ISO/IEC JTC 1/SC 29/WG 03 N00481

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

**Document type:** Output Document

**Title:** WD of ISO/IEC 23090-7 AMD 1 Common Metadata for Immersive Media

**Status:** Approved

**Date of document:** 2022-01-26

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

**No. of pages:** 11 (with cover page)

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

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

**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

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

**ISO/IEC JTC 1/SC 29/WG 03 N00481**

**January 2022, Virtual**

|  |  |
| --- | --- |
| **Title** | **WD of ISO/IEC 23090-7 AMD 1 Common Metadata for Immersive Media** |
| **Source** | **WG 03, MPEG Systems** |
| **Status** | **Approved** |
| **Serial Number** | **21197** |

[1 Introduction 3](#_Toc94083873)

[2 Scope 3](#_Toc94083874)

[3 Guiding principles 3](#_Toc94083875)

[4 Usage of Metadata in ISOBMFF 3](#_Toc94083876)

[5 Metadata of individual features 3](#_Toc94083877)

[5.1 Basic 3](#_Toc94083878)

[5.1.1 Coordinate Systems 3](#_Toc94083879)

[5.1.2 Vector3 4](#_Toc94083880)

[5.1.3 Orientation in 3D space 4](#_Toc94083881)

[5.1.4 Rotations in the Unit Sphere Coordinate System 5](#_Toc94083882)

[5.1.5 Scaling 5](#_Toc94083883)

[5.2 Viewing Spaces 6](#_Toc94083886)

[5.2.1 Cuboid Viewing Spaces 6](#_Toc94083887)

[5.3 Regions 6](#_Toc94083892)

[5.3.1 Cuboid Regions 6](#_Toc94083893)

[5.4 Viewpoint 7](#_Toc94083894)

[5.4.1 Syntax 7](#_Toc94083899)

[5.4.2 Semantics 7](#_Toc94083900)

[5.5 Extrinsics 7](#_Toc94083903)

[5.5.1 Syntax 8](#_Toc94083904)

[5.5.2 Semantics: 8](#_Toc94083905)

[5.6 Intrinsics 9](#_Toc94083906)

[6 Metadata of spatially related features 10](#_Toc94083907)

[6.1 Localized coordinate systems (e.g., one is in another “global” coordinate system) 10](#_Toc94083908)

[6.2 Localized viewing spaces (e.g., one is in another “larger” viewing space) 10](#_Toc94083909)

[6.3 Sub-regions (e.g., one or more are parts of another “source” region) 10](#_Toc94083910)

[6.4 Objects in Regions (e.g., one or more objects are in a “containing” region) 10](#_Toc94083911)

[6.5 Overlay of Regions/Objects (e.g., one is in front a “background” region) 10](#_Toc94083912)

[7 Dynamic metadata tracks 10](#_Toc94083913)

[8 DASH descriptors 10](#_Toc94083914)

# Introduction

Many parts of ISO/IEC 23090 have developed metadata definitions. Some of them are very similar to each other. In order to harmonize between the parts and to reuse already defined metadata for future development, the purpose of this amendment is to collect the metadata defined by the parts of ISO/IEC 23090, e.g. 23090-10, 23090-18 etc., and to explore harmonization between them. The scope of the new amendment does not expand the scope of the original project.

# Scope

* metadata collected from and applicable to all MPEG-I parts
* 23090: 5, 7, 9, 10, 12, 14, 18
* 23001: 10

# Guiding principles

TBD

# Usage of Metadata in ISOBMFF

* Static: extension of containing boxes
* Dynamic: timed metadata tracks

# Metadata of individual features

## Basic

### Coordinate Systems

The following MPEG-I reference coordinate system is a right-handed 3D Cartesian coordinate system with 6 degrees of freedoms (DoFs): 3 translations along the 3 x-y-z dimensions, and 3 rotations about the 3 x-y-z dimensions with the right-hand:



With this coordinate system, the following variations can be derived:

* Coordinate system – Cartesian coordinate system: the reference coordinate system with the 3 translations but without the 3 rotations.
* Unit sphere coordinate system (OMAF specific): the reference coordinate system on the unite sphere with only the 3 rotations.
* Object coordinate space – referring to object space, where manipulation is done relative to object origin: the reference coordinate system with the origin at the object origin and with the 3 translations and 3 rotations limited to the object space (or object viewing space).
* World coordinate space – referring to scene space, where manipulation is done relative to scene origin: the reference coordinate system with the origin at the scene origin and with the 3 translations and 3 rotations limited to the scene space (or scene viewing space).
* Provide example of how to move between different spaces; (TBD)

### Vector3

Dimensions, positions, sizes can be defined using the following 3D vector data structure.

#### Syntax

aligned(8) class Vector3(unsigned int precision\_bytes\_minus1) {  
 signed int((precision\_bytes\_minus1+1)\*8) x;  
 signed int((precision\_bytes\_minus1+1)\*8) y;  
 signed int((precision\_bytes\_minus1+1)\*8) z;  
}

#### Semantics

precision\_bytes\_minus1: Plus 1, specifies the precision of Vector3 components in bytes.

x, y and z: specify the x, y, and z coordinate values, respectively, of a 3D point in the Cartesian coordinate system

* Define syntax structures Vector3Uint, Vector3Int, Vector3Float
* Define translation processes whenever required
* Use them consistently when 3d positions, offsets, dimensions, translations or scaling is handled

### Orientation in 3D space

Rotations and orientations using the following quaternion representation are defined as follows:

#### Syntax

aligned(8) class QuaternionRotation () {  
 signed int(32) quat\_x;  
 signed int(32) quat\_y;  
 signed int(32) quat\_z;  
}

#### Semantics

quat\_x, quat\_y, and quat\_z, indicate the x, y, and z components, respectively, of the rotation using the quaternion representation. The values shall be in the range of – 230 to 230, inclusive. When the component of rotation is not present, its value shall be inferred to be equal to 0. The value of rotation components may be calculated as follows:

qX = quat\_x ÷ 230, qY =  quat\_y ÷ 230, qZ = quat\_z ÷ 230

The fourth component, qW, for the rotation using the quaternion representation is calculated as follows:

qW = Sqrt( 1 – ( qX2 + qY2 + qZ2 ) )

The point (w, x, y, z) represents a rotation around the axis directed by the vector (x, y, z) by an angle 2\*cos ^{-1}(w)=2\*sin ^{-1}(sqrt(x^{2}+y^{2}+z^{2})).

NOTE – As aligned ISO/IEC FDIS 23090-5, qW is always positive. If a negative qW is desired, one can signal all three syntax elements, quat\_x, quat\_y, and quat\_z with an opposite sign, which is equivalent.

### Rotations in the Unit Sphere Coordinate System

Rotations in the unit sphere coordinate system are defined as follows:

#### Syntax

aligned(8) class UnitRotation () {

signed int(32) y;

signed int(32) p;

signed int(32) r;

}

#### Semantics

y, p, and r indicate the yaw, pitch and roll specify the yaw, pitch, and roll angles, respectively, of the rotation that is applied to the unit sphere, in units of 2−16 degrees, relative to underlying coordinate axes. y shall be in the range of −180 \* 216 to 180 \*216 − 1, inclusive. p shall be in the range of −90 \* 216 to 90 \* 216, inclusive. r shall be in the range of −180 \* 216 to 180 \* 216 − 1, inclusive.

* Define syntax structures and processes for rotation and orientation
* Make sure that other syntax structures use orientation and rotation correctly and efficiently

### Scaling

Scaling in 3-dimension is defined using the following data structure:

#### Syntax

aligned(8) class 3DScaling (unsigned int(8) scale\_precision) {

Vector3 scale(scale\_precision);

}

#### Semantics

scale\_precision indicates the precision of scale in number of bits.

scale.x, scale.y, and scale.z indicate the scaling extension in the Cartesian coordinates along the x, y, and z axes, respectively, relative to the origin (0,0,0).

* TBD: How does differ from scaling defined in 4.1.2

## Viewing Spaces

### Cuboid Viewing Spaces

A cuboid viewing space is defined as follows:

#### Syntax

aligned(8) class ViewingSpace(unsigned int(8) precision) {

Vector3 anchor(precision);

Vector3 dimensions(precision);

}

#### Semantics

anchor.x, anchor.y, and anchor.z indicate the x, y, z position values of the anchor point of the viewing space, respectively, relative to the origin (0,0,0).

dimensions.x, dimensions.y, and dimensions.z indicate the dimensions (or ranges) in the Cartesian coordinates along the x, y, and z axes, respectively, from to the anchor (anchor.x, anchor.y, anchor.z).

* Dimensions: 3D and 2D
  + TBD: What is a 2D viewing space?
* Shapes: Cuboid (Bounding Box), Sphere, Cylinder, Ellipsoid, etc.

## Regions

### Cuboid Regions

aligned(8) class CubiodRegion (  
 unsigned int(1) anchor\_included,  
 unsigned int(1) scale\_included,  
 unsigned int(8) precision)  
{  
 unsigned int(16) id;  
 unsigned int(32) size;  
 if (anchor\_included) { // anchor is not 0,0,0  
 Vector3 anchor(precision);  
 }  
 if (scale\_included) { // scale is not (1,1,1)  
 Vector3 scale(precision);  
 }  
 Vector3 dimension(precision);  
 }

#### Semantics

* Dimensions: 2D and 3D
* Shapes
* Sizes
  + TBD: Depending on the definition isn’t the size of the region or shape included in the definition itself?
* IDs
  + TBD: Not sure if ID can be considered generic enough. People can create other standard specific structures that use the common metadata here.

## Viewpoint

A viewpoint (an anchor and a rotation) is defined as follows:

### Syntax

aligned(8) class Viewpoint (unsigned int anchor\_included, unsigned int precision)  
{  
 if (anchor\_included) { // anchor is not 0,0,0  
 Vector3 anchor(precision);  
 }  
 UnitRotation rotation(precision);  
}

### Semantics

## Extrinsics

* + Orientations
  + Positions
  + IDs

Camera extrinsic parameters are often represented by a static metadata, however it may also be desirable to be able to store camera extrinsics as timed metadata tracks. If so, it would be useful if the same structure could be used for both the item property and timed metadata.

To efficiently do that, the following signaling is needed:

1. A variable bit-precision is needed so that the object size can be kept down.
2. The unit positioning needs to be variable to address multiple ranges.
3. The denominator for orientation signaling needs to be variable.
4. It needs to be possible to store relative and absolute values.

### Syntax

class CameraExtrinsics(unsigned char **abs\_flag**, unsigned char **mode**, unsigned char **pos\_bytes\_minus1**, unsigned char **pos\_unit**, unsigned char **quat\_bytes\_minus1**, unsigned char **quat\_den\_bits\_minus1**) {  
 if(mode & 0x1 || mode & 0x2) {  
 signed int((pos\_bytes\_minus1+1)\*8) pos\_x;  
 }  
 if(mode & 0x2) {  
 signed int((pos\_bytes\_minus1+1)\*8) pos\_y;  
 signed int((pos\_bytes\_minus1+1)\*8) pos\_z;  
 }  
 if(mode & 0x4) {  
 signed int((quat\_bytes\_minus1+1)\*8) quat\_x;  
 signed int((quat\_bytes\_minus1+1)\*8) quat\_y;  
 signed int((quat\_bytes\_minus1+1)\*8) quat\_z;  
 }  
};

### Semantics:

**abs\_flag:** If 1, absolute position and orientation is specified. If 0, the specified values are added relative to the previously coded position and orientation.

**mode**: Signalling mode; Valid values are:

1: Only the position is signalled along the x-axis.

2: Only the position is signalled along multiple axes.

4: Only orientation is signalled.

6: Orientation and position is signalled along multiple axes.

**pos\_bytes\_minus1**: Plus 1 indicates the number of bytes to be read for pos\_x, pos\_y and pos\_z**.** Valid values are in the range from [0, 3].

**pos\_unit:** Unit of pos\_x, pos\_y and pos\_z. Valid values are in the range from [0, 2], where

0: µm

1: mm

2: m

**quat\_bytes\_minus1**: Plus 1 indicates the number of bytes to be read for quat\_x, quat\_y, quat\_z**.** Valid values are in the range from [0, 1].

**quat\_den\_bits\_minus1:** Specifies the denominator of quat\_x, quat\_y and quat\_z. Valid values for quat\_den\_bits\_minus1 are in the range from [0, 13]. The denominator is computed as follows:

denominator = 2**quat\_den\_bits\_minus1** + 1

pos\_x: Specifies the x-coordinate of the location of the camera in units specified by pos\_unit. When not present, its value shall be inferred to be 0 if abs\_flag is 1.

pos\_y: Specifies the y-coordinate of the location of the camera in units specified by pos\_unit. When not present, its value shall be inferred to be 0 if abs\_flag is 1.

pos\_z: Specifies the z-coordinate of the location of the camera in units specified by pos\_unit. When not present, its value shall be inferred to be 0 if abs\_flag is 1.

quat\_x: Specifies the x component, qX, for the rotation of the camera using the quaternion representation. The range of quat\_x shall be in the range of -2quat\_den\_bits\_minus1+1 to 2quat\_den\_bits\_minus1+1, inclusive. When not present, its value shall be inferred to be 0 if abs\_flag is set to 1.

quat\_y: Specifies the y component, qY, for the rotation of the camera using the quaternion representation. The range of quat\_y shall be in the range of -2quat\_den\_bits\_minus1+1 to 2quat\_den\_bits\_minus1+1, inclusive. When not present, its value shall be inferred to be 0 if abs\_flag is set to 1.

quat\_z: Specifies the z component, qZ, for the rotation of the camera using the quaternion representation. The range of quat\_z shall be in the range of -2quat\_den\_bits\_minus1+1 to 2quat\_den\_bits\_minus1+1, inclusive. When not present, its value shall be inferred to be 0 if abs\_flag is set to 1.

The values of the quaternion representation are computed as follows:

qX = **quat\_x** / denominator  
qY = **quat\_y** / denominator  
qZ = **quat\_z** / denominator

It is a requirement of bitstream conformance that:

qX2 + qY2 +qZ2 <= 1

The fourth component of the quaternion representation, qW, is computed as follows:

qW = Sqrt( 1 – ( qX2 + qY2 + qZ2 ) )

## Intrinsics

TBD

* Intrinsics
  + Type
  + Type dependent parameters
* Objects/Components
  + TBD: we need to discuss, if this is generally used by other specifications.
  + Shape
  + Sizes
  + Attributes
  + IDs

# Metadata of spatially related features

## Localized coordinate systems (e.g., one is in another “global” coordinate system)

* References to global coordinate systems
* Positions
* Rotations
* Scaling

## Localized viewing spaces (e.g., one is in another “larger” viewing space)

* References to global viewing spaces
* Positions
* Rotations
* Scaling

## Sub-regions (e.g., one or more are parts of another “source” region)

* Positions
* Rotations
* Scaling

## Objects in Regions (e.g., one or more objects are in a “containing” region)

* Positions
* Rotations
* Scaling

## Overlay of Regions/Objects (e.g., one is in front a “background” region)

* Positions
* Rotations
* Scaling
* Alpha blending

# Dynamic metadata tracks

* TBD: needs to discuss, if we need to define sample and track design for some common dynamic metadata scenarios like viewport tracks. One possibility is to define them in MPEG-B part 10 (Carriage of Timed Metadata in ISOBMFF)

# DASH descriptors

* TBD