ISO Base Media File Format and Apple HEVC Stereo Video

Format additions

Version 0.9 (Beta)

June 21, 2023

Note: The information contained within this document is preliminary and is subject to change.

Introduction

This document describes Apple extensions of, or specialized use of, the ISO Base Media Format (a.k.a. ISOBMFF) to support spatial media. Spatial media is intended to produce a richer experience for the user; whether a richer audio experience, a richer visual experience, or a combination of the two.

As has been true with the QuickTime File Format (QTFF) since its introduction in 1991, and ISOBMFF based upon QTFF, the ISOBMFF format is meant to serve as a container of media using tracks and movie-level structures. The movie format of that media continues to evolve from the earliest “postage stamp” (i.e., very low resolution) video with one- and two-channel uncompressed or barely compressed audio to modern formats performing remarkable levels of compression for 4K and even 8K video with very rich multichannel audio encoding. This is coupled with facilities to carry and present captions such as WebVTT text tracks or closed captions embedded with video. Static and timed metadata can be carried to augment the presented media. Still other kinds of media tracks have been supported and will likely get added.

To support media that delivers rich spatial experiences, the ISOBMFF foundations are being extended with new media formats, with extensions to supported media formats, and with new constructs to inform relationships among the new and earlier supported media. Some of these extensions are specific to their spatial nature while others are fundamental and used by the former. This is all intended to be done in a way—where possible—so an existing ISOBMFF player or processor can interact with the spatial media possibly in a reduced form while allowing new playback or processing to take fuller advantage of the newly afforded richness.

This document describes new and updated file format structures to support spatial media. Some of these structures are accessible through Apple AVFoundation and CoreMedia framework interfaces and those serve as the preferred alternative to direct structural access when running on a platform with Apple frameworks available. Those reading or writing the format directly—pursuant to relevant licensing—should however be able to accomplish their goals with the structural descriptions in the following sections.

Another consideration for ISOBMFF is that it is used in a fragmented form for HTTP based delivery technologies such as HTTP Live Streaming. The support in standalone MP4 files and fragmented MP4 resources is much the same.

Note: The words "may", "should", and "shall" are used in the conventional specification sense, that is, respectively, to denote permitted, recommended, or required behaviors.

References

[QTFF] QuickTime File Format (QTFF), 2016

[ISOBMFF] ISO/IEC 14496-12:2020 ISO Base Media File Format

[ISONALU] ISO/IEC 14496-15:2019 “Carriage of network abstraction layer (NAL) unit structured video in the ISO base media file format”

[HEVC] ISO/IEC 23008-2:2020 “High efficiency video coding”

[METADATA] “Video Contour Map Payload Metadata within the QuickTime Movie File Format—Format Additions”

Stereo Video

Stereoscopic, stereopsis and stereo media

Just as stereo audio indicates different audio for the left and the right ear, visual media can be stereoscopic in which a view is available to be presented to the left eye and another view is available to be presented simultaneously to the right eye. The presentation of both the left and right views allows for an effect known as *stereopsis* which can be defined as:

> *the perception of depth produced by the reception in the brain of visual stimuli from both eyes in combination; binocular vision*.

The production and display of this is sometimes referred in cinema as "3D" and the implementation and storage of the views can vary. This cinema use of "3D" should be distinguished from "3D rendering" involving a framework like Metal where geometry, materials, lighting and cameras are modeled and rendered by a GPU or CPU. In the latter case, such three-dimensional rendering might produce a view as seen from the left eye and a simultaneous view seen from the right eye and therefore be stereoscopic. Rendering of a scene might however produce a single-view that is not stereoscopic. This is sometimes called *monoscopic* to distinguish it from stereoscopic.

Stereoscopic media can also be captured photographically where two cameras might be offset horizontally to produce a video where the left-eye view and the right-eye view are each encoded. In this case, there's not necessarily any modeling of the scene or any GPU rendering. Playback takes care to present the left captured view to the viewer's left eye and the right captured view to the viewer's right eye. These left and right captured views might also have been processed. Different storage strategies exist to carry stereoscopic content. This document describes how this is done using standardized ISO/ITU formats and some extensions to the ISOBMFF.

Stereoscopic video tracks

ISOBMFF standalone and fragmented movies can include a single video track associated with both the left and right eyes. This video track's access units carry both a base and secondary layer that correspond to a primary stereo eye view (left or right) and the complementary stereo eye (i.e., right if the primary is the left, left if the primary is the right).

The expectation is that both stereo-eye views (i.e., the left-eye view and right-eye view) will be shown to both the left and right eyes simultaneously. Both stereo views are available and synchronized according to the movie timeline. When played, stereopsis is achieved.

> **Note:** The ability to produce a movie with just one stereo eye video track (whether left or right) can be useful in production workflows. Two tracks in the same movie or in two movies might be useful. These might be combined into a new movie either by encoding with both views in one video track or less commonly by carrying two video tracks. While potentially applicable to ISOBMFF, it is not a described use case here.

This document introduces a VisualSampleEntry extension that can signal among other things whether the associated video track is stereoscopic and which stereo eyes are carried in that

video track. This new signaling is referred to as **Video Extended Usage** and is described in a later section of this document. For now, the point is that this allows a movie reader to detect stereo-related video tracks and to identify the stereo eyes contributed by that track so it can configure presentation or other processing. While the video track itself uses a video-compression format and signals it has a left and right stereo view, the Video Extended Usage is meant to be more easily parsed and to be applicable to non-MV-HEVC video.

Multiview video tracks and MV-HEVC compression

An ISOBMFF (e.g., .mp4) movie video track encodes video as either uncompressed or compressed video media samples. In the case of compressed video, High Efficiency Video Coding [HEVC] defines extensions to encode more than one view in the compressed bitstream for each coded video frame (or access unit). Defined in Annex G of the HEVC spec [HEVC], **Multiview high efficiency video coding** defines how layers corresponding to views can be encoded and associated. This is sometimes written as "MV-HEVC" for "Multiview HEVC".

The visual sample-entry shall include a **Video Extended Usage** visual sample-entry extension box (described later in this document) indicating which stereo eye views—left, right or both—are carried in the MV-HEVC video track. For MV-HEVC, both left and right-eye views should be available. A hero eye indicating the default stereo eye may optionally be signaled. This construct allows a client to determine the stereoscopic nature of the video track without needing to parse for MV-HEVC bitstream details in the decoder configuration.

The video bitstream requirements of MV-HEVC coding, visual sample-entry and video media samples are described in the document *Apple HEVC Stereo Video Interoperability Profile*.

Video Extended Usage

This specification introduces an optional visual sample-entry extension that can signal additional aspects regarding the use of the video track's decoded frames. The new extension is called the **Video Extended Usage** and uses the box type '`vexu`' (optionally pronounced as "vex you"). Details needed for video frame decoding continue to be carried in the visual sample-entry header (e.g., the compression type, the dimensions) and other visual sample-entry extensions such as '`colr`' and compression type specific decoder configurations. The '`vexu`' extension instead describes aspects beyond fundamental decode such as whether the video frames are stereoscopic related or otherwise organized so movie format playback might need to process or display the decoded video frames in a special way before presenting to the viewer.

Traditionally, a video track within a movie file or movie fragments can be decoded and immediately presented with little additional processing other than perhaps scaling, cropping and placement. For video that is "real-world captured" such as from a camera or computer generated, this is the norm. Even non-linear editing mostly works with video as stored in the movie files, perhaps applying effects, but otherwise encoding video that is directly presentable.

Increasingly today, a video track may be used as input into a rendering process and may not be suitable to show a viewer immediately. For example, a stereoscopic "3D" movie should present the left-eye view to the viewer's left-eye, and the right-eye view to the viewer's right eye. Understanding the video track delivers stereoscopic views and which stereoscopic views are available is needed. For MV-HEVC video, the presence of [HEVC] and [ISONALU] constructs such as the '`hvcC`' and '`lhvC`' extension data might seem sufficient, but unfortunately requires all readers to parse significant HEVC detail. Here, the video track's '`vexu`' visual sample-entry

extension serves the role of easy-to-interpret signaling that the video is stereoscopic. The ‘vexu’ visual sample-entry extension must be consistent with what is signaled in the decoder configuration. Beyond MV-HEVC, it may be desirable to use other video-compression formats (e.g., non-multiview HEVC) or uncompressed video to carry stereoscopic video without requiring their fundamental decoded bitstream to signal the stereo use—something the format might not support. A video extended usage extension can be added indicating a video track carries two stereo eyes or is for only one of the two stereo eyes. A ‘vexu’ extension can also be added indicating that the decoded video is organized in some other way described in a future version of this specification. This can be combined with stereoscopic detail that there is both a left and a right stereo eye view. Here, playback and processing needs to understand the video track uses this alternative organization so it can route the left view and the right view portions of the decoded frame to the respective viewer eye.

Video Extended Usage box hierarchy

The video extended usage extension box specifies the usage of and details relevant to that usage beyond the decode of the video samples. This is an optional extension and needed only when special or useful interpretation of the video in playback or processing is required. If the state signaled is not required for playback or processing, the extension may still be present but there is no expectation the reader understands it.

This extension box is a Box hierarchy and contains further boxes signaling particular aspects of the video. These contained boxes may be leaf boxes—typically FullBoxes—or box hierarchies themselves. There is also a mechanism to indicate that contained boxes must be understood by a reader and if not, that that part of the box hierarchy has failed to be processed. That error can propagate upwards failing a local subtree or even the entire video extended usage box extension. This can in turn indicate the video should not be presented or processed as the reader’s implementation lacks sufficient support.

Current box types defined in the ‘vexu’ box hierarchy:

FourCC	FourCC	FourCC	Box syntax element
vexu			VideoExtendedUsageBox
	eyes		Video Stereo
		must	RequiredBoxTypesBox
		stri	StereoViewInformationBox
		hero	HeroStereoEyeDescriptionBox

1. Video Extended Usage (‘vexu’) box

This section describes how the Video Extended Usage extension box is organized and the constituent boxes.

1.1. Definition

Box Type: ‘vexu’
Container: Visual Sample Entry (different coding types)
Mandatory: No
Quantity: Zero or one

The Video Extended Usage extension is a QuickTime File Format atom [QTFF] which is the same as a Box in ISO/IEC 14496-12 [ISOBMFF]. As we will use the bitstream syntax from ISO/IEC 14496-12, we will use “box” interchangeably with “atom”. References to ImageDescription for QTFF are also interchangeable with references to the ISO 14496-12 VisualSampleEntry.

The Video Extended Usage box is held in a VideoExtendedUsageBox and has the ISO Box boxtype of ‘vexu’ for “video extended usage”.

As a Box, it can contain zero or more children Boxes that together signal the nature of the associated track samples’ extended usage. Having no child boxes is valid but likely not useful. Having only child FreeSpaceBoxes is appropriate if wanting to reserve space in the VisualSampleEntry.

To allow new or otherwise unknown VideoExtendedUsageBox child boxes to be introduced while allowing older readers to know they do not understand enough to process or present the video track, a mechanism is introduced to indicate mandatory and by implication optional child boxes. Additionally, child boxes can indicate if their own structure can be optional so readers can recognize versions they do not support.

New Boxes should not be introduced into the VideoExtendedUsageBox unless documented in this specification or a successor version of this specification.

1.2. Syntax

```
aligned(8) class VideoExtendedUsageBox extends Box(‘vexu’) {
   RequiredBoxTypesBox();       // optional if no required boxes specified
   StereoViewBox();             // optional
   Box()[];                     // other optional boxes with FreeSpaceBox()
reserved for its expected use
}
```

1.3.Semantics

`VideoExtendedUsageBox` contains zero or more child boxes that signal something about the use of the video. Child boxes will be defined in this specification now or in the future. Child boxes might be defined in external specifications but the box_type used there should be registered not to collide with boxes introduced in this or related specifications. The order of child boxes in the `VideoExtendedUsageBox` and in all contained boxes recursively is not prescribed. A reader should be prepared to find boxes in any order.

Note: As `FreeSpaceBox` (‘free’) has a very common meaning in ISOBMFF and QTFF, one or more `FreeSpaceBoxes` may occur among the child boxes and should be interpreted as having no other meaning than taking up space. There is no guarantee that the payload of a FreeSpaceBox will contain exclusively zero (0) bytes but that is encouraged.

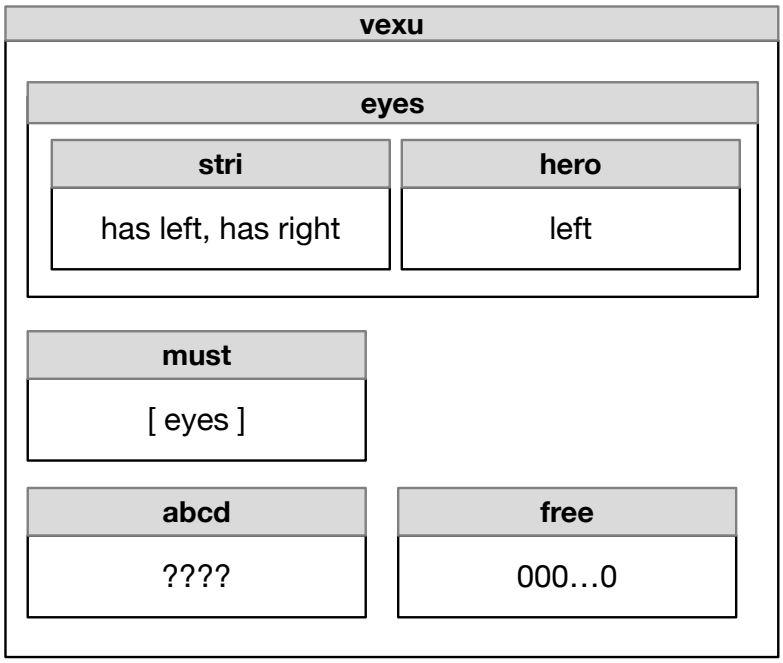
Note: An empty `VideoExtendedUsageBox` (i.e., no children boxes) is allowed but should generally not be included in the VisualSampleEntry. It may however be useful to reserve space by including a `VideoExtendedUsageBox` in concert with a contained `FreeSpaceBox` (‘`free`’).

New child boxes may be introduced in the future that are not described in this spec. There is a mechanism in the structure of VideoExtendedUsageBox to signal the set of child boxes an implementation must understand in order to usefully process the video track. This allows future boxes to be introduced and older implementations to know they should not present the video with newer signaled features.

Besides standard boxes, the `VideoExtendedUsageBox` may contain zero or more Boxes that describe specific kinds of signaling. In the following section, each kind of box is described.

Note that a `VideoExtendedUsageBox` should carry only one child box for a specific feature. So, for example, there should not be more than one feature box for stereo view signaling.

`VideoExtendedUsageBox` with contained boxes might look like the following:



This ‘`vexu`’ box here contains an ‘`eyes`’ box and another box. This box is optional but if contained is included in a ‘`must`’ box to indicate they are required to be interpreted by the reader to indicate that it must be understood.‘`free`’ boxes allow space to be reserved. The *other boxes* represent unknown boxes.

The order of child boxes within a box may vary. Readers should not expect a fixed order of child boxes at any level. A writer should not include a child box of a particular type more than once if it is documented to occur only once.

1.2. Required box types (‘must’) box

1.2.1. Definition

Box Type: `‘must’`
Container: A video box within the Video Extended Usage Box (`‘vexu’`) or within contained Boxes
Mandatory: No
Quantity: zero or one

If a parent box at any level within `VideoExtendedUsageBox` has a 'must' box, that '`must`' box contains a list of box types corresponding to boxes that are peers to the 'must' box which the reader must successfully interpret in order for the parent box to be successfully interpreted. In other words, if the reader does not recognize a required box type, or if it fails to parse that box or any *required* child box of that box, the reader must consider that to be a failure to parse the parent box. If the `VideoExtendedUsageBox` box is considered failed, the track is to be ignored by the reader.

Each kind of child box within a VideoExtendedUsageBox might serve to signal a feature according to this specification. The set of boxes is interpreted by the reader to understand what the video represents (e.g., it uses stereo views). Some of the signaling is necessary for further processing. Other boxes may be informative but not strictly required for interpretation. It's important to understand all required boxes. RequiredBoxTypesBox enables the adding of new boxes in the future that may be required for interpretation and further processing.

Each box within VideoExtendedUsageBox may contain any hierarchy of boxes suitable to signal some aspect about the video. Some of these may be Boxes with a hierarchy of other boxes and some may be FullBoxes. The RequiredBoxTypesBox enumerates the box types of its sibling boxes corresponding to required boxes. If not enumerated within the RequiredBoxTypesBox, the child box's interpretation is optional.

The RequiredBoxTypesBox contains an array of FourCCs corresponding to box types. If an entry is 0, the entry is reserved and is not interpreted as a required box type. FreeSpaceBoxes of box type 'free' should not be included in a RequiredBoxTypesBox. If RequiredBoxTypesBox includes a box type that doesn't correspond to a child box, the reader can ignore the absence but might want to log this for diagnostic purposes. The use of box types of missing boxes within a RequiredBoxTypesBox is however discouraged.

The FreeSpaceBox box type of 'free' should not be referenced from a RequiredBoxTypesBox.

The RequiredBoxTypesBox can also occur within other boxes within the VideoExtendedUsageBox box hierarchy that are themselves box hierarchies. These uses of RequiredBoxTypesBox serve to indicate local requirements on boxes that must be recognized and understood for local parsing to be valid. A local box can fail and that will influence the validity of the parent box if the parent box itself is referenced from another RequiredBoxTypesBox that is a sibling of the parent box.

1.2.2.Syntax

```
aligned(8) class RequiredBoxTypesBox extends FullBox('must', 0, 0 ) {
   unsigned int(32) required_box_types[];
}
```

1.2.3.Semantics

required_box_types is an array of zero or more box types corresponding to sibling boxes that must be understood by readers to properly process the video associated with the VideoExtendedUsageBox. For each non-zero entry in required_box_types[], the reader should confirm the box type is recognized. A value of zero (0) in a required_box_types[] entry can be ignored, allowing for space for entries to be reserved.

1.2.4.Reader behavior and RequiredBoxTypesBox

A reader of a movie file video track having an associated `VideoExtendedUsageBox` should be able to detect if it understands enough about the VideoExtendedUsageBox contents to process the video beyond fundamental decoding. This further processing, interpretation and/or rendering is the meaning of "extended" within the identifier name "`VideoExtendedUsageBox`".

This specification is intended to be extended in future versions. A particular video track may carry several kinds of signaling that differ from other video tracks within the movie file or in other movie files. The kind of signaling within a box can itself evolve over time. In all these cases, it is important to know if the set of child boxes of a Box must be understood. While the most obvious case is child boxes of `VideoExtendedUsageBox`, the approach can apply to any Box serving as the root of a box hierarchy within the larger hierarchy.

The following describes reader behavior that is aware of `RequiredBoxTypesBox`:

1. Read (or start processing) the box hierarchy (e.g., `VideoExtendedUsageBox`).
2. Retrieve the contained `RequiredBoxTypesBox` child box if any.
    1. If present, confirm all non-zero entries of `required_box_types[]` are recognized box types and if not treat the parent box of the `RequiredBoxTypesBox` as not processable.
    2. Ignore all zeroed entries of `required_box_types[].`
3. Enumerate each non-zero box type in the `required_box_types[]` of the child `RequiredBoxTypesBox` of `VideoExtendedUsageBox` using an index from 0 to the length of `required_box_types[]` minus 1 and confirm the referenced box is understood.
    1. For each successive `index`, retrieve the child box with `required_box_types[index]` and confirm understanding of its structure.
        1. For child FullBoxes, the reader should consider the version and flags to confirm understanding as well as anything else that may be relevant to its interpretation.
        2. For child Boxes that are box hierarchies themselves and allow `RequiredBoxTypesBox`, the reader should descend into the Box, retrieve the optional contained `RequiredBoxTypesBox` and perform this algorithm recursively.
        3. If the parsing of the FullBox or the child Box hierarchy fails, the reader should treat the current level as invalid and propagate that failure upwards. Any semantics discovered at the current level should not be propagated upwards as partial semantics is

misleading (e.g., stereo view and something else both being required should not signal stereo views if the other parsing failed).

The VideoExtendedUsageBox parsing follows this same algorithm. In this case, however, failing to parse required child boxes of VideoExtendedUsageBox means the track has failed to parse. This can best be interpreted as though the entire video track is unavailable.

1.3. Video stereo view signaling ('eyes') box

The StereoViewBox signals if the video track represents stereo 3D content. This can take the form of a track that delivers both a left stereo eye and a right stereo-eye view or a track that carries only the left stereo eye or only the right stereo eye.

If both left and right stereo eyes are carried, the views might be combined in one image organized in some way or might be contained in some kind of multiview coding.

If the left stereo eye is in one video track and the right stereo eye is in a second video track, each will carry its own VideoExtendedUsageBox with a StereoViewBox. The indication of which eye is carried will be appropriate for each corresponding video track.

For completeness, it is also possible to signal monoscopic video which is to say no stereo view carriage. If this is the case, however, the StereoViewBox can be eliminated from the VideoExtendedUsageBox. If the VideoExtendedUsageBox would be left with no child boxes, the VideoExtendedUsageBox can be eliminated from the VisualSampleEntry as well.

If the recorded stereo video has a designated "hero" eye, the StereoViewBox carries a HeroStereoEyeBox. There are rules specified for signaling when the stereo eye video is separated into two video tracks.

1.3.1. Definition

Box Type: 'eyes'
Container: Video Extended Usage Box ('vexu')
Mandatory: No
Quantity: Zero or one

1.3.2. Syntax

```
aligned(8) class StereoViewBox extends Box('eyes') {
   RequiredBoxTypesBox();                // as needed
   StereoViewInformationBox();
   HeroStereoEyeDescriptionBox();        // optional
   Box()[];                              // other optional boxes
}
```

1.3.3. Semantics

StereoViewInformationBox is a required box indicating which stereo eyes are present.

`RequiredBoxTypesBox` indicates the box types for other boxes that must be understood to interpret the current version of the `StereoViewsBox`. The `StereoViewInformationBox` box type of 'stri' is required within `RequiredBoxTypesBox` if a `RequiredBoxTypesBox` is used.

Other boxes indicate additional detail about the stereo view representation and are described in later sections of this document. The set of boxes may evolve.

1.4. Stereo view information

1.4.1. Definition

Box Type: `'stri'`
Container: Video Stereo View Box (`'eyes'`)
Mandatory: Yes
Quantity: One

The `StereoViewInformationBox` can carry the stereography related information indicating the presence of particular stereo eyes (i.e., left stereo eye, right stereo eye) as well as some other flags.

1.4.2. Syntax

```
aligned(8) class StereoViewInformationBox extends FullBox('stri', 0, 0) {
   unsigned int(4) reserved;       // reserved, set to 0
   unsigned int(1) eye_views_reversed;
   unsigned int(1) has_additional_views;
   unsigned int(1) has_right_eye_view;   // video contains a right-eye view
   unsigned int(1) has_left_eye_view;    // video contains a left-eye view
}
```

1.4.3. Semantics

`has_left_eye_view`: indicates the stereo left eye is present in video frames

`has_right_eye_view`: indicates the stereo right eye is present in video frames

`has_additional_views`: indicates that one or more additional views may be present beyond stereo left and stereo right eyes (e.g,. a "centerline" view)

`eye_views_reversed`: indicates the order of the stereo left eye and stereo right eye are reversed from the default order of left being first and right being second

`reserved`: 4 bits reserved for future versions of this specification; for this version of this specification, writers should set it to 0 and readers should treat any non-zero values as if this box is invalid.

Because there is a flag field for the left eye and a field for the right eye, both fields should be set to indicate both eyes are represented in video frames. Moreover, both `has_left_eye_view` and `has_right_eye_view` can be set to 0 to indicate that the frame is monoscopic.

Note: If the video is monoscopic, the `StereoViewBox` can also be absent from the `VideoExtendedUsageBox`. If the only signaling is of

monoscopic video, the `VideoExtendedUsageBox` can be absent from the `VisualSampleEntry`, too.

If an alternative organization is signaled in the future, the default order of stereo eyes in video will be left eye than right eye. By setting the `eye_views_reversed` field, the order is reversed so for the right is to the left in the frame and the left is to the right in the frame. For multiview coding, there is no implied ordering and the `eye_views_reversed` field should be set to 0.

Note: It may be useful to signal in a multiview coding approach the presence of the left stereo eye, the right stereo eye and a third view which is the "centerline" or "down the nose" view which is between these and is neither the left nor the right. It may not be possible or appropriate to use the left or the right eye for this central view. There may be coding efficiencies from being able to include such a view in multiview coding.

The `has_left_eye_view` and `has_right_eye_view` fields specify the presence of the left and right stereo eye views but the fields do not signal how those are stored. That is accomplished with other child boxes of `StereoViewBox`.

Note that the `has_additional_views` field indicates that views beyond those for the left eye and the right eye are present. One example of this might be a "centerline" view. Note that signaling the presence of the "centerline" is not necessary if both the left and right eye flags are zeroed indicating a monoscopic view.

1.5.Hero Stereo Eye Description

1.5.1. Definition

Box Type: `'hero'`
Container: Video Stereo View Box (`'eyes'`)
Mandatory: No
Quantity: Zero or one

The `HeroStereoEyeDescriptionBox` indicates which stereo eye, if any, has been denoted as the "hero" eye. If so signaled, this indicates the other stereo eye view is derived from the specified stereo eye and may be useful when choosing which eye to use in a monoscopic viewing environment. If neither eye is the hero eye, the `HeroStereoEyeDescriptionBox` does not need to be included in the `StereoViewBox`. If the hero eye is not known, a `HeroStereoEyeDescriptionBox` might not appear in the `StereoViewBox`.

It is possible to include a `HeroStereoEyeDescriptionBox` but set the flags to indicate that neither the left nor the right stereo eye are set. While unconventional, this allows an implementation to reserve space for the box for potential setting later in processing. Readers should be prepared to recognize such a `HeroStereoEyeDescriptionBox` that signals no hero eye.

1.5.2.Syntax

```
aligned(8) class HeroStereoEyeDescriptionBox extends FullBox('hero', 0, 0)
{
   unsigned int(8) hero_eye_indicator; // 0 = none, 1 = left, 2 = right, >=
3 reserved
}
```

1.5.3. Semantics

`HeroStereoEyeDescriptionBox` is used to indicate which of the left or right stereo eye is the "hero" eye, if any.

`hero_eye_indicator:` is used in the `HeroStereoEyeDescriptionBox`, to signal which hero eye, if any is specified. Defined values are:

0: the hero eye is not specified

1: indicates the left eye is the hero eye

2: indicates the right eye is the hero eye

>= 3: are reserved and should not be used for implementation of this version of this specification. If a reserved value is read, a reader should treat the signaling as though no hero eye is specified. If the hero eye is not specified, it is recommended that `HeroStereoEyeDescriptionBox` not be included in the `StereoViewBox`. The value of 0 is allowed as it can be used to reserve space for the `HeroStereoEyeDescriptionBox` that might be adjusted later.

The `HeroStereoEyeDescriptionBox` signals the left or the right stereo eye independently of whether or not the `StereoViewBox`'s `StereoViewInformationBox` indicates if the order of the stereo eyes is reversed in order. The hero left eye is always the left stereo eye and the hero right eye is always the right stereo eye.

Auxiliary Video Track Handler Type

To date, video tracks have used the handler type 'vide'. Other track types such as audio and metadata use their own handler types (i.e., 'soun' and 'meta', respectively). This has never been a problem as the decoded video can be presented as-is though perhaps with scaling or cropping. With stereo video tracks, the decoded video may require additional processing such as view extraction before being presented to the user.

Movies or fragmented movie files for HTTP Live Streaming may now use the *auxiliary video* or 'auxv' handler type to "hide" the video track from naive reader decode and presentation. An example where this is useful might be a video track with an alternative layout of images signaled using a video extended usage atom.

If the decoded video however displays in a backwards compatible way when delivered—such as MV-HEVC showing just a default view from the stereo pair—there is no need to use the 'auxv' handler type. Use 'vide' in this case. Also, HTTP Live Streaming mediates what media is shown so the multivariant playlist can serve to filter display of particular video streams.

Also, in a production workflow where users expect to see and confirm the decoded video even if further processing might be expected when delivered to an end user, it is okay to use ‘vide’ so tools that already present or process video tracks can find the track.

Spatial Audio

The experience is enhanced if audio can represent the spatial acoustic environment. Just as listening to stereo audio is richer than listening to mono audio, even richer audio representations are possible with appropriate audio coding. A number of advanced audio technologies exist and they may be used in isolation or in combination.

ISOBMFF audio tracks use audio codecs to encode and carry audio—uncompressed and compressed—and the codecs can use different audio technologies. Some technologies are applicable to spatial audio and when used in that way, the audio might be termed *spatial audio*. The ISOBMFF format can carry a wide range of uncompressed and compressed format, some supporting spatial playback. By introducing new audio codecs in audio tracks, the movie can carry spatial audio.

ISOBMFF movies may contain any supported audio format. As additional formats are supported, those may prove useful for delivering more richer experiences.

Spatial Audio Technologies

By way of a quick summary, there are three audio technologies typically used in the spatial audio realm.

One technology is termed *channel-based* and can include more than one audio channel which are each mapped onto the speaker layout. Termed *multi-channel* audio, this term is typically used with 5.1 and 7.1 audio. The number and placement of these channels in the soundscape can be more varied and the channel count can be more or less than the six of 5.1 or the eight of 7.1. Indeed, stereo has two channels, so is in fact multichannel, but that term is almost never applied to stereo.

Another technology, termed *ambisonics*, is a modeling of three-dimensional audio in a 360-degree space. It allows recording, mixing and playing back such audio. Just as multi-channel audio can vary in channel count, ambisonic audio can vary in *order* allowing more refined audio with higher degreed ambisonics. Audio is fixed in location but surrounds the user.

A third technology is termed *object-based audio*. This models each sound source as an object with associated metadata describing three-dimensional placement and other relevant characteristics. Individual objects might be fixed in 3D or might move in 3D over time.

This specification does not prescribe which encoding formats or which technologies within those formats are used to realize a richer experience.

Timed Metadata and Spatial Media

Spatial media tracks such as video may benefit from having associated timed metadata. While this might be injected in AVC or HEVC SEI signaling, an alternative is to use a parallel metadata track. This timed metadata track can use the ISOBMFF ‘`mebx`’ format’s ability to carry a number

of metadata items for a time range. Metadata item keys need not be related to other item keys allowing a flexible way to signal a variety of structural or descriptive information.

We consider one kind of timed metadata payload related to describing the parallax of decoded stereoscopic video frames.

Caption parallax timed metadata items

Traditionally, captions are placed in the horizontal and vertical axes over video. With the introduction of stereoscopic video, however, there is a risk of depth collision if captions are placed in Z so they might intersect with stereoscopic elements that have a parallax (i.e., horizontal disparity) that is less than the screen plane. This “depth conflict” can produce viewer discomfort. To account for this, captions can have their parallax adjusted to have a more negative parallax than the video elements so there is no collision.

The document “Video Contour Map Payload Metadata within the QuickTime Movie File Format—Format Additions” [METADATA] specifies the structure of a metadata payload structure that can serve to describe parallax values associated with 2D areas of a stereoscopic video frame. This metadata is specific to the time-aligned video frame.

This payload is carried as metadata items within samples within QuickTime file format [QTFF] timed metadata or ISOBMFF [ISOBMFF] multiplexed metadata. Note: Both of these use the `'mebx'` format type of the `'meta'` track handler type. The payloads can also occur in fragmented movie files in both ISOBMFF and QTFF.

Conclusion

This document describes extensions to the ISOBMFF format. These extensions are introduced by Apple to allow for delivery of stereoscopic video, spatial audio and timed metadata signaling to influence parallax of any subtitles associated with the video. This is applicable to both ISOBMFF standalone movie and fragmented movie files. It attempts to build on existing structures where that was deemed appropriate and introduces new constructs where there was a perceived deficit or a benefit in introducing a new construct. The evolution of the ISOBMFF format extensions described here may be taken through standards processes in time.

Document Revision History

Date	Revision	Notes
2023-06-21	0.9	First version