INTERNATIONAL STANDARD

ISO/IEC 23090-14

First edition
2023-06

Information technology — Coded representation of immersive media —

Part 14: Scene description

Technologies de l'information — Représentation codée de média immersifs —

Partie 14: Description de scènes

Reference number
ISO/IEC 23090-14:2023(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

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

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

A list of all parts in the ISO 23090 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

This document defines the MPEG-I Scene Description. It provides an architecture for the MPEG-I Scene Description, a set of extensions based on ISO/IEC 12113, a set of APIs, and storage formats for scene description documents and scene description updates documents.

Information technology — Coded representation of immersive media —

Part 14: Scene description

1 Scope

This document specifies extensions to existing scene description formats in order to support MPEG media, in particular immersive media. MPEG media includes but is not limited to media encoded with MPEG codecs, media stored in MPEG containers, MPEG media and application formats as well as media provided through MPEG delivery mechanisms. Extensions include scene description format syntax and semantics and the processing model when using these extensions by a Presentation Engine. It also defines a Media Access Function (MAF) API for communication between the Presentation Engine and the Media Access Function for these extensions. While the extensions defined in this document can be applicable to other scene description formats, they are provided for ISO/IEC 12113.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 12113, *Information technology — Runtime 3D asset delivery format — Khronos glTF™ 2.0*

ISO/IEC 14496-12, *Information technology — Coding of audio-visual objects — Part 12: ISO base media file format*

ISO/IEC 21778, *Information technology — The JSON data interchange syntax*

IEEE 754-2019, *IEEE Standard for Floating-Point Arithmetic*

IETF RFC 6902, *JavaScript Object Notation (JSON) Patch*

IETF RFC 8259, *The JavaScript Object Notation (JSON) Data Interchange Format*

3 Terms, definitions, abbreviated terms, and conventions

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 12113 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1.1
asset
3D scene described by a *scene description document* (3.1.10) together with corresponding *scene description data* (3.1.9)

3.1.2
node
element in the *scene graph* (3.1.12)

3.1.3
media access function
function that retrieves and prepares media for rendering on request by the *presentation engine* (3.1.7)

3.1.4
media pipeline
chain of media processing components to process media

3.1.5
object
node in a *scene description document* (3.1.10)

3.1.6
patch document
document that contains update instructions

Note 1 to entry: For example, update instruction can be provided as defined in RFC 6902.

3.1.7
presentation engine
engine that processes and renders the *asset* (3.1.1)

3.1.8
scene activation time
time on the media timeline at which the scene described by a *scene description document* (3.1.10) takes effect in the *presentation engine* (3.1.7)

3.1.9
scene description data
binary data that is described by *scene description document* (3.1.10)

3.1.10
scene description document
document describing a 3D scene

Note 1 to entry: For example, scene description document is containing description of node hierarchy, materials, cameras, as well as description information for meshes, animations, and other constructs.

3.1.11
scene description update
patch document (3.1.6) to a *scene description document* (3.1.10) or a *scene description document* (3.1.10)

3.1.12
scene graph
data structure used to represent *objects* (3.1.5) in a 3D scene and their hierarchical relationships

3.1.13
timed accessor
accessor defined in ISO/IEC 12113 that has an MPEG_accessor_timed extension and is used to describe access to timed data

3.1.14
timed data
timed media
media, which when decoded results in content, possibly containing internal timing values, to be presented at a given presentation time and for a certain duration

3.2 Abbreviated terms

3D	Three-Dimensional
3DoF	Three Degrees of Freedom
6DoF	Six Degrees of Freedom
API	Application Programming Interface
AR	Augmented Reality
DASH	Dynamic Adaptive Streaming over HTTP
dB	Decibel
DSR	Diffuse to Source Ratio
glTF	Graphics Language Transmission Format
HOA	Higher Order Ambisonics
ISOBMFF	ISO Base Media File Format
JSON	JavaScript Object Notation
MAF	Media Access Function
MPEG	Moving Picture Experts Group
IDL	Interface Definition Language
PCM	Pulse-Code Modulation
RT60	60 dB Reverberation Time
SDP	Session Description Protocol

3.3 Conventions

3.3.1 General

The mathematical operators used in this document are similar to those used in the C programming language. However, the results of integer division and arithmetic shift operations are defined more precisely, and additional operations are defined, such as exponentiation and real-valued division. Numbering and counting conventions generally begin from 0.

3.3.2 Arithmetic operators

+	addition
−	subtraction (as a two-argument operator) or negation (as a unary prefix operator)
*	multiplication, including matrix multiplication
/	integer division with truncation of the result toward zero. For example, 7 / 4 and −7 / −4 are truncated to 1 and −7 / 4 and 7 / −4 are truncated to −1.
÷	division in mathematical equations where no truncation or rounding is intended.

3.3.3 Logical operators

!	Boolean logical "not".

3.3.4 Relational operators

>	Greater than.
>=	Greater than or equal to.
<	Less than.
<=	Less than or equal to.
==	Equal to.
!=	Not equal to.

3.3.5 Bit-wise operators

~	bit-wise "not". When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.
&	bit-wise "and". When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.
|	bit-wise "or". When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.
^	bit-wise "exclusive or". When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.
x >> y	arithmetic right shift of a two's complement integer representation of x by y binary digits. This function is defined only for non-negative integer values of y. Bits shifted into the MSBs as a result of the right shift have a value equal to the MSB of x prior to the shift operation.
x << y	arithmetic left shift of a two's complement integer representation of x by y binary digits. This function is defined only for non-negative integer values of y. Bits shifted into the LSBs as a result of the left shift have a value equal to 0.

3.3.6 Assignment operators

=	assignment operator.
++	increment, i.e. x++ is equivalent to x = x + 1; when used in an array index, evaluates to the value of the variable prior to the increment operation.
--	decrement, i.e. x-- is equivalent to x = x − 1; when used in an array index, evaluates to the value of the variable prior to the decrement operation.

+= increment by amount specified, i.e. x += 3 is equivalent to x = x + 3, and x += (−3) is equivalent to x = x + (−3).

−= decrement by amount specified, i.e. x −= 3 is equivalent to x = x − 3, and x −= (−3) is equivalent to x = x − (−3).

3.3.7 Other operators

y..z range operator/notation.
This function is defined only for integer values of y and z. When z is larger than or equal to y, it defines an ordered set of values from y to z in increments of 1. Otherwise, when z is smaller than y, the output of this function is an empty set. If this operator is used within the context of a loop, it specifies that any subsequent operations defined are performed using each element of this set, unless this set is empty.

3.3.8 Order of operation precedence

When order of precedence in an expression is not indicated explicitly by use of parentheses, the following rules apply:

— Operations of a higher precedence are evaluated before any operation of a lower precedence.

— Operations of the same precedence are evaluated sequentially from left to right.

Table 1 specifies the precedence of operations from highest to lowest; a higher position in the table indicates a higher precedence.

NOTE For those operators that are also used in the C programming language, the order of precedence used in this document is the same as used in the C programming language.

Table 1 — Operation precedence from highest (at top of table) to lowest (at bottom of table)

operations (with operands x, y, and z)
"x++", "x--"
"!x", "−x" (as a unary prefix operator)
"x * y", "x / y", "x ÷ y", "x % y"
"x + y", "x − y" (as a two-argument operator)
"x << y", "x >> y"
"x < y", "x <= y", "x > y", "x >= y"
"x == y", "x != y"
"x & y"
"x | y"
"x && y"
"x || y"
"x ? y : z"
"x..y"
"x = y", "x += y", "x −= y"

3.3.9 Text description of logical operations

In the text, a statement of logical operations as would be described mathematically in the following form:

```
if( condition 0 )
   statement 0
else if( condition 1 )
   statement 1
...
else /* informative remark on remaining condition */
   statement n
```

may be described in the following manner:

... as follows / ... the following applies:

— If condition 0, statement 0

— Otherwise, if condition 1, statement 1

— ...

— Otherwise (informative remark on remaining condition), statement n

Each "If ... Otherwise, if ... Otherwise, ..." statement in the text is introduced with "... as follows" or "... the following applies" immediately followed by "If ... ". The last condition of the "If ... Otherwise, if ... Otherwise, ..." is always an "Otherwise, ...". Interleaved "If ... Otherwise, if ... Otherwise, ..." statements can be identified by matching "... as follows" or "... the following applies" with the ending "Otherwise, ...".

In the text, a statement of logical operations as would be described mathematically in the following form:

```
if( condition 0a && condition 0b )
   statement 0
else if( condition 1a || condition 1b )
   statement 1
...
else
   statement n
```

may be described in the following manner:

... as follows / ... the following applies:

— If all of the following conditions are true, statement 0:

  — condition 0a

  — condition 0b

— Otherwise, if one or more of the following conditions are true, statement 1:

  — condition 1a

  — condition 1b

— ...

— Otherwise, statement n

In the text, a statement of logical operations as would be described mathematically in the following form:

if(condition 0)

  statement 0

if(condition 1)

  statement 1

may be described in the following manner:

When condition 0, statement 0

When condition 1, statement 1

In addition, a “continue” statement, which is used within loops, is defined as follows:

The “continue” statement, when encountered inside a loop, jumps to the beginning of the loop for the next iteration. This results in skipping the execution of subsequent statements inside the body of the loop for the current iteration. For example:

```
for( j =0; j < N; j++ ) {
  statement 0
  if( condition 1 )
      continue
  statement 1
  statement 2
}
```

is equivalent to the following:

```
for( j =0; j < N; j++ ) {
  statement 0
  if( !condition 1 ) {
     statement 1
     statement 2
  }
}
```


4 Overview and architecture

4.1 Overview

This document enables inclusion of timed media in a scene description. This is achieved through first defining features of a scene description that describe how to get the timed media, and second how a rendering process expects the data once it is decoded. In this version of the document, the features are defined as extensions to the glTF format defined in ISO/IEC 12113, see Clause 5.

In addition to the extensions, which provide an integration of timed media with the scene description, the document describes a reference scene description architecture that includes components such as Media Access Function, Presentation Engine, Buffer Control & Management, and Pipelines. To enable cross-platform/cross-vendor interoperability, the document defines Media Access Function (MAF) API and Buffer API, see Clause 6. The MAF API provides an interface between the Media Access Function and the Presentation Engine. The Buffer API is used to allocate and control buffers for the exchange of data between Media Access Function and Presentation Engine.

Not only the timed media described by the scene description may change over the time but also the scene description itself. The document defines how such change of a scene description document is signalled to the Presentation Engine.

Finally, a scene description may be stored, delivered, or extended in a way that is consistent with MPEG formats. The document defines a number of new features that allow a carriage utilizing ISO/IEC 14496-12 and its derived specifications, see Clause 7.

4.2 Architecture

The scene description is consumed by a Presentation Engine to render a 3D scene to the viewer. The extensions defined in this document, allow for the creation of immersive experiences using timed media. The scene description extensions are designed with the goal of decoupling the Presentation Engine from the Media Access Function. Presentation Engine and Media Access Function communicate through the Media Access Function API, which allows the Presentation Engine to request timed media

required for the rendering of the scene. The Media Access Function will retrieve the requested timed media and make it available in a timely manner and in a format that can be immediately processed by the Presentation Engine. For instance, a requested timed media asset may be compressed and residing in the network, so the Media Access Function will retrieve and decode the asset and pass the resulting decoded media data to the Presentation Engine for rendering. The decoded media data is passed in form of buffers from the Media Access Function to the Presentation Engine. The requests for timed media are passed through the Media Access Function API from the Presentation Engine to the Media Access Function.

Figure 1 depicts the reference architecture.

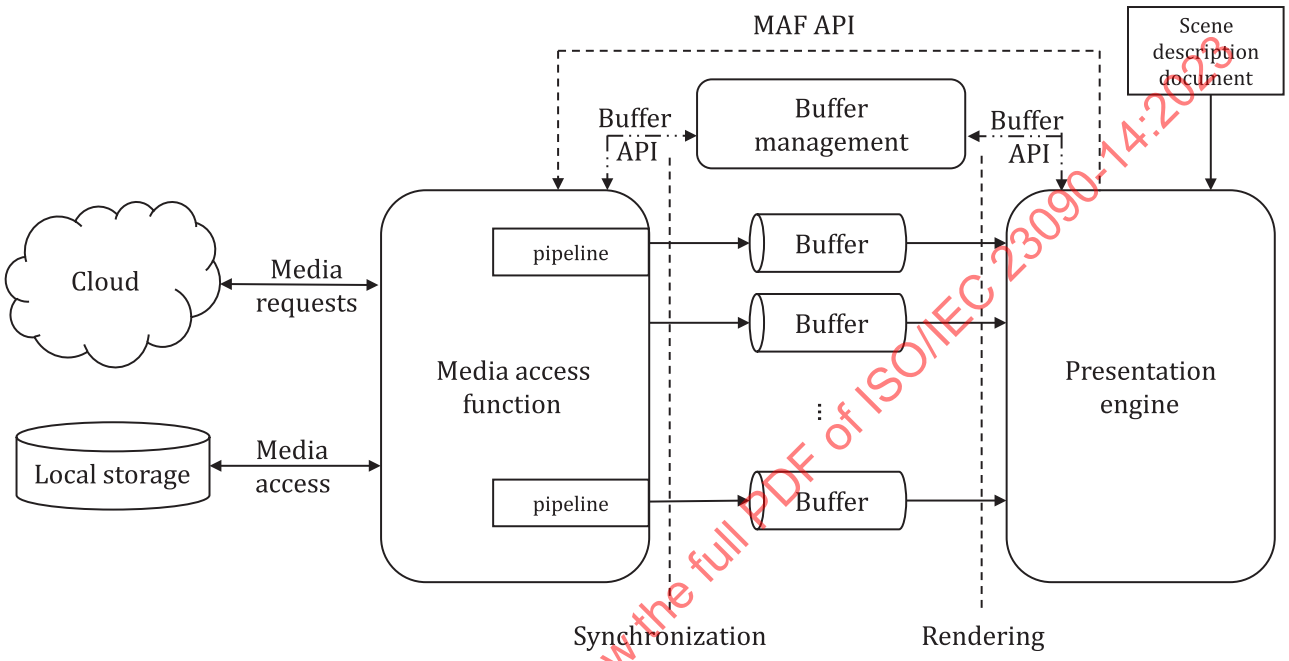

Figure 1 — Scene description reference architecture

The interfaces (MAF API, Buffer API) and extensions to ISO/IEC 12113 are within the scope of this document.

The following principles apply:

— The format of the buffers shall be provided by the scene description document and shall be passed to the MAF through the Media Access Function API

— Pipeline shall perform necessary transformations to match the buffer format and layout declared in the scene description for that buffer

— The fetching of scene description document and scene description updates may be triggered by the MAF.

Figure 1 depicts the reference architecture for scene description. The corresponding procedures are described as follows:

a) The Presentation Engine receives and parses the scene description document and following scene description updates

b) The Presentation Engine identifies timed media that needs to be presented and identifies the required presentation time

c) The Presentation Engine then uses the MAF API to request the media and provides the following information:

   1) where the MAF can find the requested media

   2) what parts of the media and at what level of detail

   3) when the requested media has to be made available

   4) in which format it wants the data and how it is passed to the Presentation Engine

d) The MAF instantiates the media fetching and decoding pipeline for the requested media at the appropriate time.

   1) It ensures that the requested media is available at the appropriate time in the appropriate buffers for access by the Presentation Engine

   2) It ensures that the media is decoded and reformatted to match the expected format by the Presentation Engine as described by the scene description document

The exchange of data (media and metadata) shall be done through buffers (circular and static buffers). The buffer management shall be controlled through the Buffer API. Each buffer should contain sufficient header information to describe its content and timing.

The information provided to the Media Access Function by the Presentation Engine allows it to

— Select the appropriate source for the media (multiple could be specified) and the MAF may select based on preferences and capabilities. Capabilities may for example be decoding capabilities or supported formats. Preferences may for example be user settings.

— For each selected source,

   i) access the media by using a media access protocol;

   ii) setup the media pipeline to provide the information in the correct buffer format.

The MAF may obtain additional information from the Presentation Engine in order to optimize the delivery, for example the required quality for each of the buffers, the exact timing information, etc.

The Media Access Function shall setup and manage the pipeline for each requested media or metadata. A pipeline takes as input one or more media or metadata tracks and outputs one or more buffers. The pipeline shall perform all the necessary processing, such as streaming, demultiplexing, decoding, decryption, and format conversion to match the expected buffer format. The final buffer or set of buffers are then used to exchange data with the Presentation Engine.

An example of pipelines setup is depicted in Figure 2 for the case of a V-PCC compressed point cloud object that is referenced in the scene description. Pipeline #1 creates four video decoders and one patch data decoder. The pipeline is also responsible for processing this data and performing 3D reconstruction based on the received information. The reconstructed data is then fed to the final buffer that is accessed by the Presentation Engine. Pipeline #2 on the other hand is not performing the 3D reconstruction process and provides decoded raw data onto the buffers, which are accessed by the Presentation Engine.

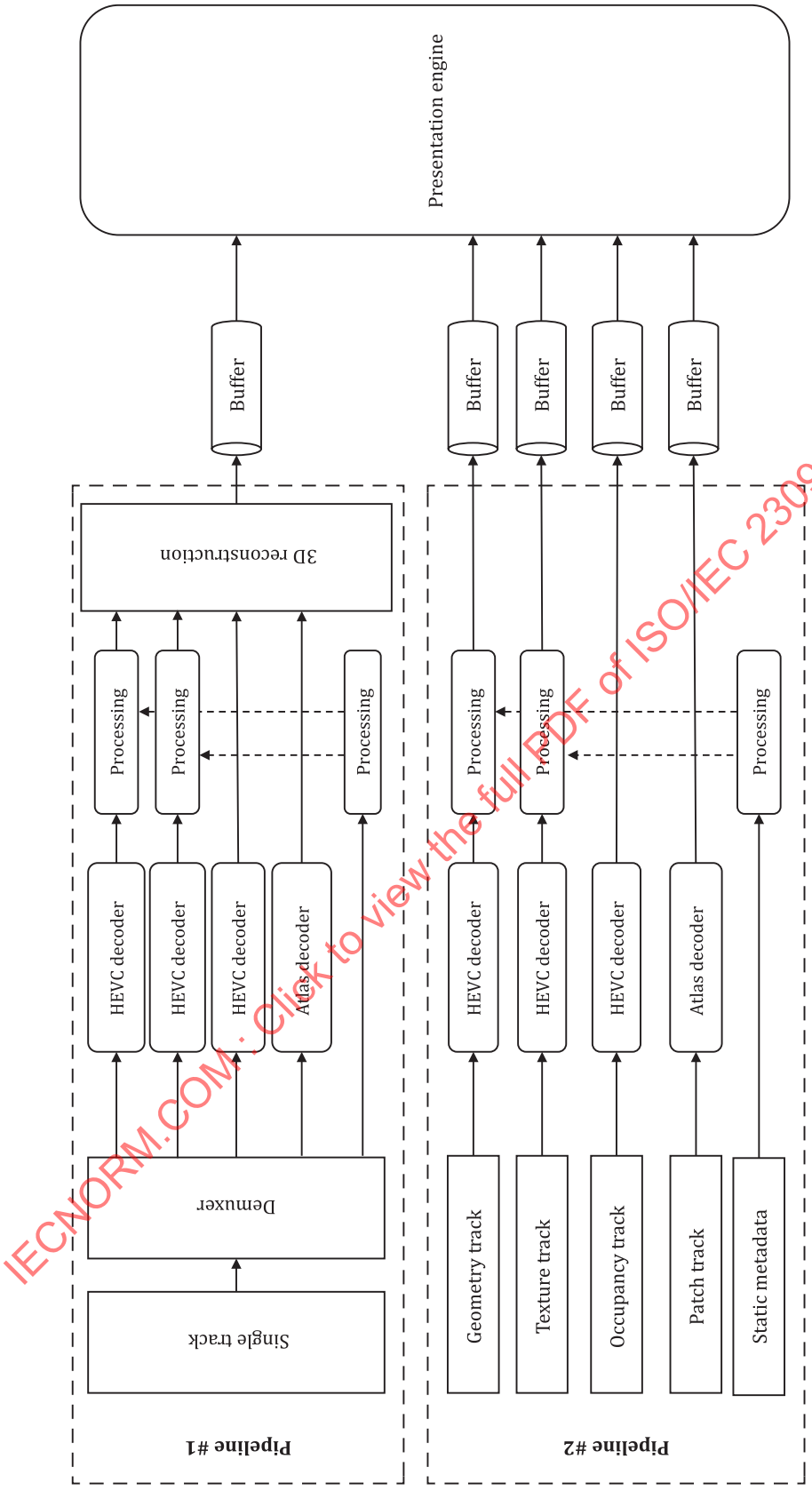

Figure 2 — An example of pipelines in scene description

4.3 Timing model

A scene and all contained nodes share a global common presentation timeline. An initial glTF document is used as an entry point for consuming a 3D scene. The scene activation time of that document may be set externally or may be determined by the user and is considered the presentation time T0 of the 3D scene. Each media is started at time T0 + T_{init}, where T_{init} is equal to startTime, when present, or equal to the earliest time at which the media is available if autoplay is equal to true and autoplayGroup is not present, or the earliest time at which all media with the same autoplayGroup are available.

The first sample of each media consumed at T0 + T_{init} is the one with presentation time equal to startTimeOffset. The media is consumed up to the sample with presentation time equal to the endTimeOffset, when present, or up to the last sample present in the media. When loop is set to true, at each loop the timeline is increased by adding the endTimeOffset – startTimeOffset, when endTimeOffset is present, or duration – startTimeOffset, when endTimeOffset is not present.

When a scene is updated through patch document to a scene description document or a scene description document, the media timeline remains unchanged and continues to be evaluated in respect to the T0, i.e. the activation time of the initial glTF document used as an entry point for consuming the 3D scene.

Animations described by the scene description document may be controlled (e.g., activated, paused, stopped) through MPEG_animation_timing extension. The activation timing of control events is identified by the timing of a sample in a metadata track. Once an animation event is activated the timeline of the animation is determined by animation data in the scene description data and the information provided by the animation sample in the metadata track (e.g., speed, start_frame, end_frame).

All static media of a scene are assumed to be presented at time T0. Timed media shall start at the indicated T0 + T_{init}. An object that has timed media components, shall not be rendered until the indicated T0 + T_{init} of these components. T0 + T_{init} of all the timed media components of the same object shall be equal.

Any extensions that include new primitive attributes shall register the attributes in Annex B.

5 Scene description extensions

5.1 General

5.1.1 Overview of extensions

An extension mechanism that allows to extend glTF 2.0 with new capabilities is defined in ISO/IEC 12113. A glTF node may have an optional extensions property that lists the extensions that are used by this node. All extensions that are used in a glTF document shall be listed in the top-level extensionsUsed array object, while extensions that are required to correctly load/render the scene shall also be listed in the extensionsRequired array.

A number of extensions, listed in Table 2, that enable support for timed media, are specified in subclauses 5.2, 5.3, 5.4, and 5.5. Extensions can be defined under Vendor, EXT, KHR, or KHX namespaces. The extensions defined in this document are under the vendor-specific extension namespaces with an MPEG prefix. Examples of how to use the extensions are provided in Annex F.

Table 2 — ISO/IEC 12113 extensions defined in this document

Extension Name	Brief Description	Type	Subclause
MPEG_media	Extension for referencing external media sources.	Generic	5.2.1
MPEG_accessor_timed	An accessor extension to support timed media.	Generic	5.2.2

Table 2 *(continued)*

Extension Name	Brief Description	Type	Subclause
MPEG_buffer_circular	A buffer extension to support circular buffers.	Generic	5.2.3
MPEG_scene_dynamic	An extension to support dynamic scenes.	Generic	5.2.4
MPEG_texture_video	A texture extension to support video textures.	Visual	5.3.1
MPEG_mesh_linking	An extension to link two meshes and provide mapping information	Visual	5.3.2
MPEG_audio_spatial	Adds support for spatial audio.	Audio	5.4.1
MPEG_viewport_recommended	An extension to describe a recommended viewport.	Metadata	5.5.1
MPEG_animation_timing	An extension to control animation timelines.	Metadata	5.5.2

Figure 3 depicts the glTF 2.0 hierarchy that includes the extensions defined in this document.

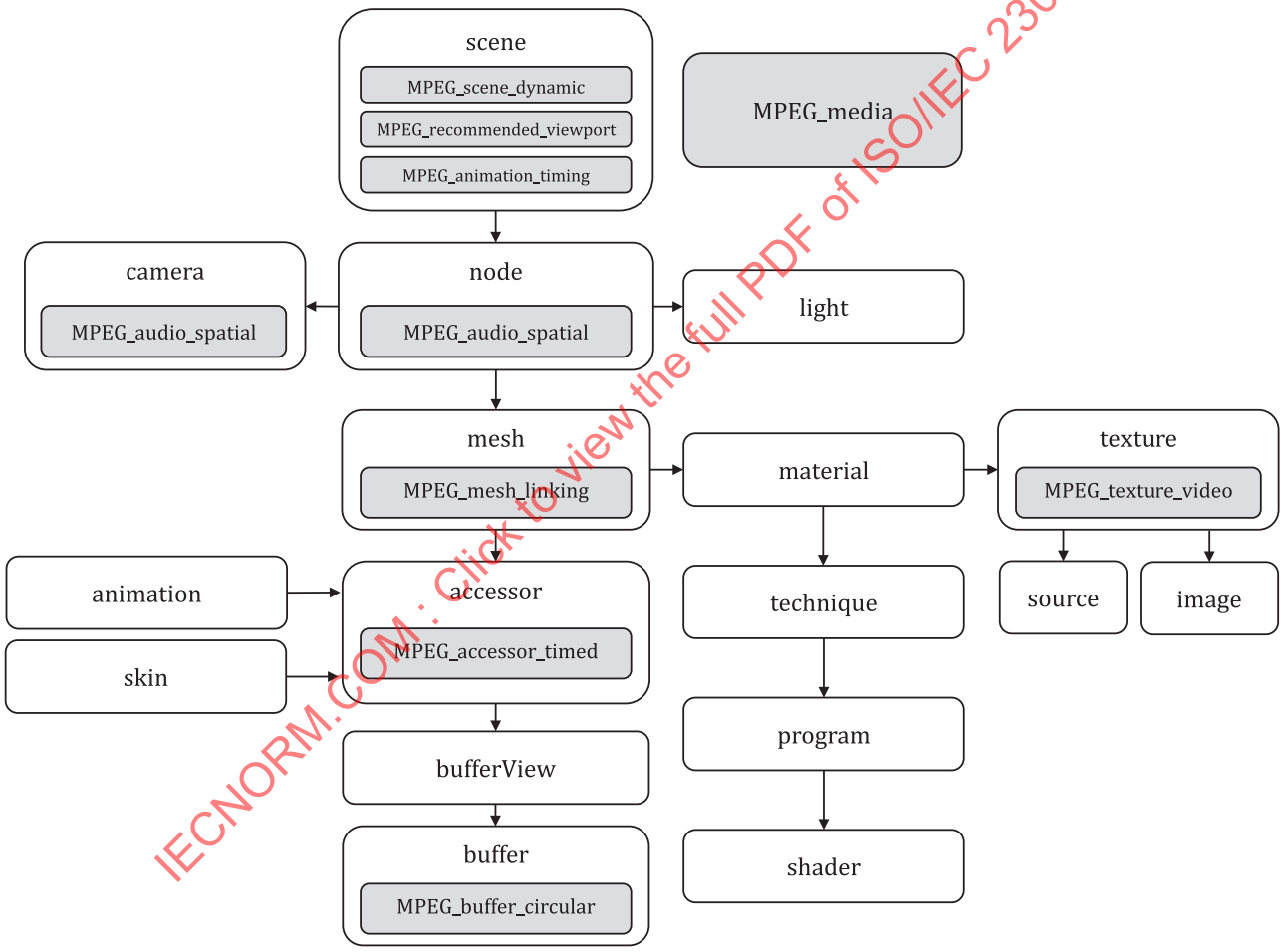

Figure 3 — An overview of the glTF document structure with MPEG extensions defined in this document

5.1.2 Formatting and typing

For binary data fields the following applies. The read_bits(n) function reads the next n bits from the buffer data and advances the data pointer by n bit positions. When n is equal to 0, read_bits(n) is specified to return a value equal to 0 and to not advance the data pointer.

The following types specify the types and parsing process for binary data fields:

— bits(n) fixed-pattern bit string using n bits written (from left to right) with the left bit first. The parsing process for this descriptor is specified by the return value of the function read_bits(n)

— bits(n)[m] array of m fixed-pattern bit strings with length of n bits.

— uint(n) unsigned integer using n bits. The parsing process for this descriptor is specified by the return value of the function read_bits(n) interpreted as a binary representation of an unsigned integer with the most significant bit written first.

— int(n) signed integer using n bits. The parsing process for this descriptor is specified by the return value of the function read_bits(n) interpreted as a two's complement integer representation with the most significant (left) bit written first. In particular, the parsing process for this type is specified as follows:

```
int(n) {
      value = read_bits( n )
      if( value < ( 1 << ( n - 1 ) ) )
         return value
      else
         return ( value | ~( ( 1 << (n - 1)) - 1 ) )
   }
```

— float(n) binary floating point value using n bits. The parsing process for this descriptor is as specified in IEEE 754-2019.

For JSON data fields, the following type definitions apply:

— number: primitive type defined in ISO/IEC 21778

— string: primitive type defined in ISO/IEC 21778

— boolean: primitive type defined in ISO/IEC 21778

— array: structured type defined in ISO/IEC 21778

— object: structured type defined in ISO/IEC 21778

5.2 Generic extensions

5.2.1 MPEG_media extension

5.2.1.1 General

The MPEG_media extension, identified by MPEG_media, provides an array of media items referenced in a scene description document.

When present, the MPEG_media extension shall be included as a top-level extension.

5.2.1.2 Semantics

The definition of all objects within MPEG_media extension is provided in Tables 3 to 6.

Table 3 — Definitions of top-level objects of MPEG_media extension

Name	Type	Default	Usage	Description
media	array	N/A	M	An array of items that describe the external media, referenced in this scene description document.

Table 4 — Definitions of item in the media array of MPEG_media extension

Name	Type	Default	Usage	Description
name	string	N/A	O	The user-defined name of the media.
startTime	number	0	O	The startTime gives the time at which the rendering of the timed media will begin. The value is provided in seconds. In the case of timed textures, the static image should be rendered as a texture until the startTime is reached. A startTime of 0 means the presentation time of the current scene. Either startTime or autoplay shall be present in glTF description.
startTimeOffset	number	0	O	The startTimeOffset indicates the time offset into the source, starting from which the timed media shall be generated. The value is provided in seconds, where 0 corresponds to the start of the source.
endTimeOffset	number	N/A	O	The endTimeOffset indicates the end time offset into the source, up to which the timed media shall be generated. The value is provided in seconds. If not present, the endTimeOffset corresponds to the end of the source media.
autoplay	boolean	True	O	Specifies that the media will start playing as soon as it is ready. Either startTime or autoplay shall be present for a media item description. Rendering of all media for which the autoplay flag is set to True should happen simultaneously.
autoplayGroup	integer	N/A	O	All media that have the same autoplayGroup identifier shall start playing synchronously as soon as all autoplayGroup media are ready. autoplayGroup is only allowed if autoplay is set to True.
loop	boolean	False	O	Specifies that the media will start over again, every time it is finished. The timestamp in the buffer shall be continuously increasing when the media source loops, i.e. the playback duration prior to looping shall be added to the media time after looping.
controls	boolean	False	O	Specifies that media controls should be displayed (such as a play/pause button etc).
alternatives	array	N/A	M	An array of items that indicate alternatives of the same media (e.g. different video codecs used) NOTE: the client can select items (i.e. uri and track) included in alternatives depending on the client's capability.

Table 5 — Definitions of items in the alternatives array of MPEG_media extension

Name	Type	Default	Usage	Description
mimeType	string	N/A	M	The media's MIME type. The profiles parameter, as defined in IETF RFC 6381, may be included as a part of the mimeType to specify the profile of the media container. (e.g. the profiles parameter indicates the DASH profile when the uri specifies a DASH manifest)
uri	string	N/A	M	The uri of the media. Relative paths are relative to the .gltf file. If the reference media is a real-time media stream, then the uri shall follow the referencing scheme as specified in Annex C. If the tracks element is present, the last part of the URI (i.e. the stream identifier such as the mid) is provided by the tracks information.
tracks	array	N/A	O	An array of items that lists the components of the referenced media source that are to be used. These can e.g. be a track number of an ISOBMFF, a DASH/CMAF SwitchingSet identifier, or a media id of an RTP stream.
extraParams	object	N/A	O	An object that may contain any additional media-specific parameters.

Table 6 — Definitions of items in the tracks array of MPEG_media.alternative extension

Name	Type	Default	Usage	Description
track	string	N/A	M	URL fragment to access the track within the media alternative. The URL structure is defined for the following formats: DASH: Using MPD Anchors (URL fragments) as defined in ISO/IEC 23009-1:2022, Annex C (Table C.1). ISOBMFF: URL fragments as specified in ISO/IEC 14496-12:2022, Annex C. SDP: stream identifier of the media stream as defined in Annex C. When V3C data is referenced in the scene description document as in item in MPEG_media.alternative.tracks and the referenced item corresponds to an ISBOBMFF track, the following applies: — For single-track encapsulated V3C data, the referenced track in MPEG_media shall be the V3C bitstream track. — For multi-track encapsulated V3C data, the referenced track in MPEG_media shall be the V3C atlas track.
codecs	string	N/A	M	The codecs parameter, as defined in IETF RFC 6381, of the media included in the track. When the track includes different types of codecs (e.g. the AdaptationSet includes Representations with different codecs), the codecs parameter may be signaled by comma-separated list of values of the codecs.

The JSON schema for the MPEG_media extension is provided in A.2.

5.2.1.3 Processing model

Processing of the MPEG_media extension depends on the referenced media. In general, media in the MPEG_media extension may be referenced by a circular buffer or by another extension defined in this document (e.g. MPEG_scene_dynamic). The Presentation Engine selects the media that is required at one of the available alternatives and is responsible for synchronization. The MAF instantiates the media fetching and two options exist. When the media in the MPEG_media extension is referenced by a circular buffer, the processing pipeline instantiated by the MAF decodes the media and reformats it to match the expected format by the Presentation Engine. This processing may also require setting some information about the header information from the MPEG_accessor_timed as defined in Table 8 appropriately based on the data included in the media. This might depend on the reference media and information in the scene description document (e.g. information in the accessor pointing to the circular buffer). When the media in the MPEG_media extension is referenced by another extension, the media is expected to be directly processed by the Presentation Engine.

5.2.2 MPEG_accessor_timed extension

5.2.2.1 General

An accessor specified in ISO/IEC 12113 defines the types and layout of the data as stored in a buffer that is viewed through a bufferView. When timed media is accessed in a buffer, the data in the buffer is expected to change dynamically with time. Timed accessor extension allows to describe access to dynamically changing data used in scene. The timed accessor is an extension to regular accessors to indicate that the underlying data buffer is dynamic.

Timed accessors may have two bufferViews, one inherited from the containing accessor and the second in the MPEG_accessor_timed extension. The former shall be used to reference the timed media data. The latter, when present, shall point to the timed accessor information header. The absence of a bufferView inside the MPEG_accessor_timed extension shall indicate that no timed accessor information header is present in the buffer. When both bufferViews are present, they shall point to the same buffer element. Accessors that include the "MPEG_accessor_timed" extension shall only point to buffers that include the "MPEG_buffer_circular" extension.

The accessor.bufferView field, in an accessor that has the MPEG_accessor_timed extension, as well as the timed accessor information header fields apply to the data of each frame within the circular buffer.

The timed accessor extension is identified by MPEG_accessor_timed. When present, the MPEG_accessor_timed extension shall be included as extension of an accessor object defined in ISO/IEC 12113.

5.2.2.2 Semantics

The definition of all objects within MPEG_accessor_timed extension is provided in Table 7.

Table 7 — Definition of MPEG_accessor_timed extension

Name	Type	Default	Usage	Description
immutable	boolean	True	O	This flag equal to false indicates the accessor information componentType, type, and normalize may change over time. The changing values of componentType, type and normalize are provided through accessor information header. This flag equal to true indicates the accessor information componentType, type, and normalize do not change over time and are not present in the accessor information header.
bufferView	integer	N/A	O	This property provides the index in the bufferViews array to a bufferView element that points to the timed accessor information header as described in Table 8. byteLength field of the bufferView element indicates the size of the timed accessor information header. The buffer properties in the bufferView element shall point to the same buffer as the bufferView in the containing accessor object. In the absence of the bufferView attribute, it shall be assumed that the buffer has no dynamic header. In that case, the immutable flag shall be present and shall be set to True.
suggestedUpdateRate	number	25.0	O	The suggestedUpdateRate provides the frequency at which the Presentation Engine is recommended to poll the underlying buffer for new data. The rate is provided in number of changes per second.

The timed accessor information header, when present, contains information required to properly access the media data in the buffer the accessor is pointing to. The timed accessor information header may change during the presentation of the scene. The timed accessor information header is provided as binary data as part of the buffer data and is accessible through the bufferView of the MPEG_accessor_timed extension.

Table 8 describes the syntax and semantics of the timed accessor information header.

Table 8 — Definition of timed accessor information header fields

timed_accessor_information_header() {	**Descriptor**
timestamp_delta	f(32)
if (!immutable) {	
componentType	u(32)
type	u(8)
normalized	u(1)
reserved_zero_bit	u(7)
}	
byteOffset	u(32)

Table 8 *(continued)*

count	u(32)
max	size(componentType)* components
min	size(componentType)* components
bufferViewByteOffset	u(32)
bufferViewByteLength	u(32)
bufferViewByteStride	u(32)
}	u(32)

timestamp_delta - provides a delta in seconds that is added to the timestamp field of the corresponding buffer frame in the referenced buffer to determine the timestamp of the referenced timed media. When accessor information header is not present, the value of timestamp_delta is inferred to be equal to 0. The sum of timestamp_delta and the timestamp field of the corresponding buffer frame shall be smaller than the timestamp field of any other following buffer frame in the buffer.

componentType - corresponds to the accessor property componentType as defined in ISO/IEC 12113.

type - The field correspond to the accessor properties type as defined in ISO/IEC 12113 with following modification:

— type equal to 0 indicates SCALAR as defined in ISO/IEC 12113.

— type equal to 1 indicates VEC2 as defined in ISO/IEC 12113.

— type equal to 2 indicates VEC3 as defined in ISO/IEC 12113.

— type equal to 3 indicates VEC4 as defined in ISO/IEC 12113.

— type equal to 4 indicates MAT2 as defined in ISO/IEC 12113.

— type equal to 5 indicates MAT3 as defined in ISO/IEC 12113.

— type equal to 6 indicates MAT4 as defined in ISO/IEC 12113.

normalized - corresponds to the accessor property normalized as defined in ISO/IEC 12113.

reserved_zero_bit - shall be equal to 0 in the timed accessor header information conforming to this version of this document. Other values are reserved for future use by ISO/IEC.

byteOffset - corresponds to the accessor property byteOffset as defined in ISO/IEC 12113.

count - corresponds to the accessor property count as defined in ISO/IEC 12113.

max - corresponds to the accessor property max as defined in ISO/IEC 12113. The max array sizes depend on the number of components as defined by the type defined in ISO/IEC 12113.

min - corresponds to the accessor property min as defined in ISO/IEC 12113. The min array sizes depend on the number of components as defined by the type defined in ISO/IEC 12113.

bufferViewByteOffset - corresponds to the bufferView property byteOffset defined in ISO/IEC 12113.

bufferViewByteLength - corresponds to the bufferView property byteLength defined in ISO/IEC 12113.

bufferViewByteStride - corresponds to the bufferView property byteStride fields defined in ISO/IEC 12113.The size( ) function returns the number of bits for a given componentType as defined by the Accessor Data Types table in ISO/IEC 12113.

The fields bufferViewByteOffset, bufferViewByteLength, and bufferViewByteStride update information of the bufferView referenced by the accessor containing the MPEG_accessor_timed extension and they provide a description of how to access the corresponding media data in the buffer.

The JSON schema for the MPEG_accessor_timed extension is provided in A.3.

5.2.2.3 Processing model

For timed and dynamic data access, the MPEG_accessor_timed shall be used to describe access to the timed data. The Presentation Engine shall extract the information about the sample format from the accessor and, when present, the configuration of the header information from the MPEG_accessor_timed and pass it to the MAF. The MAF shall provide the information in the requested format in the buffers.

The timed accessor information header shall be present when at least some of the accessor information is dynamic. The presence of a timed accessor information header shall be signalled by the presence of the bufferView attribute in the MPEG_accessor_timed extension.

If present, the MAF shall insert the timed accessor information header prior to the corresponding data in the buffer frame. The offset, length, and stride of the data in the buffer frame may change from buffer frame to buffer frame and shall be signalled as part of the timed accessor information header.

A buffer shall not mix data from dynamic and static components, i.e. components that have a timed accessor information header and other components that do not have a timed accessor information header.

The Presentation Engine shall overwrite the accessor and buffer view information view that are described or referenced by an accessor that has the MPEG_accessor_timed extension, which itself includes a bufferView reference. In other words, the dynamic accessor and buffer view information take precedence over the static accessor and buffer view information present in the scene description.

5.2.3 MPEG_buffer_circular extension

5.2.3.1 General

In order to support timed data access, the buffer element is extended to provide functionality of a circular buffer. The extension is named MPEG_buffer_circular and may be included as part of the "buffers" structures. Buffers that provide access to timed data shall include the MPEG_buffer_circular extension.

When MPEG_buffer_circular extension is present in a buffer element, the buffer element property uri shall not be present and the buffer element property byteLength shall indicate the maximum possible size of the buffer that may be needed to accommodate for the number of buffer frames indicated by count value.

When present, the MPEG_buffer_circular extension shall be included as extension of a buffer object defined in ISO/IEC 12113.

5.2.3.2 Semantics

The definition of all objects within MPEG_buffer_circular extension is provided in Table 9.

Table 9 — Definition of MPEG_buffer_circular extension

Name	Type	Default	Usage	Description
count	integer	2	O	The count field provides the recommended number of sequential buffer frames to be offered by a circular buffer to the presentation engine. This information may be used by the MAF to setup the circular buffer towards the Presentation Engine.

Table 9 *(continued)*

Name	Type	Default	Usage	Description
media	integer	N/A	M	Index of the media entry in the MPEG_media extension, which is used as the source for the input data to the buffer.
tracks	array	N/A	O	Index of a track of a media entry, indicated by media and listed by MPEG_media extension, used as the source for the input data to this buffer. When tracks element is not present, the media pipeline should perform the necessary processing of all tracks of the MPEG_media entry, referenced by the media property, to generate the requested data format of the buffer. When tracks array contains multiple tracks, the media pipeline should perform the necessary processing of all referenced tracks to generate the requested data format of the buffer. If the track attribute is present and there are multiple "alternatives" (i.e. indicating equivalent content) in the referenced media, then the selected track shall be present in all alternatives. NOTE: When more than one track is listed by tracks element, the corresponding buffer is in active state and the MAF is informed that the corresponding tracks are needed as source for the input buffer, then the MAF can optimize the delivery of multiple tracks.

The JSON schema for the MPEG_buffer_circular is provided in A.4.

5.2.3.3 Processing model

Frames of the buffer may differ in length based on the amount of data for each frame. A read and a write pointer are maintained for each circular buffer. By default, read and write access to the buffer will be served from the frame that is referenced by the read or write pointer respectively. Access to a particular frame index or timestamp should be supported.

The frames are read at the read pointer for rendering. New incoming frames from one or more media decoders are inserted at the write pointer. When present (i.e. when bufferView is included in the MPEG_accessor_timed extension of an accessor referencing this buffer) the timed accessor information header is included in the buffer frame as required, i.e. as indicated by the corresponding timed accessor. Prior data in that frame will be overwritten and the frame buffer should be resized accordingly.

Figure 4 depicts the buffer structure:

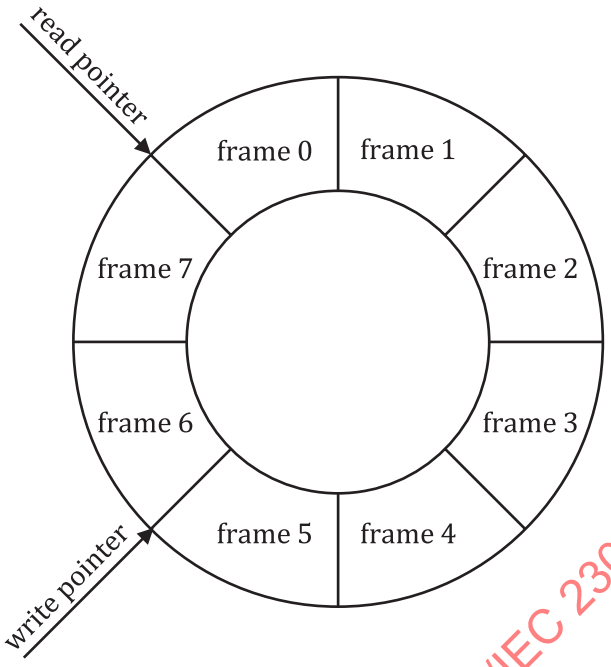

Figure 4 — An example of circular buffer operation with count value equal to 8

The buffer management ensures that Timestamp(write_pointer) > Timestamp(read_pointer) where Timestamp(pointer) is a function that returns the timestamp assigned to the buffer frame associated with that pointer. When overwriting existing data in a buffer frame with new data, the buffer management ensures that the read_pointer is moved to a buffer frame with the earliest timestamp in the buffer. This may result in data drop but it ensures that no concurrent read and write access to the same buffer frame is performed.

5.2.4 MPEG_scene_dynamic extensions

5.2.4.1 General

MPEG_scene_dynamic extension allows to indicate that the scene description document may be updated. MPEG_scene_dynamic extension points to MPEG_media extension element that contains URL information to access scene document updates. For example, scene document updates may be provided as samples of a track or items as defined in Clause 7.

Scene updates shall be expressed as a scene description document or as a patch document using the JSON Patch protocol as defined in IETF RFC 6902. ISOBMFF-based carriage format for both scene description documents and patch documents using JSON patch protocol is specified in subclause 7.2. The glTF extensions MPEG_media and MPEG_scene_dynamic shall be used in order to expose the dynamic scene updates, as described in subclause 5.2.1 and 5.2.4.

After successfully performing an update operation, the resulting scene graph shall be consistent, i.e. syntactically valid and all references shall be correct. ISO/IEC 12113 uses the order of elements for referencing, therefore particular care should be used with update operations that change the order of elements in the graph, such as `move` and `remove` operations. The client shall update all references after every successful scene document update operation.

When a patch document contains update to nodes that does not match any node of the active scene document, the update command of this node shall be discarded.

When present, the MPEG_scene_dynamic extension shall be included as extension of a scene object defined in ISO/IEC 12113.

5.2.4.2 Semantics

The extension MPEG_scene_dynamic links to one of the entries listed in MPEG_media. The definition of all objects within MPEG_scene_dynamic extension is provided in Table 10.

Table 10 — Definition of top-level objects of MPEG_scene_dynamic extension

Name	Type	Default	Usage	Description
media	integer	N/A	M	Provides the index of the media described in the MPEG_media extension and which will contain the scene update data.
track	integer	N/A	O	Provides the index of a track of a media object, referenced by media attribute and listed by MPEG_media extension. The track samples contain scene description updates and provide timing to perform these updates. If track is not provided, it shall be assumed that all tracks provided by the referenced media object are used to provide the update samples.

The JSON schema for the MPEG_scene_dynamic extension is provided in A.5.

5.2.4.3 Processing model

The Presentation Engine parses the scene description document and maintains a representation of the scene graph in memory. The Presentation Engine receives scene description updates together with scene activation times. For example, the scene description updates can be provided as samples of an ISOBMFF track, as specified in Clause 7. In that case the scene activation time of a scene description update is indicated by the presentation time of the corresponding sample. When the sample becomes active, the Presentation Engine shall load the sample data and trigger the scene update to be performed. The scene description updates themselves modify the scene graph representation in memory and it can add new media to the scene. The timing for the newly added or updated media is determined by the metadata in the updated scene description document. If the scene description update contains an insertion of new glTF nodes and/or potential modifications to existing glTF nodes, the Presentation Engine should fetch any new content, e.g. using the MAF, associated with the scene description update and present the new content accordingly.

Each update operation shall either consist of a JSON Patch document for partial updates or a scene description document for a complete update. All update operations of a JSON Patch document shall be considered as a single timed transaction.

When a patch document contains update to nodes that does not match any node of active scene document, the update command of this node shall be discarded. When all update commands have been processed or discarded, the update operation shall be considered completed.

The fetching of updates and the activation of certain nodes may be triggered by different factors including the following:

— Wallclock time

— Presentation time

— Interaction event

For live presentations, it is expected that presentation of the newly added glTF objects (e.g., new live media and potentially other dynamic objects) included in the scene during the scene updates will be synchronized with the scene presentation timeline via timing information (e.g., timestamps, etc.) included in the corresponding media formats and containers.

5.3 Visual Extensions

5.3.1 MPEG_texture_video extensions

5.3.1.1 General

MPEG texture video extension, identified by MPEG_texture_video, provides the possibility to link a texture object defined in ISO/IEC 12113 to a video source. The MPEG_texture_video extension provides a reference to the timed accessor, i.e. accessor with MPEG_accessor_timed extension, where the decoded timed texture will be made available.

When present, the MPEG_texture_video extension shall be included as extension of a textures object defined in ISO/IEC 12113.

When the MPEG_texture_video extension is not supported, the standard texture glTF element can be used as fallback.

NOTE When MPEG_texture_video extension is supported, the decoded video frames are used as textures starting from the time indicated in the corresponding MPEG_media.media.startTime. If MPEG_media.media.startTime is greater than 0, the static image frame indicated be texture.source element can be used until MPEG_media.media.startTime.

5.3.1.2 Semantics

The definition of all objects within MPEG_texture_video extension is provided in Table 11.

Table 11 — Definition of top-level objects of MPEG_texture_video extension

Name	Type	Default	Usage	Description
accessor	integer	N/A	M	Provides a reference to the accessor, by specifying the accessor's index in accessors array, that describes the buffer where the decoded timed texture will be made available. The accessor shall have the MPEG_accessor_timed extension. The type, componentType, and count of the accessor depend on the width, height, and format.
width	integer	N/A	M	Provides the maximum width of the texture.
height	integer	N/A	M	Provides the maximum height of the texture.
format	string	RGB	O	Indicates the format of the pixel data for this video texture. The allowed values are: RED, GREEN, BLUE, RG, RGB, RGBA, BGR, BGRA, DEPTH_COMPONENT. The semantics of these values are defined in Table 8.3 of OpenGL® [a] specification.[2] Note that the number of components shall match the type indicated by the referenced accessor. Normalization of the pixel data shall be indicated by the normalized attribute of the accessor.

[a] OpenGL® is the trademark of a product supplied by Khronos. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of the product named. Equivalent products may be used if they can be shown to lead to the same results.

The JSON schema for the MPEG_texture_video extension is provided in A.6.

5.3.1.3 Processing model

When meshes, or any other element in scene description document, reference a texture element that contains MPEG_texture_video it gets the texture data provided in the buffer as described by the

accessor indicated in the MPEG_texture_video extension. A new frame from the buffer should be read for each rendering cycle. If no new frame is available, the previous frame should be presented. The texture data stored into the buffer shall have the format indicated by the format attribute indicated by the MPEG_texture_video extension. When the picture size output by the media decoder are different from the size indicated by the width and height indicated by the MPEG_texture_video extension, two approaches are possible, when storing the decoded texture samples into the circular buffer. The first approach consists of storing the decoded texture samples into the buffer frames at the width and height indicated by the MPEG_texture_video extension. This requires that the size of the texture provided by the media decoder is reformatted, for example by MAF. The second approach consists of storing the decoded texture samples into the buffer frames at the width and height output by the decoder. In this case, the actual width and height of the picture shall be provided to the renderer, e.g. through the buffer frame header as specified in Table 24 if MAF is used.

NOTE Some implementations require the texture data stored within the buffer to have a static size as they cannot properly handle dynamically changing texture data formats. This requires the MAF or media decoders to generate the texture data at a static format and make it available at the buffers. On the other hand, other implementations can benefit of performing the size conversion by themselves in a more efficient manner and can cope with dynamically changing picture sizes in the underlying buffer. Derived specifications can mandate to operate in one of the two specified modes.

5.3.2 MPEG_mesh_linking extensions

5.3.2.1 General

MPEG mesh linking extension, identified by MPEG_mesh_linking, provides the possibility to link a mesh to another mesh in a glTF asset.

The shadow mesh corresponds to regular mesh data as defined by ISO/IEC 12113 without the MPEG_mesh_linking extension. The dependent mesh can be transformed/animated by relying on the shadow mesh. The MPEG_mesh_linking extension included into the dependent mesh links the dependent mesh and the shadow mesh and provides with the data and information which is used to achieve the ability to animate the dependent mesh. Hence, the shadow mesh is present in the glTF assets to assist in achieving the ability to apply transformation onto the dependent mesh.

When present, the MPEG_mesh_linking extension shall be included as extension of a mesh object defined in ISO/IEC 12113.

When the MPEG_mesh_linking extension is not supported, the dependent mesh can be rendered as a regular timely updated mesh sequence of frames.

NOTE A reasonable backup when the extension is not understood is to render the dependent mesh without performing any animation. In such a case, the shadow mesh is not rendered and therefore is not added as a node in the scene.

5.3.2.2 Semantics

The definition of all objects within MPEG_mesh_linking extension is provided in Table 12.

Table 12 — Definition of top-level objects of MPEG_mesh_linking extension

Name	Type	Default	Usage	Description
correspondence	integer	N/A	M	Provides a reference to the accessor, by specifying the accessor's index in accessors array, that describe the buffer where the correspondence values between the dependent mesh and its associated shadow mesh will be made available. The componentType of the referenced accessor shall be as indicated in subclause 7.4 and the type shall be SCALAR.

Table 12 *(continued)*

Name	Type	Default	Usage	Description
mesh	integer	N/A	M	Provides a reference to the shadow mesh, by specifying the mesh index in meshes array, associated to the dependent mesh for which the correspondence values are established.
pose	integer	N/A	M	Provides a reference to the accessor, by specifying the accessor's index in accessors array, that describe the buffer where the transformation of the nodes associated to the dependent mesh will be made available. The componentType of the referenced accessor shall be FLOAT and the type shall be MAT4.
weights	integer	N/A	O	Provides a reference to the accessor, by specifying the accessor's index in accessors array, that describe the buffer where the "weights" to be applied to the morph targets of the shadow mesh associated to the dependent mesh will be made available. The componentType of the referenced accessor shall be FLOAT and the type shall be SCALAR.

The JSON schema for the MPEG_mesh_linking extension is provided in A.7.

5.3.2.3 Processing model

The processing model could be as follows:

The corresponding shadow mesh for a dependent mesh is identified as provided by "mesh" in the MPEG_mesh_linking extension of the dependent mesh.

At runtime the Presentation Engine reads the corresponding frames from the circular buffers that the accessors with MPEG_accessor_timed extension point to as indicated by "correspondence", "pose" and "weights". The Presentation Engine glues each vertex of the dependent mesh to a face of the shadow mesh indicated by the correspondence value for the particular position and pose of the dependent mesh at that time instant. The render engine records the distance of each vertex of the dependent mesh to the plane of its corresponding shadow mesh face and records the position of the point onto which the vertex of the dependent mesh is projected within the associated face of the shadow mesh, i.e. the point within the face to which the distance is computed. With this parametrization between the two meshes, a transformation of the shadow mesh can directly be transferred to the dependent mesh.

The shadow mesh indicated as defined in ISO/IEC 12113 can transformed by means of using mesh primitives for skinning and pose-dependent morph targets.

The first step consists of linking the shadow mesh and the dependent mesh at the current pose of the dependent mesh. For this purpose, the shadow mesh is transformed to the same position and pose as the dependent mesh as provided by the data in the frames in the circular buffer corresponding to "pose" and "weights" in the MPEG_mesh_linking extension. This transformation is performed as any other transformation by means of using mesh primitives for skinning and pose-dependent morph targets. Then, the correspondence values for each of the vertices in the dependent mesh, as provided by the frames in the circular buffer corresponding to correspondences, indicating a mapping to a face of the shadow mesh is used to determine the relative location of each vertex in the dependent mesh to the associated face of the shadow mesh as explained above and in more detail in Annex E.

With the relative locations representing the linked meshes, as a second step the shadow mesh at its original position and pose is transformed as indicated by animations. With the shadow mesh at the target position and pose, the dependent mesh is transformed by following the relative locations of each vertex with respect the associated faces of the shadow mesh, explained in more detail in Annex E.

5.4 Audio extensions

5.4.1 MPEG_audio_spatial extensions

5.4.1.1 General

The MPEG audio extension adds support for spatial audio.

When present, the MPEG_audio_spatial extension shall be included as extension of a camera object, or a scene object defined in ISO/IEC 12113.

The MPEG_audio_spatial extension supports three different node types:

— source: an audio source that provides input audio data into the scene. Mono objects and HOA sources (as defined in ISO/IEC 23008-3:2022, Annex F.1) are supported in this version of the document.

  — Type: 'Object' or 'HOA'

  — HOA audio sources shall ignore the parent node's position and be rendered only in 3DoF.

— reverb: A reverb effect can be attached to the output of an audio source. Several reverb units can exist and sound sources can feed into one or more of these reverb units. An audio renderer that does not support reverb shall ignore it if the bypass attribute is set to true. If the bypass attribute is set to false, the audio renderer shall return an error message

— listener: An audio listener represents the output of audio in the scene. A listener should be attached to a camera node in the scene. By being a child node of the camera, additional transformations can be applied to the audio listener relative to the transformation applied to the parent camera.

Figure 5 depicts the processing chain for audio in a scene.

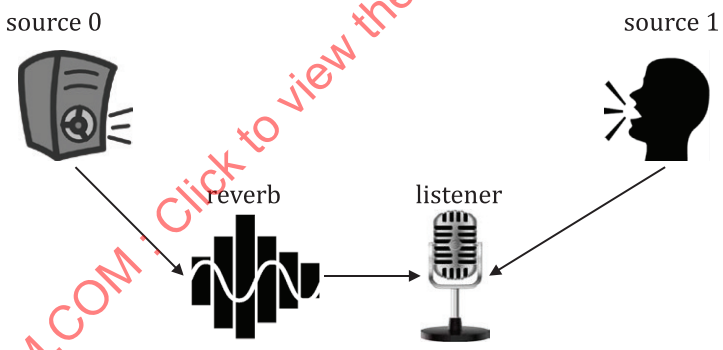

Figure 5 — An example of the processing chain for audio in a scene

The specification of any audio effect processing is outside the scope of this document.

The characteristics of a listener depend on the actual output devices available to the audio renderer.

5.4.1.2 Semantics

The definition of all objects within MPEG_audio_spatial extension is provided in Tables 13 to 17.

Table 13 — Definition of top-level objects of MPEG_audio_spatial extension

Name	Type	Default	Usage	Description
sources	array	N/A	O	an array of source objects that are attached to the current node.
listener	object	N/A	O	a listener object that places an audio listener node in the scene that should be attached to a parent camera node. The audio listener characteristics depend on the available audio output devices.
reverbs	array	N/A	O	an array of reverb objects.

Table 14 — Definition of source object of MPEG_audio_spatial.source extension

Name	Type	Default	Usage	Description
id	integer		M	unique identifier of the audio source in the scene.
type	string		M	Indicates the type of the audio source. type value equal to “Object” indicates mono object type value equal to “HOA” indicates HOA object
pregain	number	0.0	O	provides a level-adjustment in dB for the signal associated with the source.
playbackSpeed	number	1.0	O	defines the playback speed of the audio signal. A value of 1.0 corresponds to playback at normal speed. The value shall be between 0.5 and 2.0.
attenuation	enumeration	“linearDistance”	O	indicates the function used to calculate the attenuation of the audio signal based on the distance to the source. attenuation value equal to ""“noAttenuation" indicates no attenuation function should be used. attenuation value equal to "inverseDistance" indicates inverse distance function should be used. attenuation value equal to "linearDistance" indicates linear distance function should be used. attenuation value equal to "exponentialDistance" indicates exponential distance function should be used. attenuation value equal to "custom" indicates custom function should be used. The definition of custom function is outside of the scope of this document. The attenuation functions and their parameters shall be as specified in Annex D.
attenuationParameters	array	N/A	O	array of parameters that are input to the attenuation function. The semantics of these parameters depend on the attenuation function itself.

Table 14 *(continued)*

Name	Type	Default	Usage	Description
referenceDistance	number	1.0	O	provides the distance in meters for which the distance gain is implicitly included in the source signal after application of pregain. When type equals 'HOA' the element shall not be present.
accessors	array	N/A	M	provides an array of accessor references, by specifying the accessors indices in accessors array, that describe the buffers where the decoded audio will be made available.
reverbFeed	array	N/A	O	provides one or more pointers to reverb units, optionally extended by a floating point scaling factor. A reverb unit represents a reverberation audio processor that is configured by the metadata from a single reverb object. Typically, a reverb object represents reverberation properties of a single room.
reverbFeedGain	array	N/A	O	provides an array of gain [dB] values to be applied to the source's signal(s) when feeding it to the corresponding reverbFeed. The array shall have the same number of elements as the reverbFeed array field.

Table 15 — Definition of listener objects of MPEG_audio_spatial.listener extension

Name	Type	Default	Usage	Description
id	integer	N/A	M	unique identifier of the audio listener in the scene.

Table 16 — Definition of reverb object of MPEG_audio_spatial.reverb extension

Name	Type	Default	Usage	Description
id	integer	N/A	M	unique identifier of the audio reverb unit in the scene.
bypass	boolean	True	O	indicates if the reverb unit can be bypassed if the audio renderer does not support it.
properties	array		M	Array of items that contains reverbProperties objects describing reverb unit specific parameters
predelay	number	0.0	O	Delay [seconds] from onset of source to onset of late reverberation for which DSR is provided.

Table 17 — Definition of reverbProperty object of MPEG_audio_spatial extension

Name	Type	Default	Usage	Description
frequency	number	N/A	M	Frequency for the provided RT60 and DSR values.
RT60	number	N/A	M	RT60 values (in seconds) for the frequency provided in the 'frequency' field.
DSR	number	N/A	M	Diffuse-to-Source Ratio value [dB] for the frequency provided in the 'frequency' field. See explanatory text in subclause 5.4.1.3.

The JSON schema for the MPEG_audio_spatial extension is provided in A.8.

5.4.1.3 Processing model

The 60 dB reverberation time, short RT60, is defined as the time it takes for the sound pressure level in a room to reduce by 60 dB, measured after a generated steady-state test signal is abruptly ended. It is defined for a specific frequency as an attribute RT60 and specified in seconds.

The pre-delay time indicates the delay between the emission at the source and the onset of the diffuse late reverberation part of a signal (i.e. the sound after the early reflections) and is specified in seconds. It is frequency-independent.

The Diffuse-to-Source-Ratio (DSR) specifies the level of the diffuse reverberation relative to the level of the total emitted sound. This can be determined while making an RT60 measurement. It is defined for a specific <frequency> as an attribute DSR and can be computed when dividing the total diffuse reverb energy by the total emitted energy.

For example, a value of 0 indicates direct sound only, while large values will describe an almost completely reverberant (wet) acoustic environment. Note that the DSR values do not influence the amplitude of the direct sound in the process of rendering. While DSR is a general description of a room's acoustic properties, rendering reverberation using DSR requires taking into account the source's directivity pattern to find the total emitted energy from the PCM signal's reference level. DSR values are independent of directivity and may be determined with a source of any directivity, e.g. an omni-directional source. The total diffuse reverb energy denotes the reverberation energy at any point in the region for which the acoustic environment is defined and is therefore directly linked to the PCM signal's reference level.

5.5 Metadata extensions

5.5.1 MPEG_viewport_recommended extensions

5.5.1.1 General

Recommended viewport extension, identified by MPEG_viewport_recommended, provides dynamically changing recommended viewport information which includes translation and rotation as well as the intrinsic camera parameter of the camera object. The client may render the viewport according to the dynamically changing information.

When present, the MPEG_viewport_recommended extension shall be included as extension of a scene object defined in ISO/IEC 12113.

NOTE Another approach to achieve recommended viewport is to define an animation for a node with attached camera. The approach, however, does not support dynamically changing intrinsic camera and can only be defined during the creation of glTF object.

5.5.1.2 Semantics

The definition of all objects within MPEG_viewport_recommended extension is provided in Table 18.

Table 18 — Definition of MPEG_viewport_recommended extension

Name	Type	Default	Usage	Description
name	string	N/A	O	Label of the recommended viewport
translation	integer	N/A	O	Provides a reference to accessor where the timed data for the translation of camera object will be made available. The componentType of the referenced accessor shall be FLOAT and the type shall be VEC3, (x, y, z).

Table 18 *(continued)*

Name	Type	Default	Usage	Description
rotation	integer	N/A	O	Provides a reference to accessor where the timed data for the rotation of camera object will be made available. The componentType of the referenced accessor shall be FLOAT and the type shall be VEC4, as a unit quaternion, (x, y, z, w).
type	string	"perspective"	O	provides the type of camera.
parameters	integer	N/A	O	Provides a reference to a timed accessor where the timed data for the perspective or orthographic camera parameters will be made available. The componentType of the referenced accessor shall be FLOAT and the type shall be VEC4. When the type of the camera object which includes this extension is perspective, FLOAT_VEC4 means (aspectRatio, yfov, zfar, znear). When orthographic type, FLOAT_VEC4 means (xmag, ymag, zfar, znear). The requirements on the camera parameters from ISO/IEC 12113 shall apply.

The JSON schema for the MPEG_viewport_recommended extension is provided in A.9.

5.5.1.3 Processing model

When a scene contains MPEG_viewport_recommended extension, renderer should manipulate camera object parameters and position based on data provided in the buffers described by the accessors indicated in the MPEG_viewport_recommended extension.

NOTE This document does not specify how the data is transmitted or made available at the respective circular buffers. A possible approach is that the MPEG_media extension includes a media corresponding to a metadata track carrying the necessary information, which would be made available at the right format in the referenced timed accessors.

5.5.2 MPEG_animation_timing extensions

5.5.2.1 General

Animation timing extension, identified by MPEG_animation_timing, enables alignment between media timelines and animation timelines defined by ISO/IEC 12113. Using the extension narrated stories can be created. The animation timing metadata allows simultaneous pausing and other manipulation of animations defined in ISO/IEC 12113 and timed media. By manipulating the global timeline for narrated content, the animation defined in ISO/IEC 12113 and timed media can be manipulated as well.

When present, the MPEG_animation_timing extension shall be included as extension of a scene object defined in ISO/IEC 12113.

5.5.2.2 Semantics

The definition of all objects within MPEG_animation_timing extension is provided in Table 19.

Table 19 — Definition of MPEG_animation_timing extension

Name	Type	Default	Usage	Description
accessor	integer	N/A	M	Provides a reference to the accessor, by specifying the accessor's index in accessors array, that describes the buffer where the animation timing data will be made available. The sample format shall be as defined in subclause 7.6.3. The componentType of the referenced accessor shall be BYTE and the type shall be SCALAR.

The JSON schema for the MPEG_animation_timing extension is provided in A.10.

5.5.2.3 Processing model

When the MPEG_animation_timing extension is present in the scene description document, the Presentation Engine should adjust the glTF animation state based on the data provided in the buffer referenced by the accessor attribute of the MPEG_animation_timing extension.

6 Media access function and buffer API

6.1 General

Scene description document may be an entry point for an immersive media application. In such case the Presentation Engine receives the scene description document or a URL to a scene description document, which is downloaded.

The Presentation Engine parses the JSON-formatted scene description to build a scene graph in memory. It then iterates through all objects in the node and determines the associated media sources and their buffer formats and timing information.

The Presentation Engine then initializes the Media Access Function. For each attribute of each object, it requests the creation of a media pipeline for the processing of the corresponding media source. The Presentation Engine may choose either to pass a buffer handler for an existing circular buffer to the MAF or it may rely on the MAF to allocate the circular buffer.

The Presentation Engine then invokes the startFetching operation for the media pipeline ahead of the requested presentation time of the corresponding object.

The reference Media Access Function API and the Buffer API to create the media pipelines is provided in subclauses 6.2 and 6.3, respectively. Figure 6 shows a high-level overview of a typical processing procedure.

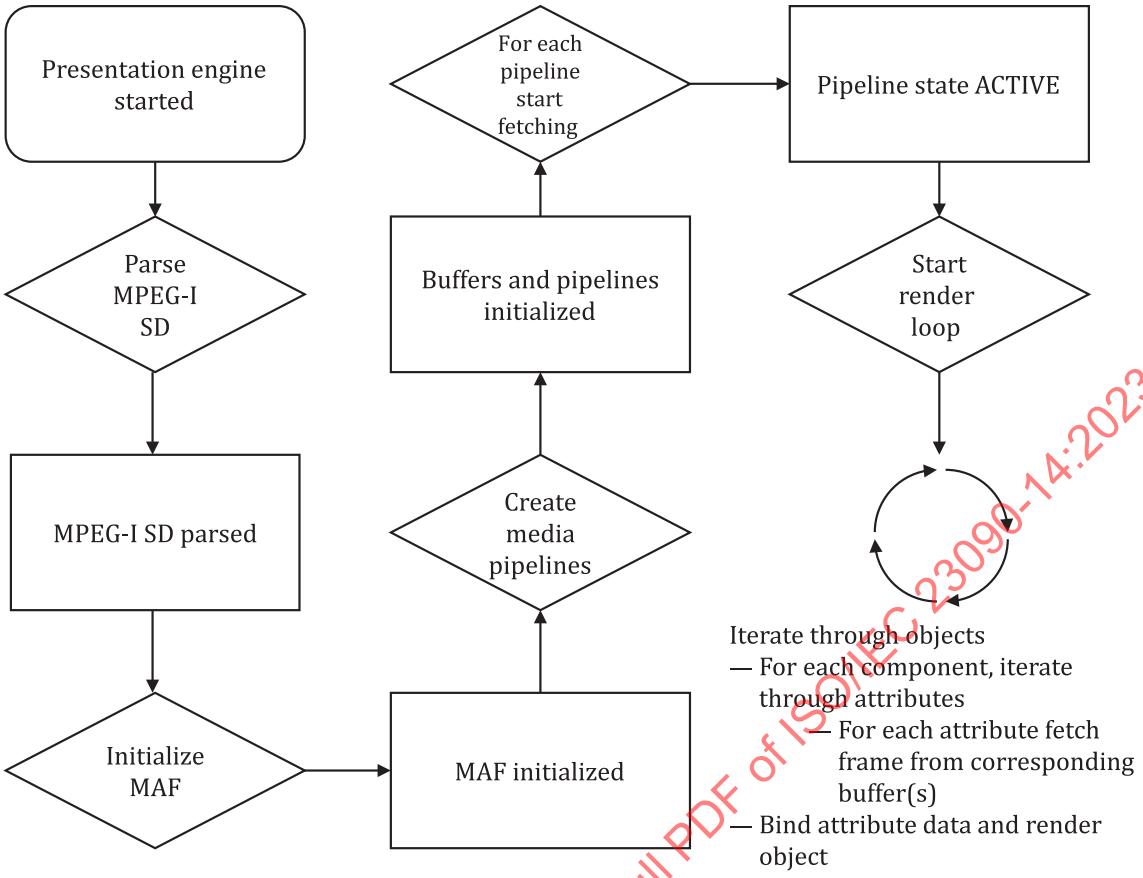

Figure 6 — High-level overview of a scene description processing procedure

6.2 Media access function API

The MAF API is an API defined by this document. A Media Access Function as defined in this document shall support the MAF API to interface with the Presentation Engine.

The following methods are offered through API defined in Table 20.

Table 20 — Description of MAF API

Method	State after Success	Description
initialize( )	READY	The Presentation Engine initializes a new media pipeline. It provides information related to the requested media or metadata. The MAF will setup the media pipeline and allocate the buffers, if they have not been allocated by the Presentation Engine.
startFetching( )	ACTIVE	Once initialized and in READY state, the Presentation Engine may request the media pipeline to start fetching the requested data.
updateView()	ACTIVE	update the current view information. This function is called by the Presentation Engine to update the current view information, if the pose or object position have changed significantly enough to impact media access. It is not expected that every pose change will result in a call to this function.

Table 20 *(continued)*

Method	State after Success	Description
stopFetching( )	READY	The Presentation Engine may request to stop data fetching through this media pipeline. If subsequently, startFetching is called again, the stream fetching will resume from the current point, unless a start time is provided.
destroy( )	IDLE	Finally, the Presentation Engine may request to destroy this media pipeline and free any associated resources.

The pipeline interface defined using the IDL syntax specified in ISO/IEC 19516 is as follows:

```
interface Pipeline {
   readonly attribute Buffer buffers[];
   readonly attribute PipelineState  state;
   attribute EventHandler onstatechange;
   void initialize(MediaInfo mediaInfo, BufferInfo bufferInfo[]);
   void startFetching(TimeInfo timeInfo, ViewInfo viewInfo);
   void updateView(ViewInfo viewInfo);
   void stopFetching();
   void destroy();
};
```

The data types defined using the IDL syntax specified in ISO/IEC 19516 are as follows:

```
interface MediaInfo {
   attribute String name;
   attribute AlternativeLocation alternatives;
};

interface AlternativeLocation {
   attribute String mimeType;
   attribute Track tracks[];
   attribute uri;
};

interface Track {
   attribute String track;
   attribute integer id;
   attribute integer bufferId;
};

interface TimeInfo {
   attribute double startTime;
   attribute double timeOffset;
   attribute boolean autoplay;
   attribute boolean loop;
};

interface BufferInfo {
   attribute integer bufferId;
   attribute BufferHandle handle;
   attribute unsigned long componentType;
   attribute SampleType type;
   attribute integer offset;
   attribute integer stride;
   attribute AttributeType attributeType;
};

interface ViewInfo {
   attribute Pose pose;
   attribute Camera camera;
   attribute Transform objectTransform;
};

interface Pose {
   attribute Position position
```

```
   attribute Quaternion orientation;
};

interface Camera {
   readonly attribute CameraProjectionType type;
   readonly attribute PerspectiveCameraViewingVolume;
   readonly attribute OrthographicCameraViewingVolume;
   readonly attribute double zNear;
   readonly attribute double zFar;
};

enum CameraProjectionType {"PERSPECTIVE", "ORTHOGRAPHIC"};

interface PerspectiveCameraViewingVolume {
   double aspectRatio;
   double yFov;
};

interface OrthographicCameraViewingVolume {
   double xmag;
   double ymag;
};

typedef Transform float[4][4];

typedef SampleType {"SCALAR", "VEC2", "VEC3", "VEC4", "MAT2", "MAT3", "MAT4"};

enum AttributeType {"ATTRIB_NORMAL","ATTRIB_POSITION","ATTRIB_COLOR", "ATTRIB_
TEXCOORD","ATTRIB_INDEX","ATTRIB_TANGENT", "ATTRIB_WEIGHTS", "ATTRIB_JOINTS"}
```

Semantics of componentType and type corresponds to the semantics of accessor properties componentType and type, respectively, defined in ISO/IEC 12113.

Table 21 provides semantics for the defined interfaces and their parameters:

Table 21 — API Structure and Parameter Semantics

Parameter Name	Description
Pipeline	Provides a representation of a media pipeline that stores its current pipeline state PipelineState and keeps track of the output buffers of that pipeline. The PipelineState may be IDLE, READY, ACTIVE, or ERROR. A pipeline in IDLE state means it has not been initialized yet. The READY state indicates that the pipeline has been initialized and is ready to start fetching media. When in ACTIVE state, the pipeline is actively fetching media. The ERROR state indicates that the pipeline has encountered an error that stopped the media access. The EventHandler holds a pointer to a callback function, which will be called upon a change in the PipelineState.
MediaInfo	The MediaInfo carries information about the location of the media that the pipeline is to access. The MediaInfo carries the same information as provided by the MPEG_media extension. A MediaInfo may be assigned a name. It has to provide at least one location in the alternatives array. Each alternative contains a MIME type, a set of tracks, and the URI that can be used to access the media in that alternative. The tracks indicate which streams/tracks/representations from the referenced media alternative are to be accessed by this pipeline. If none is specified, it shall be assumed that all components of the referenced media alternative are to be accessed.
TimeInfo	The TimeInfo indicates the time point at which the media is to be accessed and when to start the media access. The semantics of each field is identical to that provided in the MPEG_media extension.

Table 21 *(continued)*

Parameter Name	Description
BufferInfo	The BufferInfo provides information about the expected format of the output buffer of the media pipeline. The buffer may be allocated by the MAF or by the Presentation Engine. The handler is used to read or pass a reference to an allocated buffer. If allocated by the MAF, then only read access shall be allowed. If the pipeline is initialized without a valid buffer handler, then the MAF shall allocate the buffer. the buffer format information in componentType, type, offset, and stride shall correspond to componentType, type, offset, stride in the corresponding accessor and bufferView. The attributeType shall correspond to the corresponding primitive attribute. The BufferHandle holds a handle for a buffer, which can be used to access the buffer.
ViewInfo	The ViewInfo represents the current positions of the object for which the media is accessed and the viewer's pose. This information is useful to adjust the media access to the visibility of the object. For example, a far object may be accessed at a lower Level of Detail (LoD). The ViewInfo.pose contains the pose information of the viewer, provided as a position and oritentation. ViewInfo.camera provides also the intrinsic properties of the camera used by the Presentation Engine. The information about the intrinsic properties includes the type of camera projection. (e.g., perspective, orthographic, etc.) and related parameters. A camera may have intrinsic parameters such as distance to near and far clipping planes, aspect ratio, y-FoV (for perspective projection camera), and width and height of the viewing volume (for orthographic projection camera). The ViewInfo.objectTransform contains the position and orientation of the object as a transform (a 4x4 matrix as defined by ISO/IEC 12113). All information shall use the scene's coordinate system.

The MAF may use the ViewInfo to optimize the streaming of the requested media, e.g. by adjusting the level of detail (number of polygons/points, texture resolution etc.) based on the distance to and orientation of the viewer. The BufferInfo contains information about each Buffer and describes the format of the samples and frames that are stored in that buffer. One or more tracks from the MediaInfo may feed into the same buffer. The link between the track that provides the actual media and the buffer that will store the output of the media pipeline is established through the bufferId attribute.

6.3 Buffer API

The Buffer API is used by the Presentation Engine and the MAF to allocate and control buffers for the exchange of data between the Presentation Engine and the MAF through media pipelines.

The Buffer API offers the methods indicated in Table 22.

Table 22 — Description of buffer API

Method	Description
allocate()	Allocates a buffer for the data exchange between the MAF and the Presentation Engine.
writeFrame()	writes a frame to the buffer. The write pointer moves to the next frame.
readFrame()	reads a frame from the buffer. If no timestamp is provided, the buffer pointer moves to the next frame. Otherwise, the read pointer remains unchanged. If the buffer is empty, an error is returned. Prior frames may still be accessed using their timestamp as long as they are not overwritten.
free()	Destroys the buffers and frees any resources associated with it.

When allocating a buffer, sufficient information is provided about the buffer configuration. This includes the maximum size of the buffer, the static information in the buffer header, the number of frames in the buffer for circular buffers, and the update rate of the buffer.

The Buffer API interface defined using the IDL syntax specified in ISO/IEC 19516 is as follows:

```
interface CircularBuffer {
   readonly attribute Frame frames[];
   readonly attribute count;
   readonly attribute integer read_idx;
   readonly attributre integer write_idx;
   attribute integer headerLength;
   attribute EventHandler   onframewrite;
   attribute EventHandler   onframeread;
   void allocate(int count);
   void writeFrame(Frame frame);
   Frame readFrame(optional double timestamp);
   void free();
};

interface Frame {
   attribute integer index;
   attribute double timestamp;
   attribute integer length;
   attribute octet[length] data;
};
```

Table 23 provides semantics for the defined interfaces and their parameters:

Table 23 — API structure and parameter semantics

Parameter Name	Description
count	The number of frames contained in this circular buffer. Each frame of the buffer will hold data at a particular time instance and will be identified by an index in the range of [0, count-1]. The index, timestamp and length of the frame are signaled as the frame metadata.
read_idx	The index of the frame that can be read. If the read_idx is equal to the write_idx, then it shall be assumed that there is currently no frame available to be read.
write_idx	The index of the frame at which a write operation can be performed.
headerLength	This provides the length in bytes of the buffer frame header that is available at the start of every frame in the buffer as specified in Table 24. A headerLength of 0 indicates that there is no buffer header. When the buffer API is used, the buffer frame header shall not be present as part of the buffer frame data. For implementations that do not use the Buffer API, the buffer frame header information may be provided as part of the buffer frame data.
Frame	provides information about a frame in the buffer. The index is the position of the frame in the buffer. The timestamp corresponds to its presentation timestamp. The length corresponds to the length of the buffer. The semantics of the Buffer Frame fields are provided in Table 24. Note that the timestamp format is inherited from the accessed media and as such can be an NTP timestamp, a 0-offset timestamp, or any other format.

Table 24 — Syntax and semantics of the buffer frame header

Syntax		Length (bits)	Type	Semantics
index		8	uint(8)	The index of the current buffer frame. The index is a value between 0 and count -1.
timestamp		64	uint(64)	Provides the base timestamp of the data that is contained in this buffer frame. Together with the timestamp_delta in the timed accessor information header, when present, it provides the timestamp of the data within the buffer frame. The 32 most significant bits represent the seconds part and the 32 least significant bits represent the fraction of a second part. The timestamp field is not necessarily a wallclock time and the interpretation of this field is left to the Presentation Engine.
length		32	uint(32)	The length of the data of this buffer frame, including the accessor information header fields defined in Table 8.
extra_frame_info_flags		8	uint(8)	A set of flags that indicate additional information related to this buffer frame.
if (extra_frame_info_flags & 0X01 == 1)				If the dimensions flag is set, then the buffer frame contains texture data and the related width and hight parameters are provided.
	width	32	uint(32)	The width of any texture data stored within the buffer frame
	height	32	uint(32)	The height of any texture data stored within the buffer frame

7 Carriage formats

7.1 General

Clause 7 describes the carriage formats related to scene description. The scene description documents and scene description updates may be stored as samples of a track in ISOBMFF as defined in subclause 7.2, or as items as defined in subclause 7.3. Carriage for a mesh correspondence, pose and weights that are utilized by MPEG_mesh_linking extension is defined in subclauses 7.4 and 7.5. Carriage for an animation timing, that is utilized by MPEG_animation_timing extension, is defined in subclause 7.6.

Figure 7 shows an example relationship between items and tracks stored as one file. In the example JSON glTF file and a file representing glTF binary buffer are stored as items of a single file. The same file also contains tracks that can be reference from glTF file using the extensions defined in this document, such as MPEG_media, MPEG_buffer_cirrcular and MPEG_scene_dynamic.

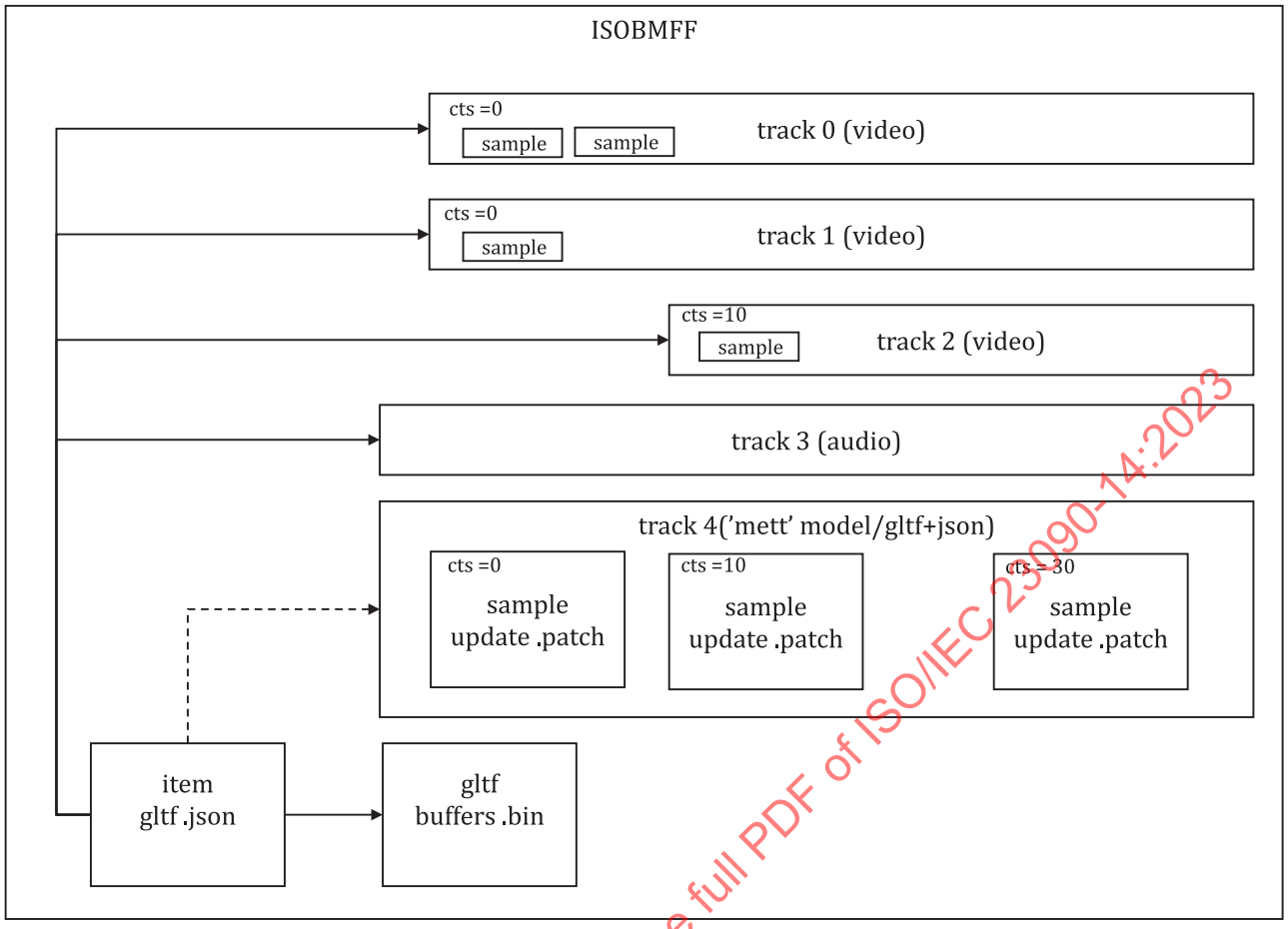

Figure 7 — Data relationship between the glTF .json stored as item and patch updates as a track.

7.2 Carriage format for glTF JSON and JSON patch

7.2.1 General

A track with samples containing glTF JSON documents and JSON patch document shall be stored as metadata media defined in ISO/IEC 14496-12 and shall fulfil the following condition:

— `'meta'` handler type shall be used in the **`HandlerBox`** of the **`MediaBox`**.

— The sample entry format shall be `'mett'` and:

  — **`mime_format`** field shall be set to model/gltf+json,

  — **`content_encoding`** field when present shall contain either an empty string or a value allowed in HTTP's Content-Encoding header.

  — **`GLTFPatchConfigBox`** may be present in the sample entry.

— Samples containing glTF JSON documents shall be marked as a sync sample and shall use UTF-8 encoding as defined in IETF RFC 8259

— Samples containing JSON patch documents shall not be marked as sync sample and shall use UTF-8 encoding as defined in IETF RFC 8259

— Samples may have the **`sample_has_redundancy`** flag set to 1, in which case processing is applied as discussed in subclause 7.7.

The presentation time of a sample identifies the scene activation time for scene resulting from loading glTF JSON document or resulting from applying the JSON patch document contained in the sample as indicated by update_mode in the GLTFPatchConfigBox to the active scene description document. When the GLTFPatchConfigBox is not present non-sync samples shall be applied as specified by update_mode equal to 0.

NOTE The update of the scene description document as well as the fetching of the assets of possible new nodes to be rendered are expected to take place sufficiently ahead of time so that the presentation of the scene can continue without interruption.

7.2.2 glTF patch config box

7.2.2.1 Definition

Box Type:	'glTC'
Container:	Sample Entry ('mett')
Mandatory:	No
Quantity:	0 or 1

The glTF configuration box provides information how to process the samples of tracks with samples contain glTF JSON documents and JSON patch documents.

7.2.2.2 Syntax

```
class GLTFPatchConfigBox() extends FullBox ('gltC', 0, 0) {
   unsigned int(3) update_mode;
   unsigned int(5) reserved;
}
```

7.2.2.3 Semantics

update_mode specifies how to determine the target file associated with the patch documents contained in the non-sync samples of the track. When update_mode is set to 0, the patch document contained in a non-sync sample of the track is to be applied to the glTF JSON document obtained by the processing of the previous sample (sync or non-sync) in decoding order, if any, or to the original glTF file including the metadata track into the MPEG_media extensions (e.g., glTF Object as non-timed item). When update_mode is set to 1, the patch document contained in a non-sync sample is to be applied to the glTF JSON document contained in the previous sync sample in decoding order, if any, or to the original glTF file including the metadata track into the MPEG_media extensions (glTF Object as non-timed item). Values from 2 to 7 are reserved for future use.

7.3 Carriage format for glTF object and glTF source object as non-timed item

7.3.1 General

glTF can be stored as items, when there is no specific time associated to the loading of that resource or when the track storage is not appropriate. As defined in ISOBMFF, items are declared in a **MetaBox** which may be present in the movie header or in movie fragment headers.

Items carrying glTF may serve either as an entry point to the file (e.g. as determined by the application using the ISOBMFF file or when the ISOBMFF is loaded with a URL with fragment identifier identifying that item) or as secondary content loaded by either other items or track samples.

7.3.2 glTF Items

The brand `'glti'` may be used to signal the use of a `MetaBox` with the following constraints:

— It shall be present at the file level.

— It shall use a `HandlerBox` with the handler_type set to `'gltf'`

— It shall contain a `PrimaryItemBox` which declares as primary item a resource of type 'model/gltf+json'.

— It shall not use any `DataInformationBox`, `ItemProtectionBox` or `IPMPControlBox`.

— It shall use a `ItemInfoBox 'iinf'` with the following constraints:

  — its version is either 0 or 1;

  — each item is described by an `ItemInfoEntry 'infe'` with the following constraints:

    — its version is set to 0;

    — its `item_protection_index` is set to 0;

    — if the item is referred to by a URL in the content of another item, its `item_name` is equal to that URL.

— It shall use an `ItemLocationBox 'iloc'` with the following constraints:

  — its version is set to 1 or 2;

  — each item is described by an entry and values 0, 1 or 2 may be used for the construction method.

— It may use any other boxes (such as `ItemReferenceBox 'iref'`) not explicitly excluded above.

— It may contain a `GroupsListBox` with a `EntityToGroupBox` with the grouping type `'gltf'`, containing an array of `entity_id` which is resolved to this item and to all the tracks in this file referenced by the glTF object stored in this item.

NOTE This entity grouping helps file parsers to understand the relationships between the item and the tracks present in the file without having to parse the glTF file in the item.

7.3.3 glTF source items

A glTF Source Object is a scene description source document and can be carried as s glTF Source Item. The brand `'glsi'` may be used to signal the use of a `MetaBox` with the following constraints:

— It shall be present at the track level, the containing track shall comply to subclause 7.2.

— It shall use a `HandlerBox` with the handler_type set to `'glsi'`

— It shall contain a `PrimaryItemBox` which declares as primary item a resource of type `'application/json'`.

— It shall not use any `DataInformationBox`, `ItemProtectionBox` or `IPMPControlBox`.

— It shall use a `ItemInfoBox 'iinf'` with the following constraints:

  — its version is either 0 or 1;

  — each item is described by an `ItemInfoEntry 'infe'` with the following constraints:

    — its version is set to 0;

    — its `item_protection_index` is set to 0;

  - if the item is referred to by a URL in the content of another item, its **item_name** is equal to that URL.

- It shall use an **ItemLocationBox 'iloc'** with the following constraints:
  - its version is set to 1 or 2;
  - each item is described by an entry and values 0, 1 or 2 may be used for the construction method.
- It may use any other boxes (such as **ItemReferenceBox 'iref'**) not explicitly excluded above.
- It may contain a **GroupsListBox** with a **EntityToGroupBox** with the grouping type **'gltf'**, containing an array of **entity_id** which is resolved to this item and to all the tracks in this file referenced by the glTF object stored in this item.

7.4 Carriage format for mesh correspondence values

7.4.1 General

A sample in a metadata track is used to provide the correspondence values that are used to map one mesh, i.e. dependent mesh, to another mesh, i.e. shadow mesh. The sample timing of the metadata track defines the time instant of a mesh sample to which the correspondence value applies. The sample format itself contains an integer value that identifies the indexed face of the shadow mesh to which the corresponding vertex applies.

In glTF JSON file, the correspondence values are provided through an extension of a mesh by referring to an accessor with MPEG_accessor_timed extension. The number of correspondence values within a sample and the number of vertices of the corresponding mesh shall be the same.

7.4.2 Vertices correspondence sample entry

7.4.2.1 Definition

Sample Entry Type: 'vcor'

Container: Sample Description Box ('stsd')

Mandatory: No

Quantity: 0 or 1

A vertex correspondence entry identifies a track containing vertex correspondence samples.

7.4.2.2 Syntax

```
aligned(8) class VertexCorrespondenceSampleEntry()
   extends MetadataSampleEntry('vcor') {
   unsigned int(3) precision;
   bits(5) reserved;
}
```

7.4.2.3 Semantics

precision specifies the length in bytes of the correspondence values within each sample. The value of precision shall be greater than 0 and smaller or equal to 4.

7.4.3 Vertices correspondence sample format

7.4.3.1 General

The sample format includes the number of correspondence values, as well as the value itself that applies to each of the vertices.

7.4.3.2 Syntax

```
aligned(8) class VerticesCorrespondenceSample
{   unsigned int(32) vertex_count;
    for( i = 0; i < vertex_count; i++ ){
    unsigned int(8 * precision) face_index[ i ];
    }
}
```

7.4.3.3 Semantics

vertex_count specifies the number of vertices of the mesh this sample applies to and the number of **face_index**[i] values present in the sample

face_index[i] specifies the index of a face in a shadow mesh to which a vertex with index equal i of dependent mesh is mapped.

7.4.3.4 Processing of correspondence samples

The value of precision in VertexCorrespondenceSampleEntry is constraint as follows. When precision is equal to 1, the componentType of the accessor referenced by correspondence attribute in the corresponding mesh with the MPEG_mesh_linking extension shall be equal to UNSIGNED_BYTE (5121). Otherwise, when precision is equal to 2, the componentType of the accessor referenced by correspondence attribute in the corresponding mesh with the MPEG_mesh_linking extension shall be equal to UNSIGNED_SHORT (5123). Otherwise, when precision is equal to 3 or 4, the componentType of the accessor referenced by correspondence attribute in the corresponding mesh with the MPEG_mesh_linking extension shall be equal to UNSIGNED_INT (5125).

The samples VerticesCorrespondenceSample are processed as follows. **vertex_count** shall be made available as "count" in the corresponding timed accessor information header fields in the corresponding frame of the circular buffer, which the accessor with the MPEG_accessor_timed indicated by correspondence attribute in the MPEG_mesh_linking extension points to. **face_index**[i] shall be made available in the corresponding frame of the circular buffer, which the accessor with the MPEG_accessor_timed indicated by correspondence attribute in the MPEG_mesh_linking extension points to using the componenType specified above.

7.5 Carriage format for pose and weight

7.5.1 General

A sample in a metadata track is used to provide the pose transformation used onto the shadow mesh and weight values that are used to apply the morph targets of the shadow mesh to transform the shadow mesh to the corresponding position and pose of the dependent mesh. The sample timing of the metadata track defines the time instant of a mesh sample to which the pose and weight information applies. The sample format itself contains the transformations matrix to be applied onto the nodes of the skeleton of the mesh and "weight" values to apply the morph targets of the shadow mesh.

In glTF JSON file, the specified values are provided through an extension of a mesh by referring to an accessor with MPEG_accessor_timed extension. The number of "weight" values within a sample and the number of morph target of the corresponding shadow mesh shall be the same.

7.5.2 Pose transformation sample entry

7.5.2.1 Definition

Sample Entry Type: 'post'

Container: Sample Description Box ('stsd')

Mandatory: No

Quantity: 0 or 1

A pose transformation sample entry identifies a track containing pose transformation samples.

7.5.2.2 Syntax

```
aligned(8) class PoseTransformationSampleEntry( )
extends MetadataSampleEntry( 'post' ) {
   unsigned int(16) number_of_nodes;
   for( unsigned int i = 0; i < number_of_nodes; i++ ){
      unsigned int (32) node_index[ i ];
   }
   unsigned int(16) number_of_morph_targets;
}
```

7.5.2.3 Semantics

number_of_nodes specifies the number of nodes for which the transformation is described.

node_index[i] specifies the index of the nodes array in the glTF file which the i-th node in the sample applies to.

number_of_morph_targets specifies the number of morph targets for which a weight is provided.

7.5.3 Pose transformation sample format

7.5.3.1 General

The sample format includes the transformations for each node, as well as weights to be applied to the associated morph targets.

7.5.3.2 Syntax

```
aligned(8) class PoseTransformationSample {
   for( i = 0; i < number_of_nodes; i++ ){
      float(32) [16] matrix[ i ];
   }
   for( i = 0; i < number_of_morph_targets; i++ ){
      float(32) weight[ i ];
   }
}
```

7.5.3.3 Semantics

matrix[i] specifies the transformation matrix of the i-th node.

weight[i] specifies the weight to be applied to the i-th morph target.

7.5.3.4 Processing of pose transformation samples

The `PoseTransformationSampleEntry` is processed as follows. When immutable is set to True and bufferView is not present the accessor within the MPEG_accessor_timed extension indicated by pose attribute or weights attribute in the MPEG_mesh_linking extension, the value of count in the corresponding accessor shall be equal to number_of_nodes or number_of_morph_targets, respectively. Otherwise, the syntax element number_of_nodes shall be made available as count attribute in the corresponding timed accessor information header fields in the corresponding frame of the circular buffer, which the accessor with the MPEG_accessor_timed indicated by pose attribute in the MPEG_mesh_linking extension points to, and syntax element number_of_morph_targets shall be made available as count attribute in the corresponding timed accessor information header fields in the corresponding frame of the circular buffer, which the accessor with the MPEG_accessor_timed indicated by weights attribute in the MPEG_mesh_linking extension points to.

The samples in `PoseTransformationSample` are processed as follows. matrix[i] shall be made available in the corresponding frame of the circular buffer, which the accessor with the MPEG_accessor_timed indicated by pose attribute in the MPEG_mesh_linking extension points to. Similarly, weight[i] shall be made available in the corresponding frame of the circular buffer, which the accessor with the MPEG_accessor_timed indicated by weight attribute in the MPEG_mesh_linking extension points to.

7.6 Carriage format for animation timing

7.6.1 General

A sample in a metadata track is used to manipulate an animation event defined in the glTF JSON file. The sample timing of the metadata track defines, when in the global timeline (i.e. common for audio/video/animation) the animation should be manipulated. The metadata track may be stored in the ISOBMFF file along with other media, which provides utility to align manipulations of glTF animations with the video and audio tracks.

The default duration of the animation would be defined by the animation data in the binary glTF buffer and not by the sample duration of ISOBMFF.

7.6.2 Animation sample entry

7.6.2.1 Definition

Sample Entry Type: 'glat'

Container: Sample Description Box ('stsd')

Mandatory: No

Quantity: 0 or 1

An animation sample entry identifies an animation track containing glTF animation samples.

7.6.2.2 Syntax

```
aligned(8) class glTFAnimationSampleEntry()
   extends MetadataSampleEntry('glat') {
}
```

7.6.3 Animation sample format

7.6.3.1 General

The sample format includes controlling parameters for animations defined in the glTF animations array.

7.6.3.2 Syntax

```
aligned(8) class glTFAnimationSample
{
   unsigned int(1) apply_to_all;
   unsigned int(7) reserved;
   unsigned int(16) num_events;
   for( i=0; i < num_events; i++ ){
      unsigned int(32) index[i];
      int(32) speed[i];
      unsigned int(8) state[i];
      if( state == 6 ){
         unsigned int (32) start_frame[i];
         unsigned int (32) end_frame[i];
      }
      unsigned int (8) order_id[i];
      unsigned int(32) num_channels[i];
      for( int j = 0; j < num_channels[i]; j++) {
         int (8) weight[i][j];
         unsigned int (32) channel_index[i][j];
      }
   }
}
```

7.6.3.3 Semantics

apply_to_all if equal to 1, the num_events shall equal 1 and the animation event in the sample is applied to all animations in the glTF animations array.

num_events – specifying number of animation events triggered at the time of the sample.

index[i] specifies the index value of animation in animation node described in the glTF json file.

speed[i] specifies a multiplier which indicate the speed of the playout of the animation. A negative value may indicate that the animation should be played in a reverse order, from the end to the start. The speed is stored as signed 15.16 fixed-point value.

state[i] specifies status of the animation as defined in Table 25.

start_frame[i] specifies the key frame of the animation used after each loop.

end_frame[i] specifies the last the key frame of the animation before looping the animation.

order_id[i] specifies a value to indicate the order in which animations are applied. Animations with lower values are applied before animation with higher values.

num_channels[i] specifies the number of channels of an animation for which a weight is provided.

weight[i][j] specifies the weight to be applied to the j-th channel of the animation in units of 1/255.

channel_index[i][j] specifies the index of the j-th channel of the animation.

Table 25 — Definition of the state values in animation sample format

state value	identifier	description
0	play	Play the animation
1	stop_at_initial	Stop the animation and return to the initial state
2	stop_at_final	Stop the animation and keep the final state
3	pause	Pause animation
4	restart	Restart the animation, equivalent to stopping animation and playing it from the beginning.