companion box to the MovieHeaderBox and carries one or more unique identifiers. When two or more ISOBMFF files carry the same unique identifiers, then the files represent the same presentation and can be combined to get a conformant ISOBMFF file. Moreover, CMAF file/segment writers and MPD generators can verify based on the unique identifiers which files represent the same presentation to prepare them for the same CMAF switching set and DASH adaptation set.

**Proposal**

* 1. **Defining Movie Presentation Identifier Box**

**XX Movie presentation identifier box**

**XX.1 Definition**

Box Type: 'mpid'  
Container: MovieBox  
Mandatory: No  
Quantity: Exactly one

The MoviePresentationIdentifierBox is a box within the MovieBox. This box carries unique identifiers for the movie in the current ISOBMFF file.

The following values are defined for the flags field of the MoviePresentationIdentifierBox:

TRACK\_MERGE\_PROCESS (flag mask is 0x000001): if this flag is set, and if the track IDs overlap in files having the same presentation\_ID, then the non-overlapping samples in decoding time and respective track metadata shall be selected from any of these tracks. The selected samples and metadata are in a manner that there shall be a sync sample when a switch to another of these tracks takes place. When the track IDs differ in the files having the same presentation\_ID, the tracks can be combined to a single file. Otherwise (when TRACK\_MERGE\_PROCESS is not set), no merging process is specified.

NOTE When tracks in different files do not carry a presentation\_ID, there is no guarantee that combining those files lead to a conforming ISOBMFF file.

The presentation\_ID present in the MoviePresentationIdentifierBox can be used for determining that different files are part of the same presentation. If a track belongs to two or more different presentations, multiple presentation\_IDs are present in the MoviePresentationIdentifierBox which can be then mapped to different presentations.

**XX.2 Syntax**

aligned(8) class MoviePresentationIdentifierBox extends FullBox('mpid', version, flags) {  
 unsigned int(8) presentation\_ids\_count\_minus1;  
 unsigned int(128) presentation\_ID[unique\_ids\_count\_minus1 + 1];  
}

**XX.3 Semantics**

presentation\_ids\_count\_minus1 plus 1 indicates the number of unique IDs present in the movie.

presentation\_ID is an array of unique values for the movie in the current file. Array elements of presentation\_ID can take integer values (UUID) as defined in RFC 4122.

**Questions/Remarks**

* The identification and the merging rules are orthogonal (sometimes no specific merging rule is needed), and should be described by independent tools

# Extensions for sample auxiliary information

ISOBMFF includes an ability to attach sample auxiliary information (SAI) to each sample in a track. SAI is a generalized capability, enabling the attachment of generic forms of information to samples in a track. Shortcomings of the current approach addressed in this contribution include:

1. A limit of 255 bytes of SAI for each sample.
2. No defined mechanism for specifically signaling an encoding method for the SAI. The default method for using the aux\_info\_type fourCC, enables signaling a type, but not a specific implementation instance of certain common encoding methods, such as SMPTE key-length-value.
3. No defined mechanism for including static information related to a specific set of SAI.

An example of encryption was discussed in a prior contribution as a justification for additional space. Other potential use cases involve pixel level metadata, where SAI can exceed the size of an uncompressed image. As a result, options for 16-bit and 32-bit size information are included in this proposal. Finally, as per the prior proposal, best practices for syntax definitions related to versioning are included in this proposal.

**Proposal**

aligned(8) class SampleAuxiliaryInformationSizesBox extends FullBox('saiz', version, flags)

{

if (flags & 1) {

unsigned int(32) aux\_info\_type;

unsigned int(32) aux\_info\_type\_parameter;

}

unsigned int sample\_size\_bits=0;

if (version==2) {

sample\_size\_bits = 32;

} else if (version==1) {

sample\_size\_bits = 16;

} else if (version==0) {

sample\_size\_bits = 8;

}

unsigned int(sample\_size\_bits ) default\_sample\_info\_size;

unsigned int(32) sample\_count;

if (default\_sample\_info\_size == 0) {

unsigned int(sample\_size\_bits) sample\_info\_size[sample\_count];

}

}

No changes on semantics

8.7.10 **Sample auxiliary information configuration box**

8.7.10.1 **Definition**

BoxType: 'saiC'  
Container: SampleTableBox or TrackFragmentBox  
Mandatory: No  
Quantity: Zero or More

The SampleAuxiliaryInformationConfigBox allows associating private information to all per-sample sample auxiliary information for a given tuple {aux\_info\_type, aux\_info\_type\_parameter}.

It allows for defining an encoding type, which when enabled, applies to the sample auxiliary information and any associated private information.

When SampleAuxiliaryInformationConfigBox is present in a track fragment, the private information applies to all sample auxiliary information for the identified type in that fragment; otherwise (not present in a fragment or file not fragmented), the private information of the SampleAuxiliaryInformationConfigBox present in the track SampleTableBox applies to the sample of the track fragment or of the track.

8.7.10.2 **Syntax**

aligned(8) class SampleAuxiliaryInformationConfigBox extends FullBox('saiC', version=0, flags=0)

{

unsigned int(32) aux\_info\_type;

unsigned int(32) aux\_info\_type\_parameter;

if ( flags & 1){

unsigned int(32) aux\_info\_enc\_type;

if (aux\_info\_enc\_type == 'mime') {

utf8string content\_type;

}

else if (aux\_info\_enc\_type == 'uri ') {

utf8string encoding\_uri\_type;

}

utf8string content\_encoding; //optional

}

bit(8) data[]; //till end of box

}

8.7.10.3 Semantics

(flags & 1) equal to 1 specifies the sample auxiliary information uses a ‘mime’ or ‘uri ‘ defined encoding method. (flags & 1) equal to 0 indicates the encoding of the sample auxiliary information is determined by association with the aux\_info\_type and aux\_info\_type\_parameter values.

aux\_info\_type and aux\_info\_type\_parameter are defined as in SampleAuxiliaryInformationSizesBox

aux\_info\_enc\_type is a 32-bit fourCC value defining a valid sample type indicator, either 'mime' or ‘uri ‘.

content\_type is the MIME type for the SAI. If the samples are content encoded (see below), then the content type refers to the SAI after content decoding.

encoding\_uri\_type is an absolute URI, indicating the method for encoding the SAI. If the samples are content encoded (see below), then the encoding URI type refers to the SAI after content decoding.

content\_encoding optionally indicates the SAI is encoded and needs to be decoded before interpreted. The values are as defined for Content-Encoding for HTTP/1.1. Some possible values are “gzip”, “compress” and “deflate”. An empty string indicates no content encoding. Note the SAI is stored after the content encoding has been applied.

data contains private information for sample auxiliary information associated with a specific aux\_info\_type and aux\_info\_type\_parameter.

# On media rating

Discussion  
<http://mpeg.expert/software/MPEG/Systems/FileFormat/isobmff/-/issues/219>

**Description**

Various media transport signaling specifications include metadata structures to indicate that the media content is suitable for presentation to audiences for which that rating is known to be appropriate. Otherwise, the presentation of the media is expected to be blocked by the playback device, a behavior which is sometimes enforced by authorities.

MPEG-DASH (ISO/IEC 23009-1, clause 5.8.4.4) or the DVB version of MPEG-2 TransportStreams (ETSI EN 300 468, clause 6.2.28) are examples of specifications that include appropriate data structure definitions. In contrast, ISO/IEC 14496-12 lacks an appropriate definition yet.

The lack of support for carriage of rating information in ISOBMFF was not yet an issue since playback devices received the media packaged into a transport format capable of carriage of this information (such as the beforementioned MPEG-2 Transport Stream). And also with applications using ISOBMFF, the playback device had also access to the streaming manifest (an example therefore is the earlier version of HbbTV, where the DASH MPD is provided to the device via MSE/HTML video element.).

However, given the recent tendency of streaming manifest being evaluated by ‘externally provided’ applications (for example a downloaded JavaScript application such as dash.js), the device implementation just receives (segmented) ISOBMFF files, with having no longer access to the rating information and consequently losing the opportunity to control whether the media content is suitable for presentation to the audience. This is a crucial situation since device manufactures can’t reliably fulfill their legal obligations.

**Questions/Remarks**

- Shouldn’t rating be handled as a form of tagging ?

- It would be good to investigate if “standard” tagging is needed, given the lack of interoperability and more importantly the lack of specification for tags in isobmf

- what should be the granularity/scope of a rating information: at the file level, at the movie level, at the track or sub-track level, does it have a temporal granularity (tagging only some parts of the timeline) ?

- This could be expressed through kindbox as well

# Undetermined mdat size

Contribution [m65338](https://dms.mpeg.expert/doc_end_user/documents/144_Hannover/wg11/m65338-v1-m65338.zip), gitlab <https://mpeg.expert/software/MPEG/Systems/FileFormat/isobmff/-/issues/245>

Use case: write (capture mode, mdat first) in non-fragmented mode on a non-seekable media.

X Data Termination Indication Box

X.1 Definition

Box Type: 'dtib'  
Container: file  
Mandatory: No  
Quantity: Zero or one

The DataTerminationIndicationBox provides the position of the first byte immediately following the last byte of a Media Data Box with undetermined size. The position is indicated from the end of the file.

The DataTerminationIndicationBox shall only be present when a media data box uses a size of 0. When present, it shall be at the end of the file with no other box following it.

A reader can locate this fixed-size box by offsetting from 12 bytes from the end of file. If these 12 bytes correspond to a DataTerminationIndicationBox, reader can then read the value of the data termination offset. From a byte position corresponding to (*fileSize - data\_termination\_offset -12)*, reader may identify zero or more top-level boxes between this byte position and the DataTerminationIndicationBox.

When this box is found, replacing size of the 0-sized top-level box to the indicated size, without using large-size escape mechanism if the indicated size is greater than 32 bits, shall result in a compliant ISOBMFF file.

X.2 Syntax

class DataTerminationIndicationBox() extends Box('dtib') {  
 unsigned int(32) data\_termination\_offset;  
}

X.3 Semantics

data\_termination\_offset indicates the number of bytes between the last byte of the preceding data box with size=0 and the first byte of this box. It is computed as follows: position of the 1st byte of ‘dtib’ - (1 + position of last byte of media data box)

# Condensed file type box

Contribution [m64695](https://dms.mpeg.expert/doc_end_user/current_document.php?id=89316) – gitlab <https://mpeg.expert/software/MPEG/Systems/FileFormat/isobmff/-/issues/238>

Proposal

in 4.3.1, in the third paragraph, add “the major\_brand” to the “specifications to which the file complies” so that it reads “a set of specifications to which the file complies (the major\_brand and the compatible\_brands)”

in 4.3.1, add the following before paragraph “The minor version is informative only…”:

“If only a single brand needs to be signaled, the compatible brands list may be empty.”

in 4.3.1, add after the paragraph “The minor version is informative only…”:

“The interpretation of the minor version is major-brand specific. The semantics of the 32 bits of the minor\_version field may be re-defined by the specification defining the major brand value, for example using these 32 bits as flags.”

# MetaBox compatibility with QT

Contribution m65602, gitlab <https://mpeg.expert/software/MPEG/Systems/FileFormat/isobmff/-/issues/257>

Proposal:

At the end of 8.11.1 add the following

NOTE: other specifications based on the box structure defined in this document use a Box instead of a FullBox for ‘meta’. Readers supporting several of these specifications need to be careful when parsing this box.

Consider rewriting the syntax from:

aligned(8) class MetaBox (handler\_type)

extends FullBox('meta', version = 0, 0) {

to

aligned(8) class MetaBox (handler\_type)

extends Box('meta') {

bit(32) reserved = 0;

with a note in semantics:

NOTE: future version of this specification will only use value 0, 2, 4, 6 for the reserved field

# Update to the Segment Index Box

## reference\_count as 32 bits

### Justification

Much of the professional content created today is longer than 18.2 hours, especially archives of CMAF content produced in live OTT setups or used as archiving for television content.

### Proposal

Alter the definition of the segment index box as follows (change highlighted)

|  |
| --- |
| 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;  }  if (version ==0) {  unsigned int(16) reserved = 0;  unsigned int(16) reference\_count  }  else {  unsigned int(32) 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;  } } |

# Signaling poisoned data for AI training

## Rationale

AI models and inference engines used in computer vision tasks make use of images/video frames for training. However, in certain scenarios the content provider or the subject in the scene do not wish their content to be part of such AI applications (for example protecting the privacy of a person without blurring the face in photos/videos). One other reason could be copyright infringement, so that the content owners would not want their media to be used for training AI models.

Number of algorithms that allows to create ’poisoned data’ for making an image unusable for AI models, but at the same time viewable and consumable by an end user, is growing.

For example:

* https://towardsdatascience.com/how-nightshade-works-b1ae14ae76c3
* https://nightshade.cs.uchicago.edu/faq.html
* https://arxiv.org/abs/2310.13828

The algorithms can be used by content creators to protect their work from being used by the AI training systems and models. At the same time, it can be assumed that those content creators do not have bad intentions. However, the images with the additional ’poisoned data’ can be harmful for the AI models and significantly reduce their accuracy and performance. Especially when an AI model is used for a mission-critical task, such as detecting pedestrians on the road or detecting certain objects in an area, avoiding AI poisoned media content for AI training is a must.

It is required for an AI expert or system to know that the image/video dataset that he/she uses does not contain such AI-poisoned image/video samples.

## Proposed approach

One way to address the issue is to define a dedicated AIUsabilityBox box describing AI related information.

class AIUsabilityBox extends Box('aiuc') {  
 unsigned int(1) valid\_for\_training;  
 unsigned int(7) reserved;  
}

valid\_for\_training equal to 1 indicates that the item/track can be used for training AI models. When this value is 0, it indicates that an AI-poisoning algorithm might be applied on the item/track which makes it unusable (even harmful) for the AI training process.

Such box could be:

* an item property and can be attached to an image item it describes.
* contain on track level, e.g., as a child box of TrackBox

# Empty Track References

## Question #1: How many elements can be in track\_IDs[]?

The ISOBMFF specification is silent about the minimum number of elements of array track\_IDs[].

track\_IDsis an array of integers providing the track identifiers of the referenced tracks or track\_group\_id values of the referenced track groups. Each value track\_IDs[i], where i is a valid index to the track\_IDs[] array,is an integer that provides a reference from the containing track to the track with track\_ID equal to track\_IDs[i] or to the track group with both track\_group\_id equal to track\_IDs[i] and (flags & 1) of TrackGroupTypeBox equal to 1. When a track\_group\_id value is referenced, the track reference applies to each track of the referenced track group individually unless stated otherwise in the semantics of particular track reference types. The value 0 shall not be present. In the array there shall be no duplicated value; however, a track\_ID may appear in the array and also be a member of one or more track groups for which the track\_group\_IDs appear in the array. This means that in forming the list of tracks, after replacing track\_group\_IDs by the track\_IDs of the tracks in those groups, there might be duplicate track\_IDs. A track\_group\_ID shall not be used when the semantics of the reference requires that the reference be to a single track.

Regarding reference software, it seems the implementation does not specifically expect to have no element in the array. Here is for instance the serialization code of the box:

bytesToRead = s->size - s->bytesRead;

self->trackIDCount = bytesToRead / sizeof(u32);

self->trackIDs = (u32 \*)calloc(1, bytesToRead);

TESTMALLOC(self->trackIDs)

for(i = 0; i < self->trackIDCount; i++)

{

GET32(trackIDs[i]);

}

In the code above, calling calloc with 0 byte and using the return pointer may cause issue since this behaviour is implementation specific.

## Question #2: Value 0 as element of Track\_IDs

As specified in ISOBMFF, the value 0 is forbidden “The value 0 shall not be present”. Therefore, this value could not be used if a file should keep a Track Reference box, for reasons that is use case specific, but not have a specific value to write, for example the exact track to refer to is not known or in a different file.

## Question #3: When tracks are split in different files (e.g. CMAF)

First of all, the CMAF specification is silent on the usage of the Track Reference box (tref). Assuming that an auxiliary track is used in the context of CMAF, it follows that the value in the track IDs array of the Track Reference Type box is unclear, since only one track would be present in the file. If no track IDs is signaled, assuming this is allowed by ISOBMFF and not breaking parsers, the Track Reference box may be kept.

The current assumption is that the question of track references would be handled at higher level than the file, i.e. in the manifest file. At the very least, there should be some guidance on how to transform the track references to CMAF, especially if the Track Reference box should be removed, and if kept, what values to put in the Track\_IDs array.

# Configuration for sample auxiliary information

ISOBMFF includes the ability to carry sample auxiliary information for each sample in a track. The existing method for signaling the type of information is based on two unregistered 4CC codes (aux\_info\_type and aux\_info\_type\_parameter). With a growing interest in using sample auxiliary information for a variety of use cases, it is recognized a more robust approach is needed to achieve desired levels of functionality and interoperability. The two issues listed above (m67816 and m67971) propose different methods for different use cases related to this topic. It is desired to study and explore the creation of a single, generalized approach for the configuration of sample auxiliary information to serve all use cases and applications utilizing SAI.

Related Issues:

1. **m67816 [NALuFF][TuC] Update to multiple layers in single-layer track design**

<https://git.mpeg.expert/MPEG/Systems/FileFormat/NALuFF/-/issues/194>

1. **m67971 [ISOBMFF] On methods for encoding sample auxiliary information**

<https://git.mpeg.expert/MPEG/Systems/FileFormat/isobmff/-/issues/289>

**Use cases**

1. Enabling multiple layers in a single-layer track in NALuFF.
2. Configuration and carriage of Key-Length-Value (KLV) encoded SAI (and other defined forms of encoding metadata)
3. Carriage of 2D grid metadata, mapped to the sample data. For instance, 1 or 8-bit cloud cover data in a geospatial image sequences, or pixel level metadata.
4. Carriage of numerically lossless compressed SAI.

**Explorations**

1. Explore/define a uniform solution for signaling configuration information (including encoding methods and layer configuration) for sample auxiliary information. Methods shall include registered methods, such as through a 4CC, as well as general methods based on MIME and URI techniques, such as found in the Metabox.
2. Explore/define methods for carrying and associating static information to all per-sample SAI for a given tuple {aux\_info\_type, aux\_info\_type\_parameter}.
3. Explore/define the location to store SAI configuration information
4. Explore/define the structure for how SAI configuration is to be carried
5. Explore/define options to handle configuration data associated with the SAI which is not static. For instance, using sample groups to update the configuration data.
6. Explore/define options for encrypting SAI payloads
7. Explore/define options for configuring the compression of SAI payloads (see 23001-17 Amd2)
8. Explore/define how to signal the presence of additional layers in the MIME type