Text

Description automatically generatedISO/IEC JTC 1/SC 29/WG 03 N0598

**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:** 2022-07-25

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

**No. of pages:** 72 (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 N0598**

**July 2022, Virtual**

|  |  |
| --- | --- |
| **Title** | **Technologies under Consideration for ISO/IEC 14496-12** |
| **Source** | **WG 03, MPEG Systems** |
| **Status** | **Approved** |
| **Serial Number** | **21727** |

**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](#_Toc109893736)

[2 Edit lists and movie fragments 3](#_Toc109893737)

[3 Multiplexed timed metadata tracks 6](#_Toc109893738)

[4 Embedded Metadata Signaling 11](#_Toc109893739)

[5 On MIME type parameters 12](#_Toc109893740)

[6 On relation of entity groups and sample groups 13](#_Toc109893741)

[7 Stronger defaulting in Track Runs 14](#_Toc109893742)

[8 Sample reordering in Track Runs 22](#_Toc109893743)

[9 Segment Index and Level Assignment 26](#_Toc109893745)

[10 Generic sub-picture track grouping extensions 33](#_Toc109893747)

[11 Signaling of Multi-Layer Picture Compositing Information in the VVC File Format 37](#_Toc109893748)

[12 Carriage of T.35 messages 43](#_Toc109893749)

[13 Integrating new codecs 46](#_Toc109893750)

[14 SubSampleInformationBox update 48](#_Toc109893751)

[15 MovieFragmentHeaderBox update 52](#_Toc109893752)

[16 Sample Run Sample Group 53](#_Toc109893753)

[17 FrameRateBox 55](#_Toc109893754)

# Handling lost or corrupted samples using a sample group for corrupted samples

Issues for the original contributions: [m57362](http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/66) and [m58084](http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/74)

*Define a new sample group ‘corr’*

class CorruptedSampleInfoEntry()  
extends SampleGroupDescriptionEntry ('corr')  
{  
 bit(2) corrupted;  
 bit(6) reserved;  
 if (corrupted==2)  
 bit(32) codec\_specific\_param;  
}

corrupted indicates the corruption state of the associated data. Value 0 means that the entire data is lost, and the associated data size (sample size, or NAL size) shall be 0. Value 1 means that the data is corrupted without any additional information on the corruption. Value 2 means that the data is corrupted with codec specific information on the corruption. Value 3 is reserved.

codec\_specific\_param indicates codec specific information on the corruption. The coding format is the one of the sample associated to this sample group description.

Note: codec\_specific\_param information being dependent on the coding format, file writers may need to add and associate a different CorruptedSampleInfoEntry() entry with a sample each time the coding format is changing across samples.

If a data is not associated with a CorruptedSampleInfoEntry or if a data is associated with a description\_group\_index = 0 by a sample group with the grouping\_type 'corr', this means the data is not corrupted.

The processing of a sample with corrupted equal to 1 or 2 is context and implementation specific.

For *NALUFF, state:*

For NALU based codecs, we propose the following semantics:

For NALU-based video formats, the codec\_specific\_param field of the CorruptedSampleInfoEntry is defined as a bit mask, with most significant bit first, of the following flags:

* ParameterSetCorruptedFlag (value 0x00000001): indicates that one or more parameter sets (DCI, VPS, SPS, PPS, APS, OPI?) in the associated data is corrupted
* SEICorruptedFlag (value 0x00000002): indicates that one or more SEI messages in the associated data is corrupted
* SliceHeaderCorruptedFlag (value 0x00000004): indicates that one or more slice headers or picture headers in the associated data is corrupted
* VCLCorruptedFlag (value 0x00000008): indicates that VCL data of one or more slices in the associated data is corrupted
* OtherNonVCLNALCorruptedFlag (value 0x00000010): indicates that one or more NAL units in the associated data with types different from the above types is corrupted

A codec\_specific\_param with value 0 means no information is available for describing the corruption.

A CorruptedSampleInfoEntry may be used with a sample group of grouping\_type 'nalm' and a NALUMapEntry, using the grouping\_type\_parameter 'corr'. The groupID of the NALUMapEntry map entry indicates the index, starting from 1, in the sample group description of the CorruptedSampleInfoEntry. A groupID of 0 indicates that no entry is associated (the identified data is present and not corrupted).

[[Ed Note on ParameterSetCorruptedFlag : do we want to split per NAL unit type ?]]

# Edit lists and movie fragments

## Potential use cases

* Documenting the equivalent of the edit list but in fragments
* Enabling the fragmentation in the middle of a sample.

## Edit Adjustment

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

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

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

Derive the following two values:

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

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

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

## Original Duration

### Introduction

Considering the discussion in [m58087](http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/77) and its issue, this proposes a way to signal:

* The original intended duration of a sample for player to know where to stop on the sample
* the "past" or elapsed duration of a cloned sample for player to know where to start rendering the content of the cloned sample
* without impacting tfdt or sample duration

[[ed Note: we propose here a new box for the sake of backward compatibility, but this feature could be directly added to ‘tfdt’ for compacity reasons, saving one full box header, as outlined in the contribution.]]  
[[ed: VTT has a more general technique that allows arbitrary sample splitting, but that is specific to VTT. We could maybe make a general technique here for splitting samples other than only at Fragment boundaries?]]

### Proposal

X Fragmented Sample Time Adjustment

X.1 Definition

Box Type: 'fsta'  
Container: TrackFragmentBox   
Mandatory: No  
Quantity: Zero or one

The FragmentedSampleTimeAdjustmentBox provides:

* the elapsed time, measured in media timescale, of the first sample in decoding order in the track fragment.
* the original duration, measured in media timescale, of the last sample in decoding order in the track fragment.

This can be useful to document that the first sample of the track fragment is a copy of the previous sample, if any, and that this sample original start time was intended to be before its actual sample decode time. This allows rewinding the sample playback time at tune-in but ignoring it in regular playback mode.

This can also be useful to document that the last sample duration was truncated to respect fragmentation constraints, and that the intended duration of the sample is longer than its actual duration in the fragment; this allows exact processing of samples with internal timing logic dependent on the sample duration (such as text animations).

The following flags are defined:

* FSTA\_ORIGINAL\_DURATION: flag value is 0x000001. If set, the box describes the original duration of the last sample in the track fragment
* FSTA\_ELAPSED\_DURATION: flag value is 0x000002. If set, the box describes the elapsed duration of the first sample in the track fragment

When this box is present and flag FSTA\_ORIGINAL\_DURATION is set, it indicates that the last sample of the track fragment has a shorter duration than originally authored, and the original duration is signaled. The originalDuration shall be equal to or greater than the duration of the last sample in this track fragment.

When this box is present and flag FSTA\_ELAPSED\_DURATION is set, the first sample of the track fragment is treated as if its associated sample\_flags value has sample\_depends\_on=2 and sample\_has\_redundancy=1.

If a previous sample was already received for this track, that previous sample duration is extended by the duration of this first sample and the elapsedDuration is ignored. If the previous sample had an originalDuration signaled, the extended duration shall be:

* less than originalDuration if this is the only sample of the track fragment and it has an originalDuration associated
* equal to originalDuration otherwise

If no previous sample were received for this track (tune in), the first sample is processed at its sample decode time as if it was being presented for the indicated elapsedDuration.

Note: FSTA\_ORIGINAL\_DURATION and FSTA\_ELAPSED\_DURATION may be set together in a track fragment with multiple samples (describing that first sample is a continuation and last sample is truncated) or with a single sample (describing the only sample is both a continuation and truncated, for example when splitting a sample of 20 second long into three or more fragments).

X.2 Syntax

aligned(8) class FragmentedSampleTimeAdjustmentBox extends FullBox('fsta', 0, flags)  
{  
 if (flags & FSTA\_ORIGINAL\_DURATION) unsigned int(32) originalDuration;  
 if (flags & FSTA\_ELAPSED\_DURATION) unsigned int(32) elapsedDuration;  
}

X.3 Semantics

originalDuration gives the original duration of the last sample of the track fragment, in media timescale of this sample

elapsedDuration gives the elapsed duration of the first sample of the track fragment, in media timescale of this sample

# 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.

# Embedded Metadata Signaling

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

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

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

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

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

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

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

# On MIME type parameters

## Introduction

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

### Video with display orientation changes

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

### Signalling of HDR/WCG information in ISOBMFF

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

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

### The restricted scheme design in ISOBMFF

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

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

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

## Problems and discussions

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

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

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

## Proposal

The following methods are proposed to solve problems:

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

# On relation of entity groups and sample groups

For tracks, ISOBMFF provides various tools at different levels either to group together tracks or samples. When a new grouping type (represented by a new 4CC) is defined, it is possible that some of the parameters characterizing this grouping type are common to all tracks or a subset of tracks in a group and some others can be changing or be overridden at sample level. Can ISOBMFF provide some guidance on how to combine entity group, track group and sample group (i.e. how they relate to each other) to declare such parameters?

The following paragraph provides guidance for the combination of entity group with sample group and could become part of a new subsection in Annex B ("guidance on deriving from this document"):

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

Similar guidance could be provided for combining a track group with a sample group when a "grouping" feature for a track requires simultaneously some parameters to be static and some other parameters to be changing over groups of samples (as already done for '2dsr' in OMAF for instance).

# 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 : New version of ‘sidx’

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

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

The flags field has the following semantics:

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

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

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

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

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

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

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

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

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

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

## Alternative improvements to the SegmentIndexBox

### Option 1

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

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

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

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

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

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

### Option 2

#### Syntax

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

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

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

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

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

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

## Attributes in DASH MPD

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

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

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

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

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

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

## Example usage of the segment index box

*Add the following clause J.2.5:*

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

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

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

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



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

# Sample reordering in Track Runs

## Introduction

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

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

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

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

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

- Option1: split the temporal layers as one per track

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

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

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

*Example:*

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

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

Our expected layout would be:

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

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

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

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

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

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

## Further discussion

### Context

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

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

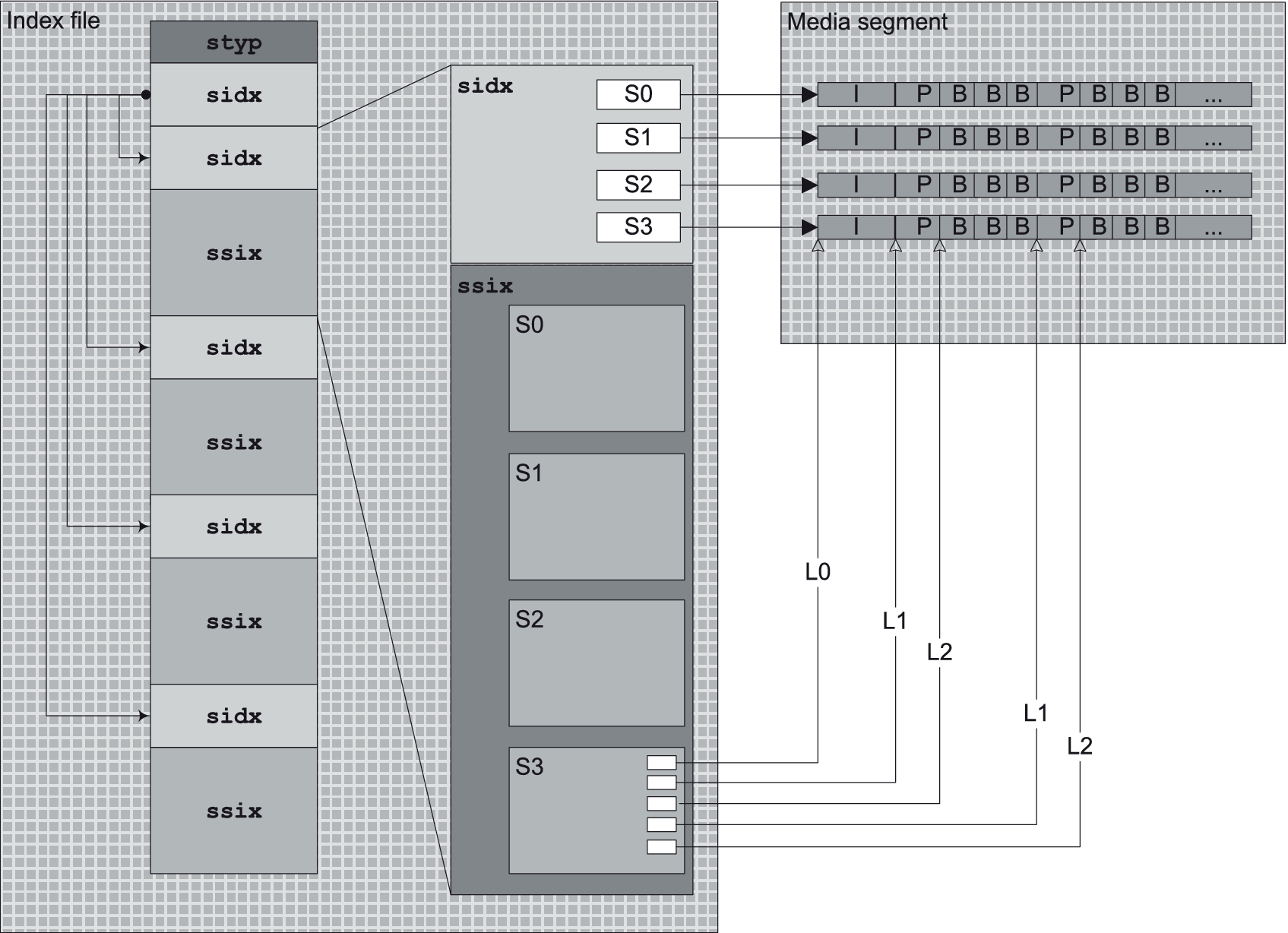


Figure : Mapping temporal sub-layers to Sub-Segment Index Box for trick mode in DASH

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

### leva and ssix clarifications

See the Defect Report for 14496-12.

### Sample reordering using leva and ssix

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

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

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

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

- it seems to break the philosophy of ssix

- multiple byte ranges will be required for a given level

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

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

- Single trun, single track

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

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

- Reusability of partially downloaded segments

- Friendliness for unequal FEC protection schemes

## Proposal

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

### Sample interleave index in (Compact) Trun

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

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

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

We then need:

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

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

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

Add in ctrn syntax section:

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

Add at the end of ctrn box syntax:

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

Add to semantics:

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

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

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

### Sample count in Compact Trun

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

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

In Section 8.8.8.2.2 of AMD4, add the following:

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

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

unsigned int(16) sample\_count;

with

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

# Segment Index and Level Assignment

Issues: *<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>*

See also *<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 - level\_assignment\_type=1 with 2 IDRs and no explicit range for moof

A picture containing shape

Description automatically generated

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

A picture containing text

Description automatically generated

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







# Generic sub-picture track grouping extensions

Issue: *<http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/53>*

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 7.1.11.2 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.

## Two dimensional spatial relationship

### 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.

The SubPictureRegionBox is optional and either:

1. is present in the SpatialRelationship2DDescriptionBox and there shall be no associated SpatialRelationship2DGroupEntry in the associated track (this track has a constant, static, size and position); or
2. is not present in the SpatialRelationship2DDescriptionBox and there shall be one or more associated SpatialRelationship2DGroupEntry(s) in the associated track (this track possibly has a dynamic size and/or position).

When constructing the composition picture as specified below, gaps between sub-pictures carried in different sub-picture tracks are allowed, and overlaps between sub-pictures carried in different sub-picture tracks with different values of the layer field in TrackHeaderBox are allowed. However, overlaps between sub-pictures carried in different sub-picture tracks with the same value of the layer field in TrackHeaderBox are not allowed.

The spatial relationship is restricted according to the chroma sub-sampling format of the associated track; total\_width and total\_height, and object\_x, object\_y, object\_width and object\_height, shall all select an integer number of samples for all planes. In effect this means that:

* when the format is 4:4:4, there is no restriction;
* when the format is 4:2:2 the total\_width, object\_x and object\_width shall be even numbers;
* when the format is 4:2:0 all of these fields shall be even numbers.

The composition picture is reconstructed as follows, with values of object\_x, object\_y, object\_width, and object\_height obtained from SubPictureRegionBox if present or otherwise from the SpatialRelationship2DGroupEntry applying to the sample:

1. Out of all tracks belonging to the same '2dsr' track group, form them into subgroups such that each subgroup contains tracks in the same alternate group; then select exactly one track from each of those subgroups.
2. For each composition-time aligned sample of each of the selected tracks, the following applies, in the front-to-back ordering (layer) indicated in the TrackHeaderBox of the picked tracks:

For each value of i in the range of 0 to object\_width − 1, inclusive, and for each value of j in the range of 0 to object\_height − 1, inclusive, the pixel value of the composition picture at pixel position ( ( i + object\_x ) % total\_width, ( j + object\_y ) % total\_height ) is set equal to the pixel value of the sample of this track at pixel position (i, j).

### 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 SubPictureRegionBox extends FullBox('sprg',0,0) {  
 unsigned int(16) object\_x;  
 unsigned int(16) object\_y;  
 unsigned int(16) object\_width;  
 unsigned int(16) object\_height;  
 bit(14) reserved = 0;  
 unsigned int(1) track\_not\_alone\_flag;  
 unsigned int(1) track\_not\_mergeable\_flag;  
}

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

### Semantics

total\_width specifies, in pixel units, 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, in pixel units, 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.

object\_x specifies the horizontal position of the top-left corner of the samples in this track within the coordinate system specified by this 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 this SpatialRelationship2DDescriptionBox.

object\_y specifies the vertical position of the top-left corner of the samples in this track within the coordinate system specified by this 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 this SpatialRelationship2DDescriptionBox.

object\_width specifies the width of the samples in this track within the coordinate system specified by this 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 track within the coordinate system specified by this 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.

track\_not\_alone\_flag equal to 1 indicates that the current sub-picture track is not intended to be presented alone without at least one other sub-picture track belonging to the same track group of grouping type '2dsr'. The value 0 indicates that the current sub-picture track may or may not be intended to be presented alone without at least one other sub-picture track belonging to the same track group of grouping type '2dsr'.

track\_not\_mergeable\_flag equal to 1 indicates that the video bitstream carried in the current sub-picture track cannot be merged with the video bitstream carried in any other sub-picture tracks belonging to the same track group of grouping type '2dsr', to generate a single video bitstream without decoding mismatch by rewriting only header data of the bitstreams, where a decoding mismatch refers to the value of any pixel when decoding the video bitstream in the current track is not identical to the value of the same pixel when decoding the merged single video bitstream. An example of such bitstream merging is the reconstruction of an HEVC bitstream as specified in clause 10.1.3.4 when the untransformed sample entry type of the track with the given track\_ID is equal to 'hvc2'. track\_not\_mergeable\_flag equal to 0 indicates that the video bitstream carried in the current sub-picture track can be merged with the video bitstream carried in at least one other sub-picture track belonging to the same track group of grouping type '2dsr' to generate such a single video bitstream in such a manner as described above.

NOTE: When HEVC (i.e., Rec. ITU-T H.265 | ISO/IEC 23008-2) is the video codec used for encoding of the bitstreams carried in the sub-picture tracks, track\_not\_mergeable\_flag equal to 0 means that the HEVC bitstream carried in the current sub-picture track contains and only contains one or more MCTSs that can be indicated by a temporal MCTSs SEI message as specified in HEVC version 5 published by the ITU-T in Feburary 2018, or a later version of HEVC.

## 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.

There are restrictions both on the presence of this sample grouping, and on the values of the fields; see clause 7.1.11.2.1.

### 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;  
}

### 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.

# Signaling of Multi-Layer Picture Compositing Information in the VVC File Format

## Introduction

From the minutes of File-format October 2020: We add to the TuC for ISOBMFF this question and problem area, with this solution as a strawman that is too VVC-specific, with notes etc. on the problems, and welcome further contributions on the subject to develop a generic solution suitable for the ISOBMFF.

VVC file format is being developed as part of the Amendment to ISO/IEC 14496-15 on Carriage of VVC and EVC in ISOBMFF [1].

Multi-layer support in the VVC file format is described in clause 11.3.4. It is desirable to provide a general and flexible solution to multi-layer picture compositing which supports bitstream extraction and merging (BEAM) functionalities, while minimizing bitrate overhead. With this motivation, two SEI messages were proposed for VVC in [2]-[3], which describe recommended composite pictures of decoded pictures from multiple layers. The layer composite SEI message is sent for each layer and contains parameters for the current layer’s decoded picture. The recommended composite layers info SEI message applies to multiple layers, and includes parameters for each output layer set (OLS).

In this contribution, we propose a system-layer alternative to the signaling of recommended multi-layer picture compositing information. In particular, two new entity groups for the VVC file format are proposed for this purpose.

The layer composite position info entity group applies to the decoded picture of the different layers of the VVC bitstream (each layer identified by nuh\_layer) and includes syntax elements to indicate the following for each layer:

* top left vertical and horizontal position of the decoded layer picture within a composite picture, in units
* height and width of the decoded layer picture within a composite picture, in units

The recommended composite layers info entity group signals parameters for a recommended composite pictures for each OLS, which apply to multiple layers of the VVC bitstream and contains syntax elements to indicate the following:

* flag to indicate if scaling of decoded layer pictures in the composite picture is enabled
* flag to indicate if decoded layer pictures may overlap in the composite picture
* number of OLSs
* For each OLS
  + size of the units in luma samples of the position and size syntax elements in the layer composite position info entity group
  + size of the composite picture
  + offset to be applied to all decoded layer pictures in the composite picture

The size of each layer’s decoded picture is not required to equal its recommended display size in the composite picture, unless the flag indicates that scaling is not enabled. If not equal, scaling is applied to the layer’s decoded picture when forming the composite picture. The scaling method to be used is not specified.

The decoded pictures may overlap one another in the composite picture, with the layer with the higher value of nuh\_layer\_id taking precedence, unless the flag indicates that overlap is not enabled. There is no requirement that the entire composite picture be covered by decoded layer pictures, Uncovered areas in the composite picture are undefined and left to the application to define.

The offset parameters are intended to be used when some layers are not included in the OLS, without needing to change the per layer parameters in each layer composite position info entity group.

## Proposed New Signaling for Amendment in [1]

### Layer Composite Position Info Entity Group

The layer composite position info Entity Group describes the recommended position and size of the decoded picture of each layer within a recommended composite picture comprised of decoded pictures from multiple layers.

**Syntax**

aligned(8) class LayerCompositePositionGroupBox extends   
 EntityToGroupBox('lcpg',0,0)  
{  
 unsigned int(16) num\_olss;  
 for (i=0; i<num\_olss; i++){   
 unsigned int(16) output\_layer\_set\_idx;  
 unsigned int(8) layer\_count;  
 for (j=0; j<layer\_count; j++) {  
 unsigned int(8) layer\_id;  
 unsigned int(12) lcpi\_param\_num\_bits\_minus1;  
 unsigned int(lcpi\_param\_num\_bits\_minus1+1)   
 lcpi\_top\_left\_ pos\_in\_units\_ver;  
 unsigned int(lcpi\_param\_num\_bits\_minus1+1)   
 lcpi\_top\_left\_pos\_in\_units\_hor;  
 unsigned int(lcpi\_param\_num\_bits\_minus1+1)  
 lcpi\_width\_in\_units;  
 unsigned int(lcpi\_param\_num\_bits\_minus1+1)  
 lcpi\_height\_in\_units;   
 }  
 }  
}

**Semantics**

num\_olss indicates the number of OLSs for which syntax elements are present in the entity group.

output\_layer\_set\_idx is the index of the output layer set. The mapping between output\_layer\_set\_idx and the layer\_id values shall be the same as specified by the VPS for an output layer set with index output\_layer\_set\_idx.

layer\_count: This field indicates the number of necessary layers, as defined ISO/IEC 23090-3, for this output layer set.

layer\_id: provides the nuh\_layer\_id values for the layers of the output layer set.

lcpi\_param\_num\_bits\_minus1 + 1 specifies the number of bits used to represent the lcpi\_top\_left\_pos\_ver, lcpi\_top\_left\_pos\_hor, lcpi\_width, and lcpi\_height syntax elements.

lcpi\_top\_left\_pos\_verandlcpi\_top\_left\_pos\_horindicate the recommended composite display vertical and horizontal positions, respectively, for the decoded picture of the current layer given by layer\_id. The number of bits to represent the syntax elements is lcpi\_param\_num\_bits\_minus1 + 1.

lcpi\_widthandlcpi\_heightindicate the recommended composite display width and height, respectively, for the decoded picture of the current layer given by layer\_id. The number of bits to represent the syntax elements is lcpi\_param\_num\_bits\_minus1 + 1.

### Recommended Composite Layers Info Entity Group

The recommended composite layers info entity group, together with the layer composite position information entity group, describe a layout of decoded pictures from the layers of an OLS within a recommended composite picture. If the recommended composite layer info entity group is present, then the layer composite position information entity group shall also be present.

**Syntax**

aligned(8) class RecommendedCompositeLayersInfoGroupBox extends   
 EntityToGroupBox('rclg',0,0)  
{  
 unsigned int(1) rcli\_layer\_scaling\_enabled\_flag;  
 unsigned int(1) rcli\_layer\_overlap\_enabled\_flag;  
 unsigned int(1) rcli\_unit\_size\_present\_flag;  
 unsigned int(1) rcli\_composite\_size\_present\_flag;  
 unsigned int(1) rcli\_offset\_present\_flag;  
 unsigned int(16) num\_olss;  
 for (i=0; i<num\_olss; i++){  
 unsigned int(16) output\_layer\_set\_idx;  
 if(rcli\_unit\_size\_present\_flag){  
 unsigned int(16) rcli\_unit\_size\_ver;  
 unsigned int(16) rcli\_unit\_size\_hor;  
 }  
 if(rcli\_composite\_size\_present\_flag){  
 unsigned int(16) rcli\_composite\_size\_ver;  
 unsigned int(16) rcli\_composite\_size\_hor;  
 }  
 if(rcli\_offset\_present\_flag){  
 signed int(16) rcli\_offset\_ver;  
 signed int(16) rcli\_offset\_hor;  
 }  
 }  
}

**Semantics**

rcli\_layer\_scaling\_enabled\_flag equal to 0 indicates that layerPicSizeInCompositeHeight[ i ] and layerPicSizeInCompositeWidth[ i ] derived in composition process in subclause 2.3 are equal to the width and height, respectively, of the coded picture with nuh\_layer\_id equal to i. rcli\_layer\_scaling\_enabled\_flag equal to 1 indicates that layerPicSizeInCompositeHeight[ i ] and layerPicSizeInCompositeWidth[ i ] may differ from the width and height, respectively, of the coded picture with nuh\_layer\_id equal to i.

rcli\_layer\_overlap\_enabled\_flag equal to 0 indicates that all values of Count[ y][ x ] shall be les than or equal to 1, as derived in subclause 2.3. rcli\_layer\_overlap\_enabled\_flag equal to 1 does not impose a restriction.

rcli\_unit\_size\_present\_flag equal to 1 specifies that the rcli\_unit\_size\_ver[ i ] and rcli\_ unit\_size\_hor[ i ] syntax elements are present. rcli\_unit\_size\_present\_flag equal to 0 specifies that the rcli\_ unit\_size\_ver[ i ] and rcli\_ unit\_size\_hor[ i ] syntax elements are not present.

rcli\_composite\_size\_present\_flag equal to 1 specifies that the rcli\_composite\_size\_ver[ i ] and rcli\_composite\_size\_hor[ i ] syntax elements are present. rcli\_composite\_size\_present\_flag equal to 0 specifies that the rcli\_composite\_size\_ver[ i ] and rcli\_composite\_size\_hor[ i ] syntax elements are not present.

rcli\_offset\_present\_flag equal to 1 specifies that the rcli\_offset\_ver[ i ] and rcli\_offset\_hor[ i ] syntax elements are present. rcli\_offset\_present\_flag equal to 0 specifies that the rcli\_offset\_ver[ i ] and rcli\_offset\_hor[ i ] syntax elements are not present.

num\_olss indicates the number of OLSs for which syntax elements are present in the entity group.

output\_layer\_set\_idx is the index of the output layer set. The mapping between output\_layer\_set\_idx and the layer\_id values shall be the same as specified by the VPS for an output layer set with index output\_layer\_set\_idx.

rcli\_unit\_size\_ver[ i ] and rcli\_unit\_size\_hor[ i ] indicate vertical and horizontal unit size parameters respectively, used in the composition process in subclause 2.3 for the i-th OLS. When not present, the values of rcli\_offest\_ver[ i ] and rcli\_offset\_hor[ i ] may be determined by external means.

rcli\_composite\_size\_ver[ i ] and rcli\_composite\_size\_hor[ i ] indicate the vertical and horizontal size, respectively, of the recommended composite picture in luma samples used in the composition process in subclause 2.3 for the i-th OLS. When not present, the values of rcli\_offest\_ver[ i ] and rcli\_offset\_hor[ i ] may be determined by external means.

rcli\_offset\_ver[ i ]andrcli\_offset\_hor[ i ]indicate vertical and horizontal offsets, respectively, of the positions of the decoded layer pictures used in the composition process in subclause 2.3 for the i-th OLS. When not present, the values of rcli\_offest\_ver[ i ] and rcli\_offset\_hor[ i ] are inferred to be equal to 0.

### Recommended composition process (from [2])

This subclause describes a composition process to derive sample values for a recommended composite picture, CompositePicture, for the i-th OLS.

CompositePicture[ 0]  is a 2-D sample array of size rcli\_composite\_size\_hor x rcli\_composite\_size\_ver, of the luma samples of CompositePicture.

CompositePicture[ cIdx ]  for cIdx in 1 .. 2 are 2-D sample arrays of size rcli\_composite\_size\_hor/ SubWidthC x rcli\_composite\_size\_ver/ SubHeightC.

For each layer j included in the i-th OLS, if a picture is present in the layer access unit with nuh\_layer\_id equal to j, the recommended size of the representation of the decoded picture in the composite picture, scaledLayerPic[ j ], is layerPicSizeInCompositeWidth[j ] x layerPicSizeInCompositeHeight[ j ], in luma samples, as derived below.

layerPicSizeInCompositeHeight[ j ] = lcpi\_height\_in\_units[j ] \* rcli\_unit\_size\_ver[ i ]

layerPicSizeInCompositeWidth[ j ] = lcpi\_width\_in\_units[ j ] \* rcli\_unit\_size\_hor[ i ]

When rcli\_layer\_scaling\_enabled\_flag equal to 0, scaledLayerPic[ j ] is set to the decoded picture.

Otherwise, scaledLayerPic[ j ] is derived by scaling the decoded picture.

scaledLayerPic[ j ] is a picture of size layerPicSizeInCompositeWidth[ j ] x layerPicSizeInCompositeHeight[ j ], in the luma samples .

The sample values of CompositePicture are derived as follows:

for (y = 0; y < rcli\_composite\_size\_ver[ i]; y++)  
 for (x = 0; x < rcli\_composite\_size\_hor[ i ]; x++)  
 Count[ y ][ x ] = 0  
for (j = 0 ; j< 64; j++)   
 if (j is in the i-th OLS && a picture is present in the AU with nuh\_layer\_id = j)  
 comp\_y = rcli\_offset\_ver[ i ] + lcpi\_top\_left\_pos\_in\_units\_ver[ j ] \* rcli\_unit\_size\_ver  
 comp\_x = rcli\_offset\_hor[ i ] + lcpi\_top\_left\_pos\_in\_units\_hor[ j ] \* rcli\_unit\_size\_hor  
 for (y = 0; y < layerPicSizeInCompositeHeight[ j ] \* ; y++)  
 for (x = 0; x < layerPicSizeInCompositeWidth[ j ]; x++)  
 CompositePicture[ 0 ] [ comp\_y + y ][ comp\_x + x ] = scaledLayerPic[ 0 ][ j ][ y ][ x ]  
 Count[[ comp\_y + y ][ comp\_x + x ]]++  
 for (y = 0; y < layerPicSizeInCompositeHeight[ j ]/SubWidth \* ; y++)  
 for (x = 0; x < layerPicSizeInCompositeWidth[ j ]/SubHeight; x++)  
CompositePicture[ 1 ][ comp\_y/SubHeight + y ][ comp\_x/SubWidth + x ] =  scaledLayerPic[ 1 ][ j ][ y ][ x ]  
CompositePicture[ 2 ][ comp\_y/SubHeight + y ][ comp\_x/SubWidth + x ] = scaledLayerPic[ 2 ][ j ][ y ][ x ]

The value of samples of CompositePicture[cIdx] not assigned above are undefined.

## Discussion

Figures 1, 2, and 3 illustrate example usages of multiple OLSs with different offset values, resulting in a different composite picture from the same bitstream for each OLS.

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | | 10 | |
| 9 | |

**Figure 1. Example layout for OLS 0**

|  |  |
| --- | --- |
| 2 | 3 |
| 6 | 7 |
| 10 | |
|

**Figure 2. Example layout for OLS 1**

|  |  |
| --- | --- |
| 5 | 6 |

**Figure 3. Example layout for OLS 2**

In this example, 11 layers are present in the bitstream, with values of nuh\_layer\_id from 0 .. 10. The recommended display sizes of the pictures of layers 0 to 7 are the same, 480x240. The recommend display size of pictures of layers 8 and 9 are 960x240. The recommended display size of the picture of layer 10 is 960x480. In this example, the unit size can be 480x240 luma samples, as that is the smallest unit of granularity of sizes and positions needed.

The sizes and positions in units are signaled for each layer in its own layer composite position info entity group. The signaled size corresponds to the recommended display size and not the coded picture size. Scaling is recommended to be applied when the coded picture size differs from the display size, but a scaling method is not defined. Explicit signaling of the recommended display size enables use of reference picture resizing, where the coded picture size may vary but the recommended display size remains constant. Scaling of the pictures from some layers and not from others may also be desired for other applications.

**Table 1. Example syntax element values in layer composite position info entity group (common to all OLSs)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **nuh\_layer\_id** | **lcpi\_top\_left\_ pos**  **\_in\_units\_ver** | **lcpi\_top\_left\_pos**  **\_in\_units\_hor** | **lcpi\_height\_in\_units** | **lcpi\_width\_in\_units** |
| 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 2 | 0 | 2 | 1 | 1 |
| 3 | 0 | 3 | 1 | 1 |
| 4 | 1 | 0 | 1 | 1 |
| 5 | 1 | 1 | 1 | 1 |
| 6 | 1 | 2 | 1 | 1 |
| 7 | 1 | 3 | 1 | 1 |
| 8 | 2 | 0 | 1 | 2 |
| 9 | 3 | 0 | 1 | 2 |
| 10 | 2 | 2 | 2 | 2 |

**Table 2. Example syntax elements in multiple recommended composite layers info entity group**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OLS idx** | **rcli\_composite\_size\_ver** | **rcli\_composite\_size\_hor** | **rcli\_offset\_ver** | **rcli\_offset\_hor** |
| 0 | 960 | 1920 | 0 | 0 |
| 1 | 960 | 960 | 0 | -960 |
| 2 | 480 | 960 | -240 | -480 |

## References

[1] w19454: Text of ISO/IEC 14496-15:2019 DAM 2 Carriage of VVC and EVC in ISOBMFF

[2] m54093: AHG9/AHG12: Recommended multi-layer composite picture SEI messages

[3] m54832/JVET-S2017: Technologies under consideration for VSEI

# Carriage of T.35 messages

## Summary of the proposal

Reference to issues: [MPEG/Systems/FileFormat/isobmff#128](http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/128)

In MPEG#139, the options given in this section were narrowed down and it was agreed that we have an intent to move this section to an amendment.

Section 12.2 provides a T.35 metadata item for static images.

Section 12.3 introduces a method for carriage of **dynamic** T.35 messages using **sample group**, where only the header of T.35 is normally included to the SampleGroupDescriptionBox. This can be used to provide the signaling of which types of T.35 to expect in the track without putting the entire T.35 message to the sample group description box, although the latter is also supported. The T.35 messages can be carried in the sample data and the placement of T.35 is out of scope in ISOBMFF. This approach is similar to the Prefix SEI concept and can be used to figure out what types of T.35 to expect and where to find them in the track.

The characteristics of these approaches, and their applicability to video codecs that are, or are not, able to carry an SEI, are summarized in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| Approach |  | SEI-supporting codec? | Not SEI-supporting? |
| In-stream | Supported today | √ | X |
| Sample Groups carry 'prefix' of in-stream SEIs, to enable indexing | Annotates today's situation | √ | X |
| Metadata item | Essentially a 'static' solution | √ | √ |

Note that the SEI prefix SEI and Manifest SEI can be used with SEI-supporting codecs today, to document presence and characteristics of SEIs.

## T.35 metadata item

### Definition

A T.35 metadata item carries an ITU-T T.35 message. The item\_type value shall be 'T.35'.

### Syntax

aligned(8) class T35Information {  
 bit(8) itu\_t\_t35\_country\_code;  
 if(itu\_t\_t35\_country\_code == 0xFF)  
 {  
 bit(8) itu\_t\_t35\_country\_code\_extension\_byte;  
 }  
 bit(8) itu\_t\_t35\_payload[];  
}

### Semantics

itu\_t\_t35\_country\_code shall be a byte having a value specified as a country code by Rec. ITU-T T.35 Annex A, or the country code extension value 0xFF.

itu\_t\_t35\_country\_code\_extension\_byte if present, shall be a byte having a value specified as a country code by Rec. ITU-T T.35 Annex B.

itu\_t\_t35\_payload shall be the payload containing data registered as specified in Rec. ITU-T T.35.

The ITU-T T.35 *terminal provider code* and *terminal provider oriented code* shall be contained in the first one or more bytes of the itu\_t\_t35\_payload, in the format specified by the Administration that issued the *terminal provider code*. Any remaining itu\_t\_t35\_payload data shall be data having syntax and semantics as specified by the entity identified by the ITU-T T.35 *country code*, *terminal provider code* and *terminal provider oriented code*.

## T.35 sample group

The T.35 sample group allows the storage of dynamic T.35 metadata. The T.35 sample group can be used to signal which types of T.35 are present in which samples. The actual T.35 payload data is stored in samples and the placement of T.35 is out of scope in ISOBMFF. The T.35 sample group is used just to signal which T.35 messages can be expected in a track and where to find them.

The 'kind' of a T.35 message is decided by some number of prefix bytes that include the terminal-provider code, terminal provider oriented code and some following bytes that indicate which format and version of T.35 message, defined by that provider, follows. Unfortunately, the number of bytes required to differentiate this varies by provider; it must be such that a terminal either supports (all) T.35 messages of that 'kind', or not.

All the T.35 metadata is stored in samples and the SampleGroupDescriptionBox is only used to indicate which T.35 messages can be expected, and where to find them. This approach allows to deal with very frequent T.35 metadata and does not interfere with currently existing T.35 carriage methods where T.35 messages are carried in samples (T.35 SEIs using HEVC or T.35 Metadata OBUs using AV1, etc.). The sample grouping mechanism is used to identify samples holding a specific T.35 message. However, the placement of T.35 within the sample is out of scope in ISOBMFF and should be defined in derived specifications. Figure 10 shows an example how different T.35 messages can be stored in the samples and SampleGroupDescriptionBox is used to signal the headers of each T.35 message type.

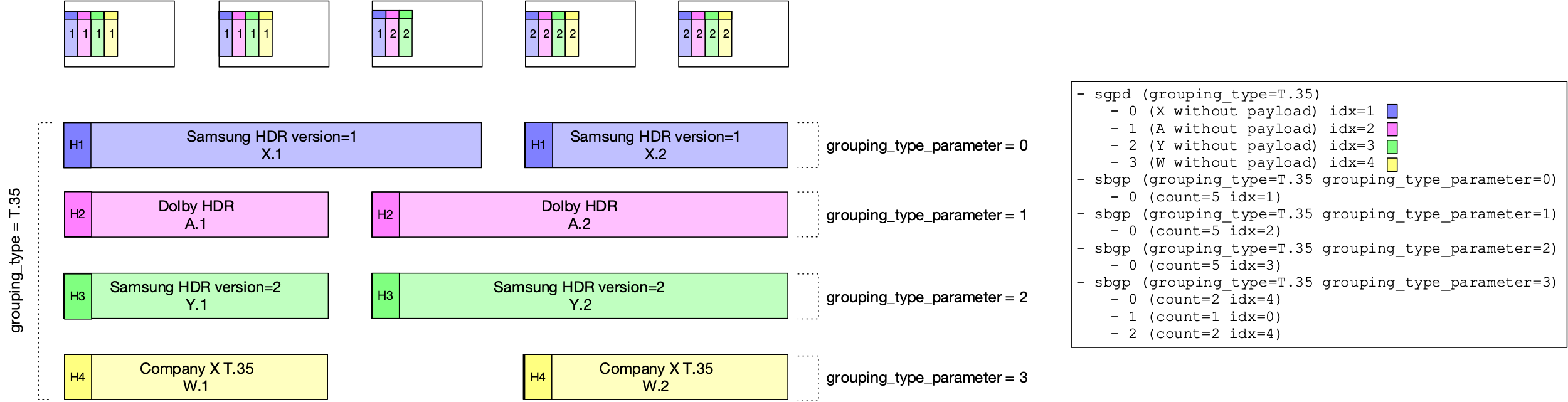


Figure : Example for T.35 signaling using ISOBMFF sample groups, where all metadata is stored in samples and only headers (including T.35 version) are stored in the SampleGroupDescriptionBox

#### Definition

Each sample of a track may be associated with zero or more sample group descriptions, each of which defines a record of T.35 information of different types. The same T.35 information may apply to different samples.

The grouping\_type=‘T.35’ is defined as a grouping criterion for T.35 metadata. The SampleTableBox or TrackFregmentBox of a track can contain zero or more SampleToGroupBoxes or CompactSampleToGroupBoxes with the grouping\_type='T.35'. Each sample group description shall document exactly one 'kind' of T.35 message, i.e. shall map to T.35 sample group entries with at least the same ITU-T T.35 *country code*, *terminal provider code* and *terminal provider oriented code,* and T.35 type and version (if applicable). The placement of T.35 metadata within the samples is out of scope and should be specified in derived specifications.

NOTE 1 both, *terminal provider code* and *terminal provider oriented code* are stored within first bytes of the itu\_t\_t35\_header. Also, the signaling of the T.35 version may appear after *terminal\_provider\_oriented\_code*. The number of bytes used to identify the origin and version of the T.35 metadata depends on the country and the entity which defines the syntax of the T.35 message.

#### Syntax

class T35SampleGroupEntry extends SampleGroupDescriptionEntry('T.35') {  
 bit(8) itu\_t\_t35\_country\_code;  
 if(itu\_t\_t35\_country\_code == 0xFF)  
 {  
 bit(8) itu\_t\_t35\_country\_code\_extension\_byte;  
 }  
 bit(8) itu\_t\_t35\_data[];  
}

#### Semantics

itu\_t\_t35\_country\_code shall be a byte having a value specified as a country code by Rec. ITU-T T.35 Annex A, or the country code extension value 0xFF.

itu\_t\_t35\_country\_code\_extension\_byte if present, shall be a byte having a value specified as a country code by Rec. ITU-T T.35 Annex B.

itu\_t\_t35\_data shall be the payload containing data registered as specified in Rec. ITU-T T.35.

The ITU-T T.35 *terminal provider code* and *terminal provider oriented code* shall be contained in the first one or more bytes of the itu\_t\_t35\_header, in the format specified by the Administration that issued the terminal provider code. It should also include a part of the ITU-T T.35 message which specifies the version of the message and may include any remaining data of the ITU-T T.35 message having syntax and semantics as specified by the entity identified by the ITU-T T.35 *country code*, *terminal provider code* and *terminal provider oriented code*.

# Integrating new codecs

Issue: *<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, …).

# SubSampleInformationBox update

Issue: *<http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/94>*

## Discussion

The G-PCC specification [1] mandates the use of ‘subs’ box to indicate the type of G-PCC units in samples of a single G-PCC track ‘gpc1’ or ‘gpcg’. In addition, due to G-PCC structure, most G-PCC samples of a given document tend to comprise the same kinds of sub-samples. As an example, each sample may for instance comprise 3 types of sub-samples: 1 geometry data unit, 1 attribute data unit indicating color, and for example 1 attribute data unit indicating reflectance.

For each sub-sample, the ‘subs’ box describes the following properties: subsample\_size, subsample\_priority, discardable and codec\_specific\_parameters. While subsample\_size generally varies for each sub-sample, subsample\_priority, discardable and codec\_specific\_parameters often remain constant for a given type of sub-sample. Consequently, in such cases, the contents of ‘subs’ box are extremely redundant. Based on simulations, the cost with 1 geometry and 2 attribute data units is about 2 kbits/sec at 10 Hz (or 6kbits/sec at 30Hz), hence even more at higher sampling frequencies.

While especially visible in the context of G-PCC, this issue is not specific to volumetric data and may apply to any media type. For instance, when using subpicture-based sub-samples (flags=4, ISO/IEC 14496-15 [3]) with fixed sub-picture positions in VVC, or when using tile-based/slice-based sub-sample description in HEVC, description of sub-samples is also likely to be heavily redundant.

## Benefits

If we consider the same G-PCC data (as in the example in the above section) to be encapsulated as a single G-PCC track, and if we suppose subsample\_size is coded on 16 bits, the description for a single sub-sample reaches 64 bits:

* 16 bits for subsample\_size;
* 8 bits for subsample priority;
* 8 bits for discardable;
* 32 bits for codec\_specific\_parameters.

Therefore, for a sequence of N samples with our G-PCC example (1 GDU plus 2 ADU data units), the total cost will be 3\*64\*N bits, i.e. 192\*N bits.

Savings using the new flags value:

* With applies\_to\_all\_samples: we save 32 bits for the entry\_count but mainly N times 32 bits for the sample\_delta.
* With fixed\_nb\_subsamples\_per\_sample, here 3, we save (N-1) times 16 bits for the subsample\_count.

Using the new syntax proposal (see the following section), the cost of describing the subsample information for the sequence of N samples is as follows:

* The first sample (1 GDU and 2 ADU) will cost an additional 8 bits per sub-sample (has\_reference equal to 0 and the 7 bits of the reserved), meanings 3\*72 bits.
* For the N-1 following samples, all sub-samples are described by reference to one of the 3 previous descriptions of the first sample. Therefore, only 24 bits are used for each sub-sample (16 bits for the subsample\_size, 1 bit for has\_reference and 7 bits for reference\_id). The description of the N-1 samples costs (N-1)\*3\*24 bits.

Consequently, the total cost is 144 + 72\*N bits.

Table 1 illustrates the gain obtained for the sub-sample information (subsample\_size, subsample\_priority, discardable, codec\_specific\_parameter) with different number of samples.

|  |  |  |  |
| --- | --- | --- | --- |
| N | Standard | Proposal | Gain |
| 10 | 1920 | 864 | -55% |
| 30 | 5760 | 2304 | -60% |

Table : Gain between proposed subs versus ISOBMFF subs (for subsample information)

## Proposal

It is proposed to support new sub-sample description, by modifying Section 8.7.7. of ISOBMFF as follows:

### Add to section 8.7.7.1 Definition

Sub-sample information box provides 2 ways to describe sub-sample information:

* Version 0 and 1 explicitly describe all the properties of each sub-sample;

Version 2 and 3 enable describing some of the properties for some sub-samples through a reference to a previous sub-sample.

### Replace section 8.7.7.2 Syntax with:

aligned(8) class SubSampleInformationBox  
 extends FullBox('subs', version, flags) {  
 if (flags & applies\_to\_all\_samples ) {  
 entry\_count = number of samples in the track of track fragment;   
 sample\_delta = 1;  
 } else {   
 unsigned int(32) entry\_count;   
 if (flags & regular\_sample\_pattern) {  
 unsigned int(32) sample\_delta;  
 }  
 }  
 if (flags & fixed\_nb\_subsamples\_per\_sample) {  
 unsigned int(16) subsample\_count;  
 }  
 for (int i=0; i < entry\_count; i++) {  
 if (!(flags & applies\_to\_all\_samples)   
 && !(flags & regular\_sample\_pattern) ) {  
 unsigned int(32) sample\_delta;   
 }  
 if (!(flags & fixed\_nb\_subsamples\_per\_sample) ) {  
 unsigned int(16) subsample\_count;  
 }  
 if (subsample\_count > 0) {  
 for (int j=0; j < subsample\_count; j++) {  
 if(version == 1 **|| version == 3**)  
 {  
 unsigned int(32) subsample\_size;  
 } else {  
 unsigned int(16) subsample\_size;  
 }

if (version == 0 || version == 1) {

unsigned int(8) subsample\_priority;  
 unsigned int(8) discardable;  
 unsigned int(32) codec\_specific\_parameters;  
 } else { // subsample information by reference  
 **unsigned int(1) has\_reference;   
 if (has\_reference == 1) {  
 unsigned int(7) reference\_id;  
 // Set fields based on ref.  
 } else {  
 unsigned int(7) reserved;**  unsigned int(8) subsample\_priority;  
 unsigned int(8) discardable;  
 unsigned int(32) codec\_specific\_parameters;  
 }  
 }  
 }  
 }  
 }  
}

### Replace section 8.7.7.3 Semantics with (new parts in yellow)

version is an integer that specifies the version of this box (0 or 1 in this document indicate explicit encoding of the sub-sample information; 2 or 3 indicate encoding possibly by reference of the sub-sample information). Version 2 or 3 also make use of generic (in opposition to codec-specific) flags values.

flags is a 24-bit integer with flags; the following values are defined for version 2 or 3 of this box:

applies\_to\_all\_samples: Flag mask is 0x800000. When set, this value indicates that the sub-sample information box applies to all samples of the track or track fragment (this avoids repeating a sample\_delta always equal to 1).

regular\_sample\_pattern: Flag mask is 0x400000. When set, this value indicates that the sample\_delta always takes the same value and is declared only once in the SubsampleInfomationBox.

fixed\_nb\_subsamples\_per\_sample: Flag mask is 0x200000. When set, this value indicates that the subsample\_count can be encoded once for all samples having a sub-sample description and is not repeated in the loop of entries.

NOTE: For using the syntax shown above, the flags parameter is defined as a generic part (for example the first byte) and a codec-specific part (for example the two last bytes). Each part defines its own set of values. The last two bytes may be used by derived specifications to define their own flags values, for example to identify the type of the subsamples.

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

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

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

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

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

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

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

has\_reference equal to 0 means that subsample\_priority, discardable and codec\_specific\_parameters are explicitly indicated for considered sub-sample. When equal to 1 means that the values of these properties are identical to the corresponding ones for the sub-sample indicated by reference\_id.

reference\_id is an integer that specifies the index of a sub-sample in a reference table. Each entry in said table comprises a set of values for subsample\_priority, discardable and codec\_specific\_parameters, a number of occurrences, and an appearance index. The reference table is initially empty. When a sub-sample for which has\_reference equals 0 is met, a new entry is created based on the values of subsample\_priority, discardable and codec\_specific\_parameters, with a number of occurrences of 0, and an appearance index equal to the number of sub-samples that have been previously processed. It is then checked whether the reference table comprises fewer than 128 entries. If so, the new entry is appended to the table. If not, the new entry replaces the entry of the table with lowest number of occurrences (1st criterion) and lowest appearance index (2nd criterion). When a sub-sample for which has\_reference equals 1 is met, the entry with corresponding reference\_id is updated by incrementing its number of occurrences by 1 and by setting its appearance index to the number of sub-samples previously processed. To preserve random access, the reference table is reset on each sync sample.

# MovieFragmentHeaderBox update

Issue : *<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

Issue : *<http://mpegx.int-evry.fr/software/MPEG/Systems/FileFormat/isobmff/-/issues/98>*

## 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 19.2, 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.

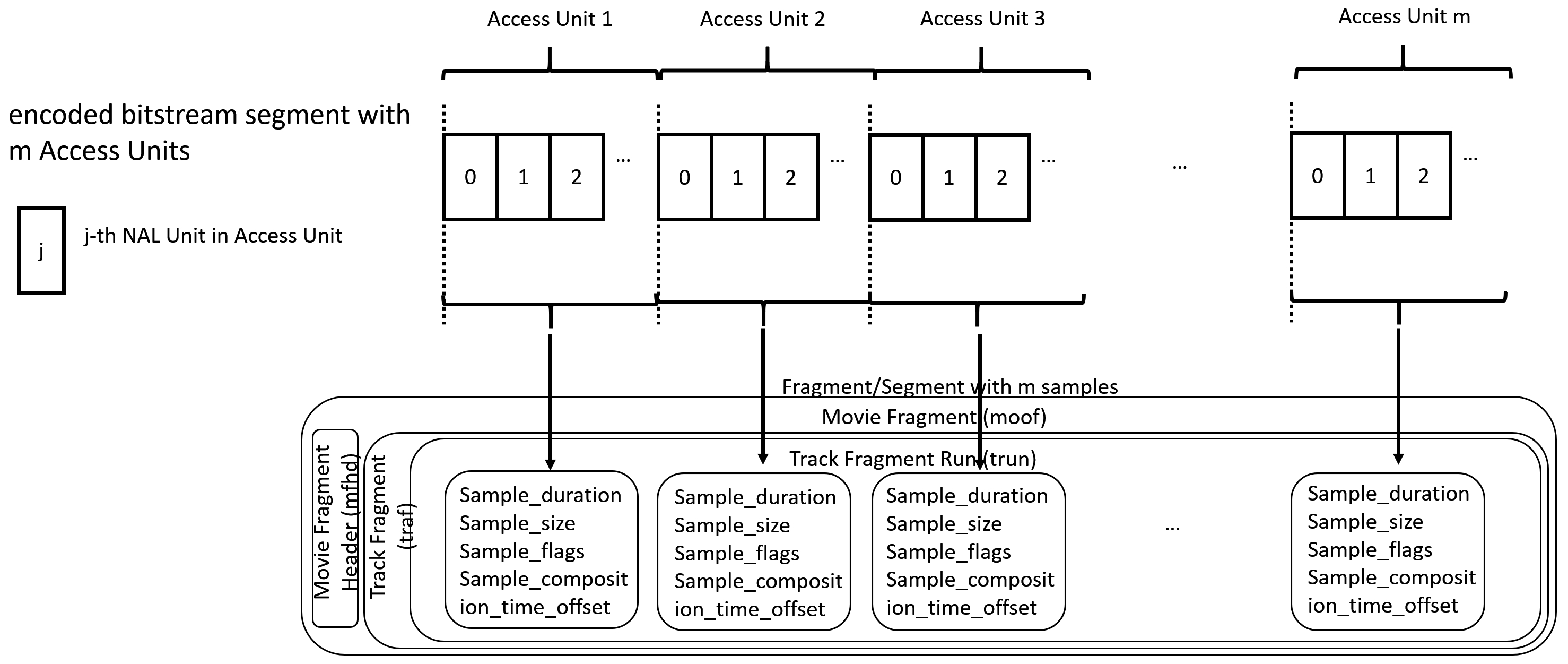


Figure 19.2.1: Conventional sample information in TrackRunBox

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.

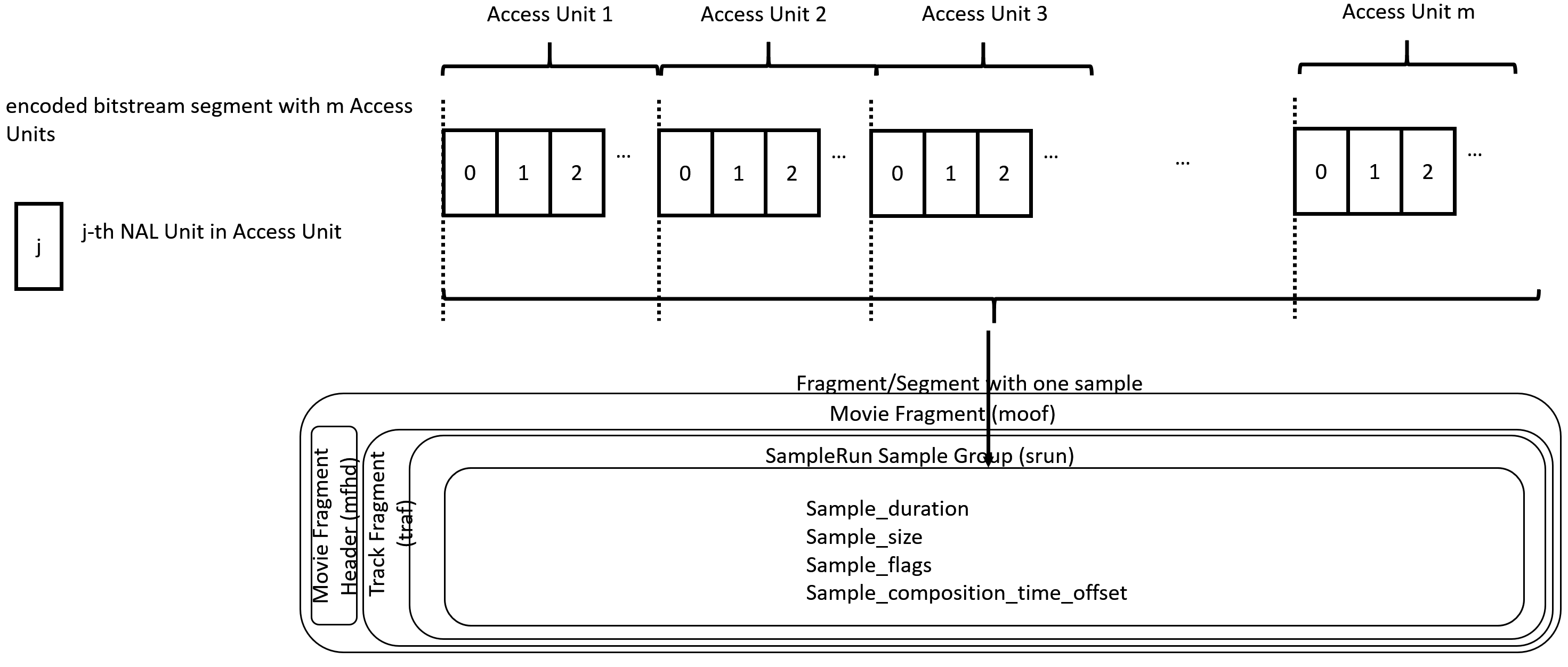


Figure 19.2.2: An example usage of SampleRun Sample group.

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 'tspt' transformed video track.
  + Mapping the samples in the 'tspt' 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 'tspt' 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 an 'tspt' transformed video track. A sample run documents contiguous samples of the untransformed track where each access unit in the mapped sample of the 'tspt' transformed video track is its own sample.

Note: The sample run sample group is limited to be used in an 'tspt' 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.

# FrameRateBox

discussions

A proposal to add an optional box for framerate signalling was intorduced in m60004 and the group agreed to study if adding such signaling is needed in ISOBMFF.

## Introduction

Frame rate signalling is relevant for many applications of the ISO Base Media File format. In particular, for live applications framerate information is traditionally carried in the bit-stream.

It is asserted that the current consensus seems to be:

* recommend setting default\_sample\_duration and timescale such that timescale/default\_sample\_duration gives the frame rate
* define a brand to let packagers explicitly signal when they followed the recommendation
* add a note indicating that default\_sample\_duration should not change in fragments (or rather that is should not lead to a change in refresh rate), maybe referring to existing sentence in C.1

However, it is asserted that some important disadvantages to generic carriage in ISO Base media file format (beyond CMAF) are:

1. This cannot signal multiple framerates in a track (e.g. 25 Hz, 50 Hz)
2. It is not trivial to update existing content with additional signalling if it was not already in this format, as default sample duration may have been used in a different way, including setting it to zero.
3. A lot of systems ignore brands, basically we cannot rely on brands, and in that case this signalling would be implicit. In fact, the current approach is implicit.
4. It does not signal the framerate in progressive mp4 which may also have some applications in the industry. In this format there is no default\_sample\_duration.

To address these issues, the FrameRateBox proposed below may be added to VisualSampleEntry, in a similar fashion as the BitRateBox, PixelAspectRatioBox and ColourInformationBox ('colr'). Parsers that do not understand this box may ignore it as is the default for ISOBMFF parsers. Parsers that need this information may interpret this box directly.

## Specification text

**12.1.6 Frame Rate Box**

**12.1.6.1 Definition**

Box Types: ‘frmt’

Container: Sample Description Box (‘stsd’)

Mandatory: No

Quantity: Zero or More

Frame rate signalling provided in the FrameRateBox does not impose any additional requirements on the presentation or decode timelines. Actual sample timing may violate frame rates signalled in the FrameRateBox. Instead, this box may be used to supplement media presentations with informative frame rates for usage in end-to-end workflows.

Frame rate information may be supplied in a FrameRateBox placed in a

VisualSampleEntry. This box supplies the framerate information related to the sample entry. Multiple FrameRateBoxes shall only be used if more than one framerate is present.

**12.1.6.2 Syntax**

class FrameRateBox extends FullBox(‘frmt’, version = 0,0){

if (version == 0){

unsigned int(32) framerateN; // framerate nominator

unsigned int(32) framerateD(1);// optional framerate denominator

}

}

**12.1.6.3 Semantics**

version: the version of this box. Default value is zero

framerateN: nominator of the frame rate information.

framerateD: denominator of the frame rate information with a default value of 1.