 ISO/IEC JTC 1/SC 29/WG 3 N00026

**ISO/IEC JTC 1/SC 29/WG 3**

**MPEG Systems   
Convenorship: KATS (Korea, Republic of)**

**Document type:** Output Document

**Title:** Text of ISO/IEC CD 23090-14 Scene Description for MPEG Media

**Status:** Approved

**Date of document:** 2020-11-30

**Source:** ISO/IEC JTC 1/SC 29/WG 3

**Expected action:** ACT

**Action due date:** 2020-12-07

**No. of pages:** 45 (with cover page)

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

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

**ISO 23090-14:2020(X)**

ISO TC JTC/SC 29/WG 03

Secretariat: JISC

**Information technology** — **Coded representation of immersive media** —**Part 14: Scene Description for MPEG Media**

CD stage

**Warning for WDs and CDs**

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

*To help you, this guide on writing standards was produced by the ISO/TMB and is available at* [*https://www.iso.org/iso/how-to-write-standards.pdf*](https://www.iso.org/iso/how-to-write-standards.pdf)

*A model manuscript of a draft International Standard (known as “The Rice Model”) is available at* [*https://www.iso.org/iso/model\_document-rice\_model.pdf*](https://www.iso.org/iso/model_document-rice_model.pdf)

© ISO 2020

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO’s member body in the country of the requester.

ISO copyright office

CP 401 • Ch. de Blandonnet 8

CH-1214 Vernier, Geneva

Phone: +41 22 749 01 11

Fax: +41 22 749 09 47

Email: copyright@iso.org

Website: www.iso.org

Published in Switzerland

Contents

[Foreword vi](#_Toc57885472)

[Introduction vii](#_Toc57885473)

[1 Scope 1](#_Toc57885474)

[2 Normative references 1](#_Toc57885475)

[3 Terms, definitions, symbols and abbreviations 1](#_Toc57885480)

[3.1 Terms and definitions 1](#_Toc57885481)

[3.2 Abbreviations 1](#_Toc57885482)

[3.3 Conventions 2](#_Toc57885483)

[4 Overview and Architecture 2](#_Toc57885484)

[4.1 Overview 2](#_Toc57885485)

[4.2 glTF 2.0 Extension Mechanisms 3](#_Toc57885486)

[4.2.1 Normative Aspects 3](#_Toc57885487)

[4.3 Architecture 3](#_Toc57885488)

[4.3.1 General Architecture 3](#_Toc57885489)

[4.3.2 Pipelines 6](#_Toc57885490)

[4.4 Data and Timing Model 7](#_Toc57885491)

[5 Scene Description Extensions 7](#_Toc57885504)

[5.1 Overview 7](#_Toc57885505)

[5.2 Generic Extensions 8](#_Toc57885518)

[5.2.1 MPEG\_media extension 8](#_Toc57885519)

[5.2.2 MPEG\_accessor\_timed extension 13](#_Toc57885582)

[5.2.3 MPEG\_buffer\_circular extension 16](#_Toc57885602)

[5.2.4 MPEG\_scene\_dynamic extensions 18](#_Toc57885935)

[5.3 Visual Extensions 20](#_Toc57885953)

[5.3.1 MPEG\_texture\_video extensions 20](#_Toc57885954)

[5.4 Audio Extensions 21](#_Toc57885982)

[5.4.1 MPEG\_audio\_spatial extensions 21](#_Toc57885983)

[5.5 Metadata Extensions 26](#_Toc57885984)

[5.5.1 MPEG\_viewport\_recommended extensions 26](#_Toc57885985)

[5.5.2 MPEG\_animation\_timing extensions 29](#_Toc57885986)

[6 Media Access and Buffer API 31](#_Toc57885987)

[6.1 Media Access API 31](#_Toc57885988)

[6.2 Buffer API 32](#_Toc57885989)

[7 Scene Processing Model 33](#_Toc57885991)

[7.1 General 33](#_Toc57885992)

[7.2 Scene Updates 33](#_Toc57885993)

[8 Carriage Formats 34](#_Toc57885994)

[8.1 General 34](#_Toc57885995)

[8.2 JSON Patch Document Carriage Format for Scene Updates 34](#_Toc57885996)

[Annex A (normative) 36](#_Toc57886000)

[Bibliography 37](#_Toc57886001)

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

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 ISO documents 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](https://www.iso.org/directives-and-policies.html)).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see [www.iso.org/patents](https://www.iso.org/iso-standards-and-patents.html)).

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](https://www.iso.org/foreword-supplementary-information.html).

This document was prepared by Technical Committee *[or Project Committee]* ISO/TC *[or ISO/PC]* ###, *[name of committee]*, Subcommittee SC ##, *[name of subcommittee]*.

This second/third/… edition cancels and replaces the first/second/… edition (ISO #####:####), which has been technically revised.

The main changes compared to the previous edition are as follows:

— xxx xxxxxxx xxx xxxx

A list of all parts in the ISO ##### series can be found on the ISO website.

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](https://www.iso.org/members.html).

Introduction

Note: Detailed procedures on how to propose technologies for this Working Draft are provided in document N19192.

*Identification of patent holders, if any.*

Information technology — Coded representation of immersive media — Part 14 : Scene Description for MPEG Media

# 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 applications 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 in combination with a presentation engine. It also defines Media Access APIs for communication between the presentation engine and the Media Access Function for these extensions. While the extensions defined in this part may be applicable to other scene description formats, a specific instantiation is provided for "The GL Transmission Format (glTF) 2.0" as defined by Khronos.

# 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.

|  |  |
| --- | --- |
| [23001-15] | ISO/IEC 23001-15 Information technology — MPEG systems technologies — Part 15: Carriage of web resources in ISOBMFF |
| [glTF2.0] | glTF 2.0 Khronos Group, The GL Transmission Format (glTF) 2.0 Specification, https://github.com/KhronosGroup/glTF/tree/master/specification/2.0/ |
| [RFC6902] | IETF RFC 6902 (April 2013): JavaScript Object Notation (JSON) Patch |

# Terms, definitions, symbols and abbreviations

## Terms and definitions

For the purposes of this document, the terms and definitions given in [glTF2.0] and the following apply.

3.1.1

Presentation Engine

Engine that processes and renders the content of a scene

Note 1 to entry: Text of the note.

3.1.2

Media Access Function

Function that retrieves and prepares media for rendering on request by the Presentation Engine

## Abbreviations

3D Three-Dimensional

6DoF Six Degrees of Freedom

API Application Programming Interface

AR Augmented Reality

glTF Graphics Language Transmission Format

JSON JavaScript Object Notation

PBR Physically-Based Rendering

MAF Media Access Function

VR Virtual Reality

## Conventions

The following conventions apply in this document.

# Overview and Architecture

## Overview

Immersive media applications, for example those that aim to provide true AR and 6DoF experiences, require a scene description format that describes a rich 3D scene that enables physically-based rendering (PBR) of the audio-visual content.

Instead of specifying a new Scene Graph format for this purpose, this specification builds on the well-established glTF 2.0 format that is standardized by the Khronos Group.

The following diagram depicts the glTF 2.0 format hierarchy and shows the extensions defined in this specification:



**Figure 1 – glTF 2.0 Scene Structure**

In addition to the extensions, which provide a tight integration of MPEG media with the Scene Description, the interface between the Presentation Engine and the Media Retrieval Engine is defined. Finally, a processing model as well as conformance and validation definitions of scene descriptions according to this specification are provided.

## glTF 2.0 Extension Mechanisms

glTF 2.0 defines an extension mechanism ([glTF2.0]#specifying-extensions) that allows the base format to be extended with new capabilities. Any glTF object can have an optional extensions property that lists the extensions that are used by that object.

All extensions that are used in a glTF scene must be listed in the top-level extensionsUsed array object. Extensions that are required to correctly render the scene must also be listed in the extensionsRequired array.

As an example of a possible extension, MPEG is currently defining a 6DoF audio encoder input format to address the MPEG-I requirements on 6DoF scene audio. glTF does not provide any support for audio scenes. To address this gap, a new node type and new material extension should be defined.

Similar to Javascript for HTML documents, an active processing may be supported in order to update a glTF scene description. This allows to update the description object model in an asynchronous manner (based on events such as interactivity or server events) as well as in a synchronous manner with a media source. In the latter case, a model as defined for Web Resource Track model for which updates are timed using an ISOBMFF track format aligned with ISO/IEC 23001-15 should be defined.

glTF 2.0 can be extended beyond the core specifications by one of four means:

* Vendor extensions
* EXT Extensions
* KHR Extensions
* KHX Extensions

This document defines extensions to glTF 2.0 under the vendor-specific extension framework with an MPEG namespace.

### Normative Aspects

The extensions, their syntax and semantics, as well as their processing model are normative. Furthermore, the MAF and the Media Access API and Buffer API are also normative.

## Architecture

### General 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 MPEG 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 API, which allows the Presentation Engine to request media data required for the rendering of the scene. The Media Access Function will retrieve the requested 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 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 media data to the Presentation Engine for rendering. The media data is passed in form of buffers from the Media Access Function to the Presentation Engine. The requests for media data are passed through the Media Access API from the Presentation Engine to the Media Access Function.

The following diagram depicts the reference architecture as described previously:



**Figure 2 – Scene Description Reference Architecture**

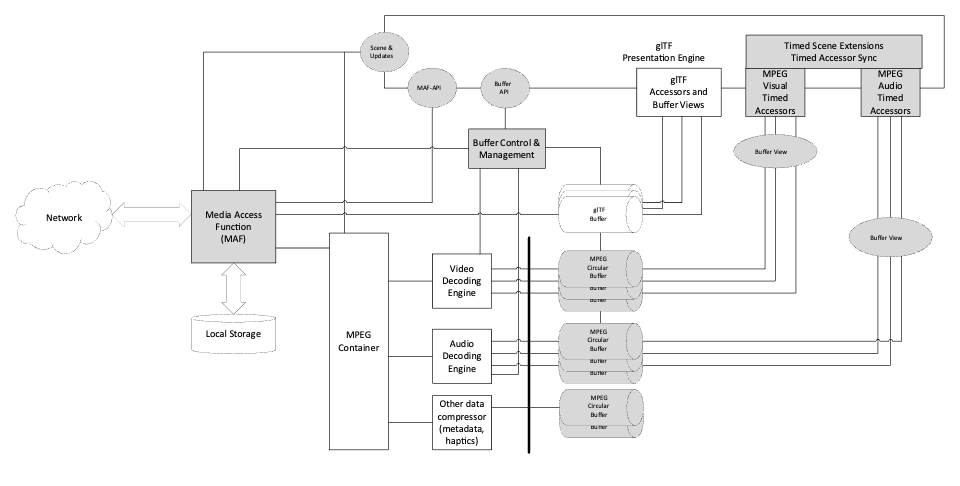
The interfaces and components in green are in scope for this document.

The following assumptions apply:

* The format of the buffers is dictated by the scene description and is passed to the MAF through the Media Access API
* Decoder might need to perform necessary transformations to match the buffer format and layout declared in the scene description for that buffer
* The fetching of scene description and scene description updates may be triggered by the MAF.

NOTE – Upon definition of the scene update mechanism, the architecture might need to be adjusted.

A more detailed description of the interactions between the Presentation Engine and the MAF is provided by **Figure 3**:



**Figure 3 – Relationship between Presentation Engine and MAF**

The diagram shows the buffers in the core of the communication between the Presentation Engine and the MAF.

The procedure can be briefly described as follows:

1. The Presentation Engine receives and parses the Scene Description and following Scene Description updates
2. It identifies external media that needs to be presented and identifies the required presentation time
3. The Presentation Engine then uses the MAF API to request the media and provides the following information:
   1. **Where**: where the MAF will find the requested media
   2. **What**: what parts of the media and at what level of detail
   3. **When**: when the requested media has to be made available
   4. **How**: in which format it wants the data and how it is passed to the Presentation Engine
4. 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

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

The basic concept is that for each buffer that is to be filled, the Media Access function has sufficient information to

1. Select the appropriate source for the media (multiple could be specified) and the MAF selects based on preferences and capabilities.
2. For each selected source,
   1. it has sufficient information to access the media by using a media access protocol.
   2. It has sufficient information to setup the pipeline to provide the information in the correct buffer format



**Figure 4 – Example Instantiation of the Media Pipelines**

The MAF may query or obtain additional information from the Presentation Engine in order to optimize the delivery, for example the required quality for each of the buffer, the exact timing information and so on.

### Pipelines

The concept of pipelines is essential for the processing model of the MPEG-I scene description. Media and Metadata are fetched and processed through pipelines that always end into a final buffer or set of buffers. The final buffer is used to exchange data with the Presentation Engine. The Media Access Function is responsible for the setup and management of the pipeline for each requested media.

A pipeline takes as input one or more media or metadata tracks and outputs one or more buffers. It performs all the necessary processing, such as streaming, demultiplexing, decoding, decryption, and format conversion to match the expected buffer format.

Figure 5 depicts different examples of pipelines. Data may come from one or more tracks, get decoded, processed and then output in one or multiple buffers.



**Figure 5 – Media Pipelines in Scene Description**

In the example, the Scene Description describes all five buffers feeding into the presentation engine. The Processing step may convert formats or may even create new static or dynamic buffers, e.g. creating index buffers based on metadata received from the file itself.

## Data and Timing Model

[TODO]

# Scene Description Extensions

## Overview

The following extensions to glTF 2.0 are defined by this document:

**Table 1 – glTF 2.0 Extensions in this document**

|  |  |  |  |
| --- | --- | --- | --- |
| **Extension Name** | **Brief Description** | **Type** | **Reference** |
| 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 |
| MPEG\_buffer\_circular | A buffer extension to support circular buffers. | Generic | 5.2.3 |
| MPEG\_scene\_dynamic | An extension to support scene updates. | Generic | 5.2.4 |
| MPEG\_texture\_video | A texture extension to support video textures. | Visual | 5.3.1 |
| 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 |

## Generic Extensions

### MPEG\_media extension

#### General

MPEG media extension, identified by MPEG\_media, provides an array of MPEG media items used in the scene.

If MPEG scene description is supported, then the MPEG\_media extension shall be supported. The MPEG media extension shall be included in the extensionsUsed and extensionsRequired of the scene description document for scene descriptions that require the use of MPEG media support.

The extension shall be declared at the top-level as follows:

{  
 "extensionsRequired": [  
 "MPEG\_media"  
 ]  
 "extensionsUsed": [  
 "MPEG\_media"  
 ]  
 }

#### Semantics

The definition of all objects within MPEG\_media extension is provided in the tables below.

**Table 2 – Definitions of top-level objects of MPEG\_media extension**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| Media | Array | N/A | An array of items that list the media referenced by timed Accessors in a scene. |

**Table 3 – Definitions of item in the media array of MPEG\_media extension**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| Name | String | N/A | Label of the media. |
| renderingRate | Number | 25.0 | The renderingRate attribute is used to indicate the frequency at which the timed media is expected to be updated as frames per second. |
| startTime | Number | 0 | The startTime gives the time at which the rendering of the timed media will be in seconds. By default, the referenced image will be rendered as a static texture until the startTime. A startTime of 0 means the presentation time of the current scene.  Either startTime or autoplay shall be present in glTF description. |
| timeOffset | Number | 0 | The timeOffset 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. |
| autoplay | Boolean | N/A | Specifies that the media will start playing as soon as it is ready.  Either startTime or autoplay shall be present in glTF description. |
| loop | Boolean | False | Specifies that the media will start over again, every time it is finished. |
| controls | Boolean | N/A | Specifies that media controls should be displayed (such as a play/pause button etc). |
| alternatives | Array | N/A | An array of items that indicate alternatives of the same media (e.g. different video code used)"  Note: the client could select items (i.e. uri and track) included in alternatives depending on the client’s capability. |

**Table 4 – Definitions of items in the alternatives array of MPEG\_media extension**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| mimeType | string | N/A | The MPEG media's MIME type.  Note : the profiles parameter, as defined in RFC6381, might be included as a part of the mimeType to specify the profile of the MPEG media container. (e.g. the profiles parameter indicates the DASH profile when the uri specifies a DASH manifest) |
| uri | string | N/A | 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 defined in section 4.5. |
| tracks | Array | N/A | An array of items that lists the referenced tracks in them MPEG media container (e.g. mp4 file or DASH manifest). |

**Table 5 – Definitions of items in the tracks array of MPEG\_media extension**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| track | string | N/A | URL fragments to access the track within MPEG media.  e.g.  DASH : Using MPD Anchors (URL fragments) as defined in Annex C of 23009-1 (Table C.1).  MP4: URL fragments as specified in Annex L of 14496-12. |
| codecs | string | N/A | The codecs parameter, as defined in RFC 6381, of the media included in the track.  Note: When the track includes different types of codecs (e.g. the AdaptationSet includes Representations with different codecs), the codecs parameter could be signaled by comma-separated list of values of the codecs. |

#### JSON Syntax/Schema

|  |
| --- |
| {  "$schema": "http://json-schema.org/draft-04/schema",  "title": "Media",  "type": "object",  "description": "MPEG media used to create a texture, audio source, or any other media type defined by MPEG.",  "allOf": [ { "$ref": "glTFChildOfRootProperty.schema.json" } ],  "properties": {  "name": { },  "renderingRate": {  "type": "number",  "minimum": 0.0,  "exclusiveMinimum": true,  "default": 25.0,  "description": "The renderingRate attribute is used to indicate the frequency at which the timed texture is expected to be updated as frames per second."  },  "startTime": {  "type": "number",  "minimum": 0.0,  "default": 0.0,  "exclusiveMinimum": false,  "description": "The startTime gives the time at which the rendering of the timed texture will be in seconds. By default, the referenced image will be rendered as a static texture until the startTime. A startTime of 0 means the presentation time of the current scene."  },  "timeOffset": {  "type": "number",  "minimum": 0.0,  "default": 0.0,  "exclusiveMinimum": false,  "description": "The timeOffset indicates the time offset into the source, starting from which the timed texture shall be generated. The value is provided in seconds, where 0 corresponds to the start of the source."  },  "autoplay": {  "type": "boolean",  "description": "Specifies that the MPEG media start playing as soon as it is ready."  },  "loop": {  "type": "boolean",  "default": false,  "description": "Specifies that the MPEG media start over again, every time it is finished."  },  "controls": {  "type": "boolean",  "description": "Specifies that MPEG media controls should exposed to end user"  },  "alternatives": {  "type": "array",  "description": "An array of alternatives of the same media (e.g. different video code used)",  "items": {  "uri": {  "type": "string",  "description": "The uri of the media.",  "format": "uriref",  "gltf\_detailedDescription": "The uri of the media. Relative paths are relative to the .gltf file.",  "gltf\_uriType": "media"  },  "mimeType": {  "anyOf": [  {  "enum": [ "video/mp4" ]  },  {  "enum": [ "application/dash+xml" ]  },  {  "type": "string"  }  ],  "description": "The MPEG media's MIME type."  },  "tracks": {  "type": "array",  "description": "List of all tracks in MPEG media container (e.g. mp4 file or DASH manifest",  "items": {  "track": {  "type": "string",  "description": "URL fragments e.g, DASH : Using MPD Anchors (URL fragments) as defined in Annex C of 23009-1 (Table C.1). MP4: URL fragments as specified in Annex L of ISOBMFF."  },  "codec": {  "type": "string",  "description": "The codecs parameter, as defined in RFC 6381, of the media included in the track."  }  }  },  "required": ["uri", "mimeType" ]  },  "minItems": 1  }  },  "oneOf": [  { "required": [ "startTime" ] },  { "required": [ "autoplay" ] }  ]  } |

#### Processing Model

Processing of the MPEG\_media extension depends on the referenced media.

#### Example

In the example below, two media items are listed by MPEG\_media object. The first media item contains only one item within alternatives, which is a DASH manifest that contains one track. Even though there are no alternatives at the MPEG media level, DASH manifest may still have different Representations within the Adaptation Set (but this is outside of the scope of the extension). The second media item contains two items within the alternatives. The first one lists an mp4 file that contains data compressed using AVC codec, while the second one lists an mp4 file that contains data compress using HEVC codec. Each item within alternatives array has to have the same amount of track items within tracks object. However, each track item may contain different information, which depends on the structure of the MP4 file.

|  |
| --- |
| {  "extensions": {  "MPEG\_media": {  media: [  {  "name": "source 0",  "renderingRate": 25.0,  "timeOffset": 0.0,  "autoplay": "true",  "loop": "true",  "controls": "false",  "alternatives": [  {  "mimeType": "application/dash+xml",  "uri": "manifest.mpd",  "tracks": [  {  "track": "#track=1"  }  ]  }  ]  },  {  "name": "source 1",  "renderingRate": 30.0,  "startTime": 9.0,  "timeOffset": 10.0,  "loop": "true",  "controls": "false",  "alternatives": [  {  "mimeType": "video/mp4;codecs=\"avc1.42E01E\"",  "uri": "video1.mp4",  "tracks": [  {  "track": "#track\_ID=1"  },  {  "track": "#track\_ID=2"  }  ]  },  {  "mimeType": "video/mp4;codecs=\"hev1.1.6.L93.B0\"",  "uri": "video2.mp4",  "tracks": [  {  "track": "#track\_ID=3"  },  {  "track": "#track\_ID=1"  }  ]  }  ]  }  ]  }  }  } |

### MPEG\_accessor\_timed extension

#### General

In order to provide access to timed media and metadata in a scene, a new glTF extension is specified to define timed accessors. An accessor in glTF 2.0 defines the types and layout of the data as stored in a buffer that is viewed through a bufferView.

When timed data is read from a buffer, the data in the buffer is expected to change dynamically with time. The buffer element is extended to add support for a circular buffer that is used with timed data.

A scene that contains timed media and/or metadata shall use the timed accessor extension to access the data. The timed accessor is an extension to regular accessors to indicate that the underlying data buffer is dynamic.

The timed accessor extension is identified by "MPEG\_accessor\_timed", which shall be included in the extensionsUsed and extensionsRequired of the scene description document, whenever timed data is used in a scene.

#### Semantics

The "MPEG\_accessor\_timed " extension shall be defined on "accessors" structures. It may contain the following properties:

**Table 6 – Definition of MPEG\_accessor\_time extension**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| immutable | boolean | False | This flag indicates if the accessor information such as the componentType, bufferView, and type are allowed to change over time. Note that count, max, min, and byteOffset are expected to change and are always included as part of the timed accessor information header. |
| bufferView | integer | N/A | This provides the reference id of a bufferView that points to the timed accessor information header. |
| updateRate | number | 25.0 | The updateRate provides the frequency at which the underlying buffer data is expected to change. The rate is provided in number of changes per second. |

The timed accessor information header contains the dynamic metadata that is needed to access the timed data.

The following table describes the syntax and semantics of the timed accessor information header:

**Table 7 – Definition of timed accessor information header fields**

|  |  |  |  |
| --- | --- | --- | --- |
| **Syntax** | **Length (bits)** | **type** | **Semantics** |
| timestamp\_delta | 32 | float | Provides a delta in seconds that is added to the timestamp of the referenced buffer to determine the timestamp of the referenced timed media. |
| if (!immutable) {  componentType  bufferView  type  normalized  reserved  } | 32  32  8  1  7 | integer  integer  integer  boolean | These fields correspond to the accessor properties as defined in [glTF2.0]. The type differs from the definition in [glTF2.0] in that it provides a 0-based index of the allowed types as defined in [glTF2.0]. For example a type of 0 indicates that the data is a "SCALAR". |
| byteOffset  count  max  min | 32  32  32  32 | integer  integer  float  float | These fields correspond to the accessor properties as defined in [glTF2.0]. |
| bufferViewByteOffset  bufferViewByteLength  bufferViewByteStride | 32  32  32 | integer  integer  integer | These fields correspond to the bufferView fields byteOffset, byteLength, and byteStride respectively. |

Note that the timed accessor information header is provided as binary data as part of the buffer data and is accessible through the bufferView of the timed accessor extension.

#### JSON Syntax/Schema

|  |
| --- |
| {  "$schema" : "http://json-schema.org/draft-04/schema",  "title" : "MPEG\_accessor\_timed extension",  "type" : "object",  "description": "glTF extension to specify timed accessor format formats",  "allOf": [ { "$ref": "glTFChildOfRootProperty.schema.json" } ],  "properties" : {  "immutable": {  "type": "boolean",  "default": false,  "description": "This flag indicates if the accessor information such as the componentType, bufferView, and type are allowed to change over time. Note that count, max, min, and byteOffset are expected to change and are always included as part of the timed accessor information header. "  },  "bufferView": {  "allOf": [ { "$ref": "glTFid.schema.json" } ],  "description": "This provides the reference id of a bufferView that points to the timed accessor information header. "  },  "updateRate": {  "type": "number",  "default": 25.0,  "description": "The updateRate provides the frequency at which the underlying buffer data is expected to change. The rate is provided in number of changes per second."  }  },  "required": [ "bufferView" ]  } |

#### Processing Model

The processing model of the MPEG\_accessor\_timed extension is to be defined.

#### Example

The following is an example showing the new extension:

|  |
| --- |
| {  "accessors": [  {  "bufferView": 0,  "componentType": 5126,  "byteOffset": 0,  "count": 12323,  "type": "VEC4",  "extensions": {  "MPEG\_timed\_accessor": {  "immutable":1,  "bufferView":1,  "updateRate":25.0  }  }  }  ],  } |

### MPEG\_buffer\_circular extension

#### General

In order to support timed data access, the buffer element is extended to provide circular buffer functionality. 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.

#### Semantics

The following properties are defined for the "MPEG\_buffer\_circular ":

**Table 8 – Definition of MPEG\_buffer\_circular extension**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| count | integer | 5 | The count field provides the number of frames that are offered by this circular buffer. Each frame 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 buffer header, which shall always be accessible at byte 0 of the frame data. |
| headerLength | integer | 12 | The headerLength provides the length of the buffer header and is the offset into the dynamic actual data. |
| updateRate | number | 25.0 | The updateRate provides the frequency at which the underlying buffer data is expected to change. The rate is provided in number of changes per second. |
| source | number |  | Index of the MPEG media entry that will be used as the source for the input data to this buffer. |

**Table 9 – Syntax and semantics of the buffer header**

|  |  |  |  |
| --- | --- | --- | --- |
| **Syntax** | **Length (bits)** | **type** | **Semantics** |
| index | 8 | integer | The index of the current buffer frame. The index is a value between 0 and count -1. |
| timestamp | 64 | integer | Provides the timestamp of the data that is contained in this buffer. The format of this field is in NTP Timestamp Format with 32 MSB for seconds and 32 LSB for fraction of seconds. Note that this timestamp field is not necessarily a wallclock time and the interpretation of this field is left to the rendering engine. |
| length | 32 | integer | The length of the data of this buffer frame, including the buffer header. |

#### JSON Syntax/Schema

|  |
| --- |
| {  "$schema" : "http://json-schema.org/schema#",  "title" : "MPEG\_buffer\_circular extension",  "type" : "object",  "description": "glTF extension to specify circular buffer",  "allOf": [ { "$ref": "glTFChildOfRootProperty.schema.json" } ],  "properties" : {  "count": {  "type": "boolean",  "default": false,  "description": "This provides the number of frames that are offered by this buffer."  },  "headerLength": {  "type": "number",  "default": 12,  "description": "This provides the length of the buffer header."  },  "updateRate": {  "type": "number",  "default": 25.0,  "description": "The updateRate provides the frequency at which the underlying buffer data is expected to change."  },  "source": {  "type": "number",  "description": "The index of the MPEG media entry that provides the source."  }  },  "required": [ "source" ]  } |

#### 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 the media decoder are inserted at the write pointer. Prior data in that frame will be overwritten and the frame buffer should be resized accordingly.

Figure 6 depicts the buffer structure:



**Figure 6 – Circular buffer operation**

The renderer shall maintain that Timestamp(write\_pointer) > Timestamp(read\_pointer). When overwriting a frame in the buffer with new timed data, the renderer shall make sure that the read\_pointer is moved to the frame with the oldest timestamp. This would result in a frame drop but will ensure that no concurrent access to the same frame in the buffer is performed.

#### Example

|  |
| --- |
| "buffers": [  {  "byteLength": 12661584,  "uri": "longdress\_vox10\_1080\_791326.bin",  "extensions": {  "MPEG\_circular\_buffer": {  "count": 5,  "headerLength": 13,  "updateRate": 25,  "source": 1  }  }  },  } |

### MPEG\_scene\_dynamic extensions

#### General

In order to expose the dynamic scene updates using the JSON patch protocol, the glTF extension MPEG\_media shall be used. Accordingly, the MPEG\_scene\_dynamic extension shall contain the URL information to access dynamic scene updates based on JSON patch documents as described in [clause 7.2](#_Scene_Updates), based on the carriage formats for JSON patch documents defined in [clause 8.2](#_JSON_Patch_Document).

#### Semantics

The extension MPEG\_scene\_dynamic links to one of the entries listed in MPEG\_media.

**Table 10 – Definition of top-level objects of MPEG\_scene\_dynamic extension**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| source | Number | N/A | Provides the index of the media listed by MPEG\_media extension that provide scene updates. |
| track | Number | N/A | Provides the index of a track of a media object, indicated by source object and listed by MPEG\_media extension, which samples contain JSON patch updates and provide timing to perform update. |

The extension would be declared at the top-level as follows:

{

"extensionsRequired": [

"MPEG\_scene\_dynamic"

]

"extensionsUsed": [

"MPEG\_scene\_dynamic"

]

}

#### JSON Syntax/Schema

|  |
| --- |
| {  "$schema" : "http://json-schema.org/draft-04/schema",  "title" : "MPEG\_scene\_dynamic extension",  "type" : "object",  "description": "glTF extension to expose dynamic scene updates using the JSON patch protocol with MPEG media",  "allOf": [ { "$ref": "glTFChildOfRootProperty.schema.json" } ],  "properties" : {  "source": {  "allOf": [ { "$ref": "glTFid.schema.json" } ],  "description": "The index of the MPEG media that provides dynamic scene update information."  },  "track": {  "allOf": [ { "$ref": "glTFid.schema.json" } ],  "description": "The index of a track of the MPEG media that provides dynamic scene update information."  }  },  "required": [ "source" ]  } |

#### Processing Model

The processing model could be as follows: When sample becomes active, the media player loads the sample data into the presentation engine and this triggers the scene update performed by the presentation engine. If the scene update contains an addition of new glTF nodes and/or potential modifications of existing glTF nodes, the presentation engine interacts with the MAF to fetch any new content associated with the scene update and presents the new content accordingly.

#### Example

In the example below, the media object includes the JSON patch document format file name and its track index. The mimeType indicates that the data is JSON patch information for dynamic scene updates.

|  |
| --- |
| "extensions": {  "MPEG\_media": {  "media": [{  "name“: “dynamic\_scene\_data1",  "alternatives": [{  "mimeType": "application/mp4;codecs=\"json-patch+json\"",  "uri": "file\_containing\_scene\_update\_track.mp4",  "tracks": [ {"track": "#track=1"} ]  }]  }]  }  "MPEG\_scene\_dynamic ": {  "source": 0,  "track": 0,  }  } |

## Visual Extensions

### MPEG\_texture\_video extensions

#### General

MPEG texture video extension, identified by MPEG\_texture\_video, provides the possibility to link a glTF 2.0 texture object to MPEG media and its respective track, listed by an MPEG\_media object. MPEG texture video extension also provides a reference to the MPEG\_accessor\_timed, using timedAccessor object, where the decoded timed texture will be made available.

If MPEG scene description is supported, then the MPEG\_video\_texture extension shall be supported. The MPEG texture video extension shall be included in the extensionsUsed and extensionsRequired of the scene description document for scene descriptions that require the use of timed textures.

When the MPEG\_texture\_video extension is not supported, a texture buffer will be filled by data described by the standard glTF 2.0 source object.

#### Semantics

**Table 11 – Definition of top-level objects of MPEG\_texture\_video extension**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| timedAccessor | number | N/A | Provides a reference to the timed accessor where the decoded timed texture will be made available. |

#### JSON Syntax/Schema

|  |
| --- |
| {  "$schema" : "http://json-schema.org/draft-04/schema",  "title" : "MPEG\_texture\_video extension",  "type" : "object",  "description": "glTF extension to specify textures using MPEG defined formats",  "allOf": [ { "$ref": "glTFChildOfRootProperty.schema.json" } ],  "properties" : {  "source": {  "allOf": [ { "$ref": "glTFid.schema.json" } ],  "description": "The index of the MPEG media used by this texture."  },  "track": {  "allOf": [ { "$ref": "glTFid.schema.json" } ],  "description": "The index of a track of the MPEG media used by this texture."  },  "timedAccessor": {  "allOf": [ { "$ref": "glTFid.schema.json" } ],  "description": "Provides a reference to the timed accessor where the decoded timed texture will be made available."  }  },  "required": [ "source" ]  } |

#### Processing Model

#### Example

In the example below, two texture items are listed. Each texture item use MPEG\_video\_texture extension. The first texture item is linked with source 1 listed by MPEG\_media and track 0, and it is expected that decoded texture will be available in buffer indicated by timed accessor 2. The second texture item is linked with the same source 1 listed by MPEG\_media but with a different track, track 0, and it will be available in buffer indicated by timed accessor 3.

|  |
| --- |
| {  "textures": [  {  "sampler": 0,  "source": 1,  "extensions": {  "MPEG\_video\_texture": {  "timedAccessor": 2,  }  }  },  {  "sampler": 1,  "source": 0,  "extensions": {  "MPEG\_video\_texture": {  "timedAccessor": 3,  }  }  }  ]  } |

## Audio Extensions

### MPEG\_audio\_spatial extensions

#### General

The MPEG audio extension adds support for spatialized audio to the MPEG scene description based on glTF 2.0. This extension is identified by MPEG\_audio\_spatial, which can be included at top level or attached to any node in the scene.

The MPEG\_audio\_spatial extension supports four different node types:

* AudioSource: an audio source that provides input audio data into the scene. Currently, only mono sources are supported.
* Type: 'Object' or 'HOA'
* HOA audio sources shall ignore the parent node's position and be rendered only in 3DoF.
* AudioReverb: A reverb effect can be attached to the output of an audio source. sceneSeveral 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
* AudioListener: An audio listener represents the output of audio in the scene. They are usually attached to camera nodes 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 7** depicts the processing chain for audio in a scene. Note that specification of any effects processing (green arrows) is out of scope for Scene Description:



**Figure 7 – An example of the processing chain for audio in a scene**

Note that the characteristics of Audio Listener depend on the actual output devices available to the audio renderer.

#### Semantics

**Table 12 – Definition of top-level objects of MPEG\_audio\_spatial extension**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
|  |  |  |  |
| Sources | Array of audio sources |  | Provides a list of AudioSource elements that are attached to this node |
| Id | Number |  | unique identifier of the audio source in the scene. |
| Type | String |  | “Object” or “HOA” |
| Pregain | Float | 0 | provides a level-adjustment in dB for the signal associated with the source. |
| playbackSpeed | Float | 1 | defines the playback speed of the audio signal. A value of 1.0 corresponds to playback at normal speed. |
| Attenuation | Enumeration | linear | provides the function used to calculate the attenuation of the audio signal based on the distance to the source. An enumeration of predefined attenuation functions is defined. |
| attenuationParameters | Array of float |  | array of parameters that are input to the attenuation function. The semantics of these parameters depend on the attenuation function itself. |
| referenceDistance | Float | 1 | provides the distance in meters for which the distance gain is implicitly included in the source signal after application of pregain.  Disregarded for HOA audio sources. |
| timedAccessor | Number |  | provides a pointer to the timed accessor that will provide the audio data for this source. |
| reverbFeed | Array of id’s |  | If present: One or more pointers to reverb units, optionally extended by a floating point scaling factor. |
| Listener | Object |  | places an audio listener node in the scene that is attached to a parent camera node. The audio listener characteristics depend on the available audio output devices. |
| Id | Number |  | unique identifier of the audio listener in the scene. |
| Reverb | Array of objects |  | list of audio reverb units that are defined at the top level of the scene. |
| Id | Number |  | unique identifier of the audio reverb unit in the scene. |
| Bypass | Boolean | true | indicates if the reverb unit can by bypassed if the audio renderer does not support it. |
| properties | Arry of object |  | contains reverb unit specific parameters |
| frequencies | float |  | Frequencies for the provided RT60 and DSR values. |
| RT60 | float |  | RT60 values(s) for the frequencies provided in the ‘frequencies’ field. |
| DSR | float |  | Diffuse-to-Source Ratio values [dB] for the frequencies provided in the ‘frequencies’ field. See explanatory text below. |
| predelay | Number |  | Delay from onset of source to onset of late reverberation for which DSR is provided. |

#### JSON Syntax/Schema

|  |
| --- |
| {  "$schema" : "http://json-schema.org/draft-04/schema",  "title" : "MPEG\_audio\_spatial extension",  "type" : "object",  "description": "glTF extension to specify spatial audio support",  "allOf": [ { "$ref": "glTFChildOfRootProperty.schema.json" } ],  "definitions": {  "audiosource": {  "type": "object",  "properties": {  "id": {"type": "number"},  "type": {"type": "string"},  "pregain": {"type": "number", "default": 0},  "playbackSpeed": {"type": "number", "default": 1},  "attenuation": {"type": "string", "enum": {"linear"}, "default": "linear"},  "attenuationParameters": {"type": "array", "items": {"type": "number"}},  "referenceDistance": {"type": "number", "default": 1},  "timedAccessor": {"type": "number"},  "reverbFeed": {"type": "array", "items": {"type": "number"}},  }  },  "audiolistener": {  "type": "object",  "properties": {  "id": {"type": "number"}  }  },  "reverbproperty": {  "type": "object",  "properties": {  "frequencies": {"type": "number"},  "RT60": {"type": "number"},  "DSR": {"type": "number"}  }  }  "reverb": {  "type": "object",  "properties": {  "id": {"id": "number"},  "bypass": {"type": "boolean"},  "properties": {"type": "array", "items": {"type": "#/definitions/reverproperty"}},  "predelay": {"type": "number"}  }  }  },  "properties" : {  "sources": {  "type": "array",  "items": {  "type": "#/definitions/audiosource"  },  "description": "The index of the MPEG media used by this texture."  },  "listener": {  "type": "#/definitions/audiolistener",  "description": "The index of a track of the MPEG media used by this texture."  },  "reverbs": {  "type": "#/definitions/reverb",  "description": "Provides a reference to the timed accessor where the decoded timed texture will be made available."  }  },  "required": [ "sources", "listener" ]  } |

#### 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 as follows:

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.

#### Example

|  |
| --- |
| {  "asset": {  "generator": "MPEG",  "version": "2.0"  },  "scene": 0,  "scenes": [  {  "nodes": [  0, 1  ]  }  ],  "nodes": [  {  "mesh": 0,  "children": [2]  },  {  "camera": 0,  "children": [3]  },  {  "extensions": {  "MPEG\_audio\_spatial ": {  “source”: {  "id": 0,  "volume": 110,  "distance": 30,  "attenuation": "linear",  "attenuationParameters": [-5.0],  "timedAccessors": 0  }  }  }  },  {  "extensions": {  "MPEG\_spatial\_audio": {  "listener": {  "id": 0  }  }  }  }  ],  "MPEG\_media": {  "media": [  {  "name": "audio\_source\_1",  "loop": true,  "alternatives": [  {  "mimeType": "audio/aac",  "uri": "https://example.com/audio\_source\_0.aac"  }  ]  }  ]  }  }  } |

## Metadata Extensions

### MPEG\_viewport\_recommended extensions

#### General

MPEG viewport recommended extension, identified by MPEG\_viewport\_recommended, provides the link from a glTF2.0 camera object to recommended viewport timed metadata by referencing to the MPEG accessor timed, where the sample of recommended viewport timed metadata will be made available. The MPEG\_viewport\_recommended extension shall be included in the extensionsUsed and extensionsRequired of the scene description document, whenever MPEG\_viewport\_recommended extension is used in a scene.

The recommended viewport timed metadata provides dynamically changed information which includes translation and rotation of the node which includes the camera object, as well as the intrinsic camera parameter of the camera object. The client renders viewport according to the dynamically changed information.

#### Semantics

**Table 13 – Definition of MPEG\_viewport\_recommended extension**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| name | string | N/A | Label of the recommended viewport |
| translation | number | N/A | Provides a reference to timed accessor where the timed data for the translation of camera object will be made available. The type of the referenced accessor is FLOAT\_VEC3 value, (x, y, z). |
| rotation | number | N/A | Provides a reference to timed accessor where the timed data for the rotation of camera object will be made available. The type of the referenced accessor is FLOAT\_VEC4 unit quaternion value, (x, y, z, w). |
| perspective | number | N/A | Provides a reference to timed accessor where the timed data for the perspective of the camera object will be made available. The type of the referenced accessor is FLOAT\_VEC4 value, (aspectRatio, yfov, zfar, znear). |

#### JSON Syntax/Schema

|  |
| --- |
| {  "$schema": "http://json-schema.org/draft-04/schema",  "title": "MPEG\_viewport\_recommended extension",  "type": "object",  "description": "glTF extension for specifying recommended viewport.",  "allOf": [  {  "$ref": "glTFProperty.schema.json"  }  ],  "properties": {  "name": {  "type": "string",  "description": "The name of the recommended viewport."  },  "translation": {  "allOf": [  {  "$ref": "glTFid.schema.json"  }  ],  "description": "The index of the accessor that contains the translation value of camera object."  },  "rootation": {  "allOf": [  {  "$ref": "glTFid.schema.json"  }  ],  "description": "The index of the accessor that contains the rotation value of camera object."  },  "perspective": {  "allOf": [  {  "$ref": "glTFid.schema.json"  }  ],  "description": "The index of the accessor that contains perspective parameters of the camera object."  }  }  } |

#### Processing Model

The processing model could be as follows: According to the hypothetical model defined in this document, when sample of recommended viewport timed metadata track becomes active, the media player loads the sample data into the presentation engine and this triggers the change of the setting of glTF2.0 camera object performed by the presentation engine.

#### Example

In the example below, one camera object is listed. This camera object uses MPEG\_viewport\_recommended extension. it is expected that recommended viewport information will be available in buffer indicated by timed accessor 0, 1, 2.

|  |
| --- |
| "cameras":[{  "name":"Finite perspective camera",  "type": "perspective",  "perspective": {  "aspectRatio": 1.5,  "yfov": 0.660593,  "zfar": 100,  "znear": 0.01  },  "extensions": {  "MPEG\_viewport\_recommended":[{  "name": "Recommended view 1",  "translation": 0,  "rotation": 1,  "perspective": 2  }]  }  }] |

In the example below, one media object is listed. This media object includes the recommended viewport file name and its track index. The mimeType indicates that the data is recommended viewport information.

|  |
| --- |
| "extensions" : {  "MPEG\_media" : {  "media" : [{  "name": "recommended\_viewport\_data1",  "alternatives": [{  "mimeType": "application/mp4;codecs=\"recv\"",  "uri": "recommnededviewport1.mp4",  "tracks": [ {"track": "#track=1"} ]  }]  }]  }  } |

### MPEG\_animation\_timing extensions

#### General

In order to link an animation glTF 2.0 to timed metadata and its respective track listed by MPEG\_media object an MPEG animation extension is defined. The MPEG animation timing extension is identified by MPEG\_animation\_timing, which shall be included in the extensionsUsed and extensionsRequired of the scene description document, whenever animation timing is used in a scene.

Alignment between MPEG media timelines and glTF 2.0 animation timeline enables creation of narrated stories. The animation timing metadata could allow simultaneous pausing and other manipulation of glTF 2.0 animation and MPEG media. By manipulating the global timeline for narrated content, the glTF 2.0 animation and MPEG media would be manipulated as well.

#### Semantics

**Table 14 – Semantics of MPEG animation timing extension**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| source | number | N/A | Provides the index of the media listed by MPEG\_media extension that provide animation timing information. |
| track | number | N/A | Provides the index of a track of a media object, indicated by source object and listed by MPEG\_media extension, which samples contain animation timing information. |

#### JSON Syntax/Schema

|  |
| --- |
| {  "$schema" : "http://json-schema.org/draft-04/schema",  "title" : "MPEG\_animation\_timing extension",  "type" : "object",  "description": "glTF extension to specify timing information that allow to synchronized animation with MPEG media",  "allOf": [ { "$ref": "glTFChildOfRootProperty.schema.json" } ],  "properties" : {  "source": {  "allOf": [ { "$ref": "glTFid.schema.json" } ],  "description": "The index of the MPEG media that provides timing information."  },  "track": {  "allOf": [ { "$ref": "glTFid.schema.json" } ],  "description": "The index of a track of the MPEG media that provides timing information."  }  },  "required": [ "source" ]  } |

#### Processing Model

The processing model could be as follows: when a sample of animation timing track becomes active, the media player loads the sample data into the presentation engine and this triggers the change of the state of glTF 2.0 animation performed by the presentation engine.

#### Example

|  |
| --- |
| {  "extensions": {  "MPEG\_media": {  media: [  {  "name": "source 0",  "alternatives": [  {  "mimeType": "application/mp4",  "uri": "file.mp4",  "tracks": [  {  "track": "#track\_ID=1"  }  ]  }  ]  }  ]  }  },  "animations": [  {  "name": "Animate all properties of one node with different samplers",  "channels": [  {  "sampler": 0,  "target": {  "node": 1,  "path": "rotation"  }  },  {  "sampler": 1,  "target": {  "node": 1,  "path": "scale"  }  },  ],  "samplers": [  {  "input": 4,  "interpolation": "LINEAR",  "output": 5  },  {  "input": 4,  "interpolation": "LINEAR",  "output": 6  }  ]  }  "extensions": {  "MPEG\_animation\_timing": {  "source": 0,  "track": 0  }  }  } |

# Media Access and Buffer API

## Media Access API

The Media Access API is a standardized API that is offered by any compliant Media Access Function to the Presentation Engine.

The following methods are offered through this API:

**Table 15 – Description of Media Access API**

| **Method** | **State after Success** | **Brief Description** |
| --- | --- | --- |
| initialize( ) | READY | The Presentation Engine initializes a new media access pipeline. It provides information related to the requested media or metadata. The MAF will setup the 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 Pipeline to start fetching the requested data. |
| stopFetching( ) | READY | The Presentation Engine may request to stop data fetching through this pipeline. |
| destroy( ) | IDLE | Finally, the Presentation Engine may request to destroy this pipeline and free any associated resources. |

The IDL description of this interface is provided in the following table:

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

The defined data types are provided in the following table:

|  |
| --- |
| 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 BufferHandler handler;  attribute ComponentType componentType;  attribute SampleType sampleType;  attribute integer offset;  attribute integer stride;  attribute AttributeType attributeType;  };  interface ViewInfo {  attribute Pose pose;  attribute Transform objectPosition;  };  Enum AttributeType {"ATTRIB\_NORMAL”,”ATTRIB\_POSITION”,”ATTRIB\_COLOR”, ”ATTRIB\_TEXCOORD”,”ATTRIB\_INDEX”,”ATTRIB\_TANGENT”, ”ATTRIB\_WEIGHTS”} |

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, …) 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 ore 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 pipeline is established through the bufferId attribute.

## 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 following methods:

**Table 16 – Description of Buffer API**

| Method | Brief Description |
| --- | --- |
| allocate() | Allocates a buffer for the data exchange between the MAF and the Presentation Engine. |
| writeFrame() | writes a frame to the buffer. |
| readFrame() | reads a frame from the buffer. |
| 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 IDL description of the Buffer API interface is provided in the following table:

|  |
| --- |
| interface CircularBuffer {  readonly attribute Frame frames[];  readonly attribute count;  readonly attribute integer read\_ptr;  readonly attributre integer write\_ptr;  readonly attribute decimal updateRate;  attribute integer headerLength;  attribute EventHandler onframewrite;  attribute EventHandler onframeread;  void allocate(int count);  void writeFrame(Frame frame);  Frame readFrame();  Void free();  };  interface Frame {  attribute integer index;  attribute long timestamp;  attribute integer length;  }; |

# Scene Processing Model

## General

This clause describes the scene processing model.

## Scene Updates

Scene Updates shall be expressed using the JSON Patch protocol as defined in IETF RFC 6902 [RFC6902]. ISOBMFF-based carriage format for JSON patch documents is specified in subclause 8.2. The glTF extensions MPEG\_media and MPEG\_scene\_dynamic shall be used in order to expose the dynamic scene updates using the JSON patch protocol, as described in subclause 5.2.4.

Each update operation shall consist of a JSON Patch document, where all update operations are considered as a single timed transaction. When no node matches the node selection, the update command of this node shall be discarded. When all update commands have been processed or discarded, the scene graph update shall be considered completed.

After successfully performing an update operation, the resulting scene graph shall be consistent, valid, and all references shall be correct. Since glTF 2.0 uses the order of elements for referencing, 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 update operation.

When a JSON Patch document contains a target scene description version, the client shall not perform any updates from the scene description that it is using if the target scene description version does not match the current scene description version.

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

The Presentation Engine parses the scene graph and maintains a representation of the graph in memory. The Presentation Engine needs to know when a certain scene update has to be applied. The update synchronization information can be obtained from the container format for the scene update samples as described in [clause 8.](#_JSON_Patch_Document)2. The scene updates themselves will modify the graph representation in memory and may add new media to the scene. The timing for the newly added or updated media is determined by the metadata in the scene description.

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.

# Carriage Formats

## General

This clause describes the carriage formats related to scene description for MPEG media.

## JSON Patch Document Carriage Format for Scene Updates

The ISOBMFF format based on ISO/IEC 23001-15 [23001-15] shall be used toward carrying the JSON patch documents for scene updates. The glTF extensions MPEG\_media and MPEG\_scene\_dynamic shall be used in order to expose the dynamic scene updates using the JSON patch protocol, as described in subclauses 5.2.4 and 7.2.

The brand 'scen' shall be used to signal the presence of tracks with the following constraints:

* The track handler type shall be 'meta'.
* The sample entry format shall be 'stxt' and:
  + its mime\_format field shall be set to application/json-patch+json,
  + its content\_encoding field shall contain either an empty string or a value allowed in HTTP's Content-Encoding header.
* The content of each sample shall be compliant to JSON patch format as defined in IETF RFC 6902 [RFC6902] based on the MIME type application/json-patch+json. Each sample shall be marked as a sync sample. Samples may have the sample\_has\_redundancy flag set to 1, in which case ISO/IEC 14496-12 processing is applied as discussed in clause 4.4 of ISO/IEC 23001-15 [23001-15].

Apart from the scene update operations described in IETF RFC 6902, track samples may additionally contain:

* The presentation timestamp identifying the execution time of the scene update transaction, which may correspond to wall clock time for live presentations based on absolute timestamps in UTC or TAI formats.
* An identifier that corresponds to the version of the scene update on which the update transactions shall apply.
* A triggering event identifier (user interaction, event happening in the scene, ...).

These relevant parameters for dynamic scene updates are defined in **Table 17**.

**Table 17 – Definitions of relevant attributes for dynamic scene updates**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| version\_id | String | n/a | Identifier for the version of the dynamic scene update |
| event\_id | String | n/a | Identifier for the event triggering the dynamic scene update |
| absolute\_time\_UTC | DateTime | n/a | Wall clock time identifying the execution time of the scene update transaction on the glTF object. The value is denoted in UTC. |
| absolute\_time\_TAI | DateTime | n/a | Wall clock time identifying the execution time of the scene update transaction on the glTF object. The value is denoted in TAI. |

The relevant parameters in **Table 17** may be signalled as part of the samples based on suitable extensions of ISO/IEC 23001-15.

As per the above syntax, the signalling of the absolute timestamp in both UTC and TAI formats is permitted. While MPEG media formats (e.g., DASH) support UTC-based signalling for live media, UTC format suffers from the issue of leap seconds caused by irregularities in earth’s rotation. In particular, the unpredictability of leap seconds may be problematic for dynamic scenes of time-critical nature that require precise timestamping. The option to signal the absolute timestamp information in TAI format is provided to avoid the leap second problem, since TAI is precisely defined in a manner that is independent of earth’s rotation. It is required that the scene description contains absolute\_time\_UTC, while signalling of absolute\_time\_TAI is optional.

A presentation engine may use several methods in order to obtain the wall clock times as used by the JSON patch-based Scene Description update and may synchronize its clock to the one used to generate the Scene Description update. Such methods may include the NTP protocol as defined in IETF RFC 5905, SNTP protocol as defined in IETF RFC 5905 as well as others.

1. (normative)

Bibliography

[1] IETF RFC 5905 (in force) Network Time Protocol Version 4: Protocol and Algorithms Specification

[2] ISO #####‑##:20##, *General title — Part ##: Title of part*