 ISO/IEC JTC 1/SC 29/WG 6 N0169

**ISO/IEC JTC 1/SC 29/WG 6**

**MPEG Audio Coding Convenorship: US**

**Document type:** General

**Title:** MPEG-I Immersive Audio Encoder Input Format, Version 3

**Status:** Approved

**Date of document:** 2022-10-28

**Source:** ISO/IEC JTC 1/SC 29/WG 6

**Expected action:** None

**Action due date:** None

**No. of pages:** 37 (including cover page)

**Email of Convenor:** srq @ audioresearchlabs . com

**Committee URL:** https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg6

**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC1/SC29/WG6, MPEG AUDIO CODING**

**ISO/IEC JTC1/SC29/WG6 N0169**

**October 2022, Hybrid Mainz &Virtual**

|  |  |
| --- | --- |
| **Title** | **MPEG-I Immersive Audio Encoder Input Format, Version 3** |
| **Source** | **WG 6, MPEG Audio Coding** |
| **Status** | **Approved** |
| **Serial Number** | **22054** |

# Introduction

This document specifies the MPEG-I 6DoF audio encoder input format (EIF) for the   
MPEG-I 6DoF Audio encoder. The architecture of the evaluation platform can be found in [1].

The encoder input data contains information describing an MPEG-I 6DoF Audio scene. This covers all contents of the virtual auditory scene, i.e. all of its sound sources, and resource data, such as audio waveforms, source radiation patterns, information on the acoustic environment, etc. The input data also allows to describe changes in the scene. These changes, referred to as updates, can either happen at distinct times, allowing scenes to be animated (e.g. moving objects). Alternatively, they can be triggered manually or by a condition (e.g. listener enters proximity) or be dynamically updated from an external entity.

The document is organized as follows: First, the meta model of the MPEG-I 6DoF audio encoder input format is defined. Then several specifications of important entities according to the audio requirements in [2] are presented. Annex A defines conventions (e.g. for radiation patterns, geometry, etc.). The specifications in this document are used to describe the input to 6DoF metadata encoders.

A clarifications and information document also exists to provide further descriptive guidance on the intent of the scene content providers [4] and may be further updated.

## Files and directories

The audio encoder input data for a single scene consists of a package of files. These files are to be stored in a single directory that has the name of the scene (e.g. *FountainMusicVR*). A scene’s input data consists of the following files:

* the scene description XML file, and additional files for
* WAV files containing original (un-coded) audio signals,
* SOFA files describing source directivities

In a different directory the WAV files containing MPEG-H LC encoded and decoded audio signals are provided. More details of the folder structure can be found in [1]. MPEG-I encoders can access the whole scene described above.

The EIF file declares the scene’s entities and contains references to data that is in the other input files. These input files (actually file pathnames) are referenced in the declaration syntax of the entities. All pathnames are specified relative to the directory containing the EIF file. The EIF file also describes dynamic behavior over time (e.g. animations), if such exists.

# Scene meta model

The general structure of a MPEG-I 6DoF audio scene description is defined by a meta model (see Fig. 1). Note, that this structure is used for the evaluation only and does not represent a possible MPEG-I bitstream. It describes the general anatomy of a scene, without focus on specific entities:

1. An audio scene consists of entities (e.g. sources, geometry, resources, etc.).
2. Entities are identified by an ID (string).  
   The format of this ID is arbitrary, but IDs must be unique.
3. Entities exist over the entire time of the scene.  
   They are not created/deleted dynamically, i.e. during runtime.
4. Entities can be associated with each other using their IDs  
   (e.g. a source with a directivity).
5. Entities have properties (e.g. position, volume, material, etc.).
6. Entities have a specified entity type, determined by their declaration.
7. The type of an entity determines its properties and possible modifications  
   of these (similar to a datatype or class in programming).

When the scene is written down in XML, entities are represented by XML nodes. Additional nodes are used to specify the data. These nodes do not represent entities (with an ID).

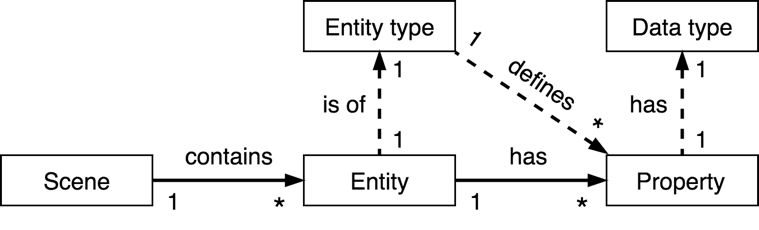


Figure 1: Scene meta model

## Behavior over time

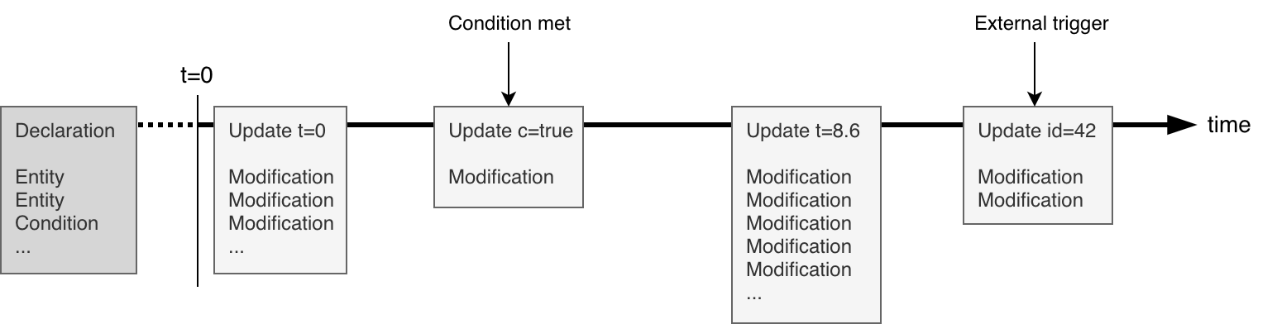
An MPEG-I audio scene can vary over time with the following behavior, which is illustrated in Fig. 2

* Entities in a scene cannot be created/deleted dynamically, they exist by declaration,
* Modifiable properties of some entities can change over time.

Changes in the scene are expressed by *updates*, which are defined as follows:

1. An *update* occurs at a specific point in time and  
   contains one or more *modifications*.
2. An *update*, can be initiated by a condition, manually triggered or dynamically updated by an external entity.
3. A *modification* changes one or more properties of a single entity.  
   Not every property of an entity can be changed (e.g. IDs cannot be changed).
4. An *update* can bundle several *modifications* for different entities.
5. Updates shall be handled *atomically*.

To clarify, ‘atomically’ means that encoders only observe the final outcome of an entire update, but not its individual modifications. This is essential to allow for changing multiple entities synchronously, e.g. moving several sources coherently.

Figure 2: Scene behavior over time

## Scene declaration

The EIF file consists of a single <AudioScene> node, in which all entities are declared (e.g. sound sources, acoustic environment, geometric elements, audio signals, etc.) and updates are defined. How the elements may be arranged is defined by a syntax tree, shown in Fig. 3. It defines the valid hierarchy (parent-child relations) and multiplicities. Elements visualized by rectangles are EIF entities (with an ID) represented by XML nodes. Rounded rectangles mark additional XML nodes used for defining the elements (e.g. data points). The allowed multiplicities are specified in UML format. *0..\** indicates that the parent node may have any number of these child nodes. *1..\** indicates that is must at least one child.

The following brief example defines a simple, time-invariant test scene with one MPEG-I Audio Object, that represents a trumpet with directivity. Note, that this example is neither complete nor does it specify any syntax of an ObjectSource or other entities.

<AudioScene>

<AudioStream id=”signal:trumpet”  
 file=”armstrong.wav”

aepInputChannels=”0, 1” />

<SourceDirectivity id=”dir:trumpet”

file=”trumpet.sofa” />

<ObjectSource id=”src:trumpet”

position=”2 1.7 -1.25”

orientation=”30 -12 0”

signal=”signal:trumpet”

directivity=”dir:trumpet”

gainDb=”-2”

active=”true” />

</AudioScene>

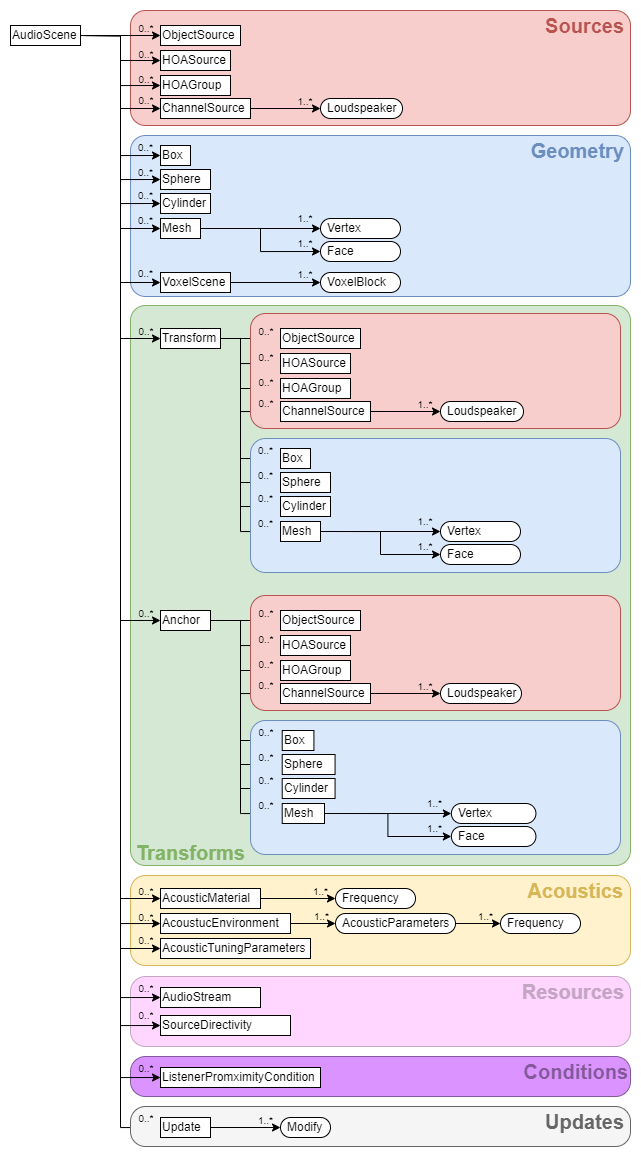


Figure 3: Structure of an EIF file

## Scene updates

The following types of updates can be defined in an EIF file

* A *timed update* (L1) is executed by the renderer once at a fixed, predefined time.
* A *conditional update* (L2) is initiated by the renderer when a certain condition is met.
* A *triggered update* (L2) is manually triggered from an external entity (using an OSC message).
* A *dynamic update* (L3) is an update from an external entity that includes the values of the attributes to be updated.

The attributes and values that an update modifies are defined in the EIF file and known to the encoder. The only exception are dynamic updates, which allow to insert parameter variables for attribute values to be modified. These values are provided to the renderer by the update message. Parameter variables are indicated by ‘$’ followed by a number.

For the CfP only the audio subgroup agreed to restrict the use of dynamic updates (L3) as follows:

* Dynamic updates (L3) may only modify the position, orientation, or gain of

maximally three audio elements.

These audio elements must not be inside a transform.

* Dynamic updates (L3) may only modify the position and orientation of geometric elements when no acoustic material is specified and when it is used as the extent of an audio element only.

After the CfP there are no restrictions on the use of Dynamic Updates (L3).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <Update> | | | | |
| Declares one or more changes to the audio scene. The update is performed, when   * the specified time is reached, or * the condition changed its state to the logical value expressed by fireOn, * the update is triggered by its ID or index by an external entity   The fireOn parameter determines whether the update fires when the condition changes from false-to-true (fireOn=”true”) or from true-to-false (fireOn=”false”). This is helpful for if-else type conditional updates.  An <Update> node has one or more <Modify> child nodes. | | | | |
| Child node | Count | Description | | |
| <Modify> | >=1 | Modifications (see below) | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | O\* | none | Identifier |
| index | Integer | O\* | none | Index identifying the update (globally unique) |
| time | Value | O\* | none | Time when the update is initiated (seconds).  Note: Must be less than or equal to the duration attribute of the AudioScene, and the update doesn’t affect the scene before this time stamp. |
| condition | Condition ID | O\* | none | Condition |
| fireOn | Boolean | O\* | true | Update fires when this state is reached |
| delay | Float >= 0 | O\* | 0 | Postpone the update (seconds) |

\* in accordance with Table 1

<Update> nodes can only be used in specific ways to make it possible to clearly identify the different update types. Table 1 gives an overview about the types of updates and the viable attributes and variables.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Update type | <Update>  Attributes  id, index | <Update>  Attribute  time | <Update>  Attribute  condition | <Update>  Attribute fireOn, delay | <Modify>  Attribute  duration | <Modify>  Parameter variables $0, $1, $2, … |
| Timed update | optional | required | not allowed | not allowed | optional | not allowed |
| Conditional update | optional | not allowed | required | optional | optional | not allowed |
| Triggered update | required | not allowed | not allowed | not allowed | optional | not allowed |
| Dynamic update | required | not allowed | not allowed | not allowed | not allowed | present |

Table 1: Allowed attributes and variables for different types of updates and modifications

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <Modify> | | | | |
| Declares a modification of modifiable parameters of a single entity. The target entity is selected by the id attribute. Following attributes must be attributes of the corresponding entity. The attribute values are assigned the entities property values.  For scene updates where attribute values are not available until rendering, typically user interactions, the attribute values are replaced by parameter indicators to specify in which order the attribute values are expected on the renderer interface. Parameter indicators are $0, $1, $2, etc.  The ‘duration’ attribute controls how the value change shall be handled. A value of zero indicates that an immediate switch to the new value is intended. Positive nonzero ‘duration’ values represent the timespan for a continuous (e.g. interpolated) changeover. How this is handled and realized is up to the renderer. The ‘duration’ attribute is not allowed for Dynamic updates (See Table 1).  **Example:**  <Modify id=”src1” position=”1 2 3” orientation=”-20 5 0” /> sets the attributes position and orientation for the entity with ID src1 | | | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Target entity to be modified |
| duration | Float >= 0 | *see Table 1* | 0 | Period for adapting from the current values to the new values (seconds) |
| \* | \* | \* | \* | Attribute of the target entity |

Note, that not every attribute can be changed. Only those entities that have an entity type specification that allows for modification can be modified (labelled ‘M’).

As updates are atomic, the following rules must be met to prevent ambiguities

* No two timed updates may have the same execution time.
* No two conditional updates may refer to the same condition with same firing order.
* No two triggered updates may have the same id or index.
* Any attribute of any entity may only be modified once within one update

Different updates types can be mixed to control an entity. However, scene creators are advised not to mix different update type for the same attribute of an entity, which may result in an undesired behavior. If an entities' attribute is statically animated (i.e. modified by timed updates), it may not be updated with conditional or triggered updates. Note that this distinction is on attribute-level not entity-level.



Figure 4: Example of a moving car modeled by several AudioElements

The following timed updates synchronously move three *ObjectSources* of a vehicle in motion (see Fig. 4) along a trajectory.

<Update time=”0.2”>

  <Modify id=”engine” position=”2.2 1.7 -1.25” />

  <Modify id=”tire1” position=”2.2 1.7 0.75” />

  <Modify id=”tire2” position=”2.2 1.7 -0.95” />

</Update>

<Update time=”0.4”>

  <Modify id=”engine” position=”2.4 1.7 -1.20” />

  <Modify id=”tire1” position=”2.4 1.7 0.70” />

  <Modify id=”tire2” position=”2.4 1.7 -0.95” />

</Update>

...

The scene loops at the rate of the scene duration as specified in the AudioScene’s attribute. Timed updates are triggered for every loop of the scene.

The following conditional updates turn on the sources of a car when the listener gets close.

<Box id=”geo:region1” position=”5 0 -5” size=”10 2 10” />

<ListenerProximityCondition id=”cond:listenerNearCar” region=”geo:region1” />

<!—- Turn on the engine sound 100ms after the listener entered

the region. Smoothly activate the source within 50ms. -->

<Update condition=”cond:listenerNearCar” delay=”0.1”>

<Modify id=”engine”   
 duration=”0.05” active=”true” />

</Update>

<!—- Turn on the other sources 100ms later from the engine -->

<Update condition=”cond:listenerNearCar” delay=”0.2”>

<Modify id=”radio” duration=”0.2” active=”true” />

<Modify id=”exhaust” duration=”0.1” active=”true”/>

</Update>

The following triggered updates are used to turn on/off a radio source.

<Update id=”upd:radio\_on” index=”0”>

  <Modify id=”src:radio” active=”true” />

</Update>

<Update id=”upd:radio\_off” index=”1”>

  <Modify id=”src:radio” active=”false” />

</Update>

This last example shows how the radio can be positioned dynamically (i.e. at presentation time) using a dynamic update. The position/orientation values are provided when the update is triggered by the external source (e.g. VR/AR engine).

<Update id=”upd:radio\_place” index=”2”>

  <Modify id=”src:radio” position=”$0 $1 $2” orientation=”$3 $4 $5” />

</Update>

# Entity types

In the following, entities and their properties are defined. These definitions were derived from the  
MPEG-I Audio requirements [2]. Each table lists the properties of an entity, their semantic, possible values, etc. The types of the properties are defined in the appendix. The flags show if a property is required ‘R’ (i.e. it must be defined) or ‘O’ (i.e. it can be defined, otherwise it has default value). ‘M’ indicates that a property can be modified using updates (see 2.1). Properties not labelled with an ‘M’ are static and can only be assigned in the entity’s declaration (see 2.2).

## Audio scene

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <AudioScene> | | | | |
| Declares an audio scene. Each scene description XML file must have exactly one AudioScene node. This node marks the root node of the static description of the audio scene. It contains all entities of the scene as children. | | | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Identifier |
| duration | Float | O | 0 | The duration of the scene, in seconds. If the value is zero, then the scene will not loop |
| type | String | O | ‘VR’ | Scene type. Set to ‘VR’ for VR scenes and ‘AR’  for AR scenes. |

In the evaluation procedure, the scene’s ID is used to identify different scenes for the test. This ID must be unique for all scenes in the evaluation. The type attribute informs encoders whether the scene is intended for AR or VR applications.

## Audio elements

Geometric elements, meshes and primitives, may be used to specify the extent of an audio source, indicating the geometry that the sound is intended to radiate from. Any mesh used must be manifold, and primitives must have all dimensions > 0.

If a geometric element has materials assigned, such that it is also used for acoustic effects, the materials are not taken into consideration for the rendering of the audio source that references it.

When a geometric element is specified for the spatial extent, the position, orientation and cspace of the source is taken to be that of the geometric element. The front and up directions (see Annex A.3) are taken from the geometric element.

For an ObjectSource, if the user is inside of the extent the sound field is assumed to be homogeneous, and does not change with user position.

For an HOA source the expected rendering behavior is as defined in Table 3.

## ObjectSource

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <ObjectSource> | | | | |
| Declares an ObjectSource which emits sound into the virtual scene. The ObjectSource has a position/orientation in space. The radiation pattern can be controlled by a directivity. If no directivity attribute is present, the source radiates omnidirectional. Optionally it can have a spatial extent, which is specified through a geometric object. If no extent is specified, the source is a point source. The signal component of the ObjectSource must contain at least one waveform. When the signal has multiple waveforms, the spatial layout of these waveforms must be specified in an <InputLayout> subnode (details see 3.2.1). | | | | |
| Child node | Count | Description | | |
| <InputLayout> | 0..1 | Signal positioning (required when signal has multiple waveforms) | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Identifier |
| position | Position | R, M |  | Position |
| orientation | Rotation | O, M | (0° 0° 0°) | Orientation |
| cspace | Coordinate space | O | relative | Spatial frame of reference |
| active | Boolean | O, M | true | If true, then render this source |
| gainDb | Gain | O, M | 0 | Gain (dB) |
| refDistance | Float > 0 | O | 1 | Reference distance (m) (see comment below) |
| signal | AudioStream ID | R, M |  | Audio stream |
| extent | Geometry ID | O, M | none | Spatial extent |
| directivity | Directivity ID | O, M | none | Sound radiation pattern |
| directiveness | Value | O, M | 1 | Directiveness (see 3.4.1) |
| aparams | Authoring parameters | O | none | Authoring parameters (see 4.13) |
| mode | Playback mode | O | continuous | Playback mode {“continuous”, “event”} |
| play | Boolean | O, M | False | Playback enabled? |

**Example:**

<!-- This transform marks the pivot point of the car.

It can be used for animations. -->

<Transform id=”tf:car”

position=”23.5 0.0 102.3”

orientation=”-12.4 0 0” />

<!—- The car’s source is place 0.4m above the ground and  
 0.8m towards the front (negative z-direction) -->

<ObjectSource id=”src:engine”

position=”0 0.4 -0.8”

signal=”signal:engine”

extent=”geo:car\_extent” />

<!-- This box is relative to the origin of the car.

It raises 1.4 meters high from the ground. -->

<Box id=”geo:car\_extent” position=”0 0.7 0” size=”1.6 1.4 4.5” />

</Transform>

### Extended ObjectSources with multi-channel signals

An ObjectSource with spatial extent can have a multi-channel AudioStream to give the renderer the possibility to render the ObjectSource with greater realism than what is possible with a mono AudioStream. This can for example be useful when rendering diffused audio sources such as fountains, waterfalls, rivers, breaking waves, etc.

An ObjectSource with an extent is always perceived by the listener in an elevation-azimuth sector from the listener. This sector is determined by the relative position of the ObjectSource with respect to the listener and the extent of the ObjectSource, all in an acoustical perceptual sense. This is exemplified in Figure 5a for an object source with a cylindrical extent where the ObjectSource is in the front-right hemisphere of the listener. The intersection of a plane that is orthogonal to the observation vector to the center of the elevation-azimuth sector and the elevation-azimuth sector specifies a rectangle. This rectangle represents the acoustically perceived horizontal and vertical extent of the ObjectSource by the listener from the position of the listener. As the listener moves around the ObjectSource, moves closer to it or further away from it, this rectangle will be translated, rotated and resized in the world space coordinate system. Figure 5b illustrates this when the cylindrical ObjectSource is positioned in the front-left hemisphere of the listener. But in the x-y coordinate system that has the origin at the center of these perceived extent rectangles, these rectangles are always positioned with the center in the (0, 0) point of the source.

|  |  |
| --- | --- |
| 1. observed in the front-right hemisphere of a listener | 1. observed in the front-left hemisphere of a listener |

Figure 5: Object Source with cylindrical extent and “user” alignment

An InputLayout child node of an ObjectSource is composed of an alignment flag and a string, containing positioning mnemonics separated by whitespaces:

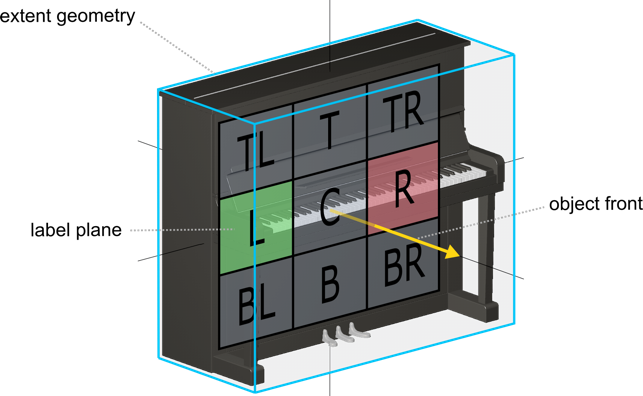
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <InputLayout> | | | | |
| Inside of an <ObjectSource> node, it describes the positioning and handling of several associated waveforms. | | | | |
| Attribute | Type | Flags | Default | Description |
| alignment | String | R |  | Indicates anchoring mode:   * “user” indicates that the labels are anchored by the user viewing direction * “object” indicates that the labels are anchored by the position/orientation of the ObjectSource element |
| positioning | String | R |  | Indicates intended spatial assignment of waveforms |

The alignment attribute defines the way how the waveforms (channels) of the associated audio stream are located/anchored with respect to the source. The positioning attribute is a string, containing mnemonic labels separated by whitespaces where for each waveform a mnemonic label has to be supplied. Nine relative position mnemonics referring to the previously described x-y coordinate system are supported in the as described in Table 2.

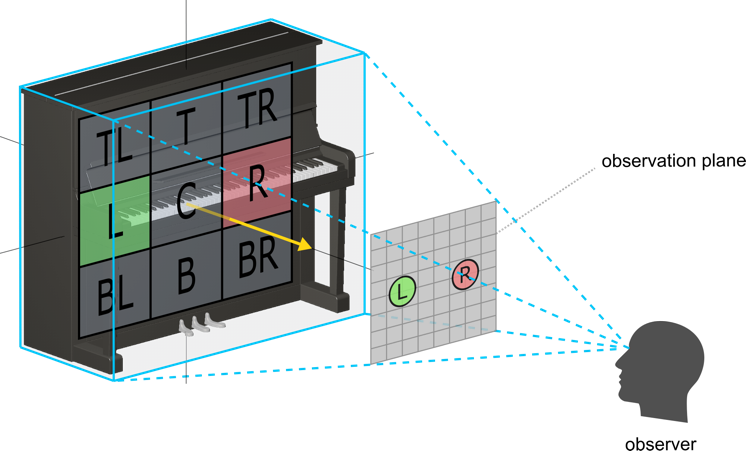
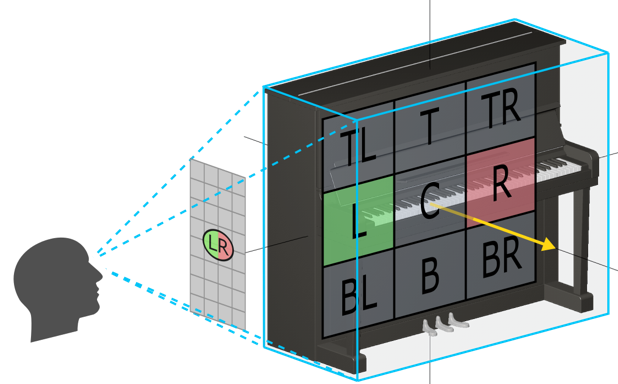
|  |  |  |
| --- | --- | --- |
| TL  top-left | T  top | TR  top-right |
| L  left | C  center | R  right |
| BL  bottom-left | B  bottom | BR  bottom-right |

Table 2: Relative signal channel positions

Alternatively to the anchoring of the labels relative to the listener viewing direction (as described in the initial example in Figure 5 with attribute alignment=”user”), the signal channel relative position labels can be used to indicate the usage of the waveforms for rendering an ObjectSource with size such that it is anchored to certain object in the scene (attribute alignment=”object”). An example is the sound of a Piano with one signal channel predominantly containing lower notes, the other containing predominantly higher notes. In this case, the position labels apply to a rectangle on a plane through the object position (center) that is perpendicular to the orientation of the ObjectSource when looking at its front (the ObjectSource’s ‘orientation’ attribute must be present). During rendering, the positions indicated by the labels are then projected to the user observation plane (plane that is orthogonal to the observation vector), as shown in Figure 6. Please note that this may also imply putting the sources (potentially with extent) ‘behind’ each other (when looking at the Piano from the side, see Figure 6c) or even swapping them (when looking at the Piano from the back).



1. Piano with orientation (front), extent geometry and label plane.

(b) Observed from the front of the Piano (c) Observed from the left of the Piano

Figure 6: ObjectSource (piano) with box-shaped extent and “object” alignment

**Examples:**

<InputLayout alignment=”user” positioning=”L R” />

indicates that 2 waveforms are used to render a horizontal width of a source.

<InputLayout alignment=”user” positioning=”B T” />

indicates that 2 waveforms are used to render vertical width of a source.

<InputLayout alignment=”user” positioning=”BL TL TR BR” />

indicates that 4 waveforms are used to render both horizontal and vertical width of a source.

<ObjectSource id=”src:piano”

position=”1.2 1.0 -0.3”

orientation=”38 0 0”

extent=”geo:piano\_extent”

signal=”signal:piano”>

<InputLayout alignment=”object” positioning=”L R” />

</ObjectSource>

indicates that 2 waveforms are used to render left and right of a Piano object.

## HOASource

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <HOASource> | | | | |
| Declares a source which emits sound into the virtual scene, that is described by higher-order Ambisonics (HOA). The HOASource has a position/orientation in space.    The HOASource can describe an incoming soundfield or an outgoing soundfield, as denoted by the representation attribute. An *internal* representation (representation=0) describes a sound field resulting from sources located outside the spatial sampling area (see Fig. 7a). An *external* representation (representation=1) describes a sound field resulting from sources located inside the spatial sampling area (see Fig. 7b). The spatial sampling area is defined by the extent attribute.    It is expected that a HOASource is also rendered outside its region of validity. For an internal HOASource the region of validity is inside the source extent, while for an external HOASource the region of validity is outside of the source extent. If such behavior is not desired, this is signaled t by setting the extentTransform attribute to False. The transitionDistance attribute determines the area in which the listener transitions between external and internal representation.    The HOASource can be rendered either in 3DOF or 6DOF, which is specified with the is6DOF Boolean attribute. The specification of this attribute in combination with the attributes extent,  extentTransform and cspace  makes it possible to specify a range of HOASource rendering behavior that is summarized in Table 3 and exemplified in the examples provided below. | | | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Identifier |
| position | Position | R, M |  | Position |
| orientation | Rotation | O, M | (0° 0° 0°) | Orientation |
| cspace | Coordinate space | O, M | relative | Spatial frame of reference Cannot be set to ‘user’, when is6DOF==True |
| active | Boolean | O, M | true | If true, then render this source |
| gainDb | Gain | O, M | 0 | Gain (dB) |
| signal | AudioStream ID | R, M |  | Audio stream |
| aparams | Authoring parameters | O | none | Authoring parameters (see 4.13) |
| mode | Playback mode | O | continuous | Playback mode {“continuous”, “event”} |
| play | Boolean | O, M | False | Playback enabled? |
| extent | Geometry ID | O, M | none | Region of validity, spatial extent |
| extentTransform | Boolean | O | True | When an extent is defined, turns on/off the external rendering of an interior source or the internal rendering of an exterior source. |
| transitionDistance | Value | O | 0 | When an extent is defined, determines the transition area between external and internal representation (m) (see Fig. 8). |
| representation | Integer | R |  | Interior or exterior HOA expansion  0 = interior, 1 = exterior |
| is6DoF | Boolean | O, M | False | If True, the source is rendered in 6DoF within its region of validity (or everywhere if no extent is defined). Also enables the following attributes when True: group, refDistance |
| group | ID | O, M | none | Parent HOAGroup |
| refDistance | Float >= 0 | O | 0 - interior representation  1 - exterior representation | Reference distance (m) (see Reference Distance section below)    For an internal HOASource the distance is from the origin/center of the extent, while for an external HOASource the distance is from the extent boundary into the region of validity in the direction of the normal vector. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <HOAGroup> | | | | |
| Declares a group of HOA and optionally associated object sources which represent multiple HOA and optionally associated object sampling points of a sound scene. HOASources in a HOAGroup are meant to be processed as a unit. | | | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Identifier |
| region | Geometry ID | O, M | none | Spatial region of validity |
| coSources | List of ObjectSource IDs | O | none | ObjectSources sampled simultaneously with the HOASources |



Figure 7: HOA sound field representations

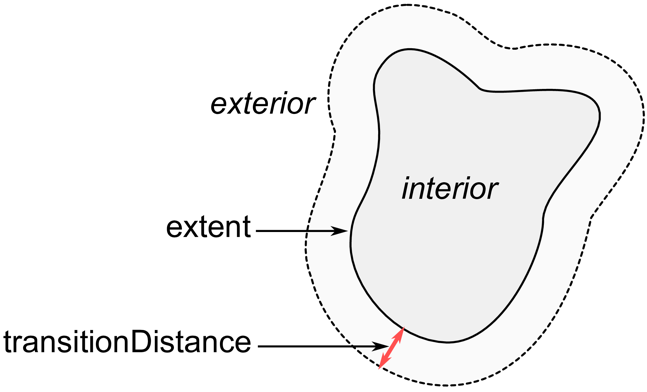


Figure 8: HOA transition distance

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **is6DoF** | **extent** | **extentTransform** | **cspace** | **HOA Rendering** |
| False | None | N/A | Relative | 3DoF everywhere in the scene |
| False | None | N/A | User | 0DoF everywhere in the scene |
| False | Defined | True | Relative | 3DoF within the region of validity  Renderer dependent outside of the region of validity |
| False | Defined | True | User | 0DoF within the region of validity  Renderer dependent outside of the region of validity |
| False | Defined | False | Relative | 3DoF within the region of validity  No rendering outside of the region of validity |
| False | Defined | False | User | 0DoF within the region of validity  No rendering outside of the region of validity |
| True | None | N/A | Relative | 6DoF everywhere in the scene |
| True | Defined | True | Relative | 6DoF within the region of validity  Renderer dependent outside of the region of validity |
| True | Defined | False | Relative | 6DoF within the region of validity  No rendering outside of the region of validity |
| True | None/  Defined | True/  False | User | Not valid |

Table 3 - Summary of HOA rendering behavior for combinations of is6DoF and extent/cspace attributes.

**Examples:**

***Example of legacy HOA sources rendered in 3DoF***

In this example, the HOA signal is premixed ambience content. With is6DoF set to “False”, it is assumed that the origins of the sounds in the HOA signal are sufficiently far away (or intangible like the wind) such that only 3DoF rendering is required. Note that no extent is specified and cspace is “relative” so that the HOA rendering is fixed to the scene.

<AudioStream id="signal:ambienceHOA"

aepInputChannels="1:16"

file="ambienceHOA.wav"/>

<HOASource id="src:ambience"

signal="signal:ambienceHOA"

position="0, 0, 0"

orientation="30, 0, 0"

cspace="relative"

mode="continuous"

play="true"

representation="0"

is6DoF="False"/>

***Example of 3DoF HOA with extent***

If the HOA signal is music content to be used as background music following the listener, then cspace would be set to “user”. In this case the HOA source’s orientation is always aligned with the listener.

<AudioStream id="signal:musicHOA"

aepInputChannels="1:16"

file="musicHOA.wav"/>

<HOASource id="src:backgroundmusic"

signal="signal:musicHOA"

orientation="0, 0, 0"

cspace="user"

mode="continuous"

play="true"

representation="0"

is6DoF="false"/>

In this example the listener can explore the inside and outside of an aviary. The HOASource includes an HOA signal with bird sounds and an extent with the same geometry as that of the aviary. The HOASource will be rendered as a 3DoF HOA signal when the listener is inside the aviary. It will be up to the renderer to transition from 3DoF to an unspecified rendering as the listener travels the 2 meters of transitionDistance beyond the extent. When the listener is outside the aviary (outside the extent), it is expected that the listener hears the bird sounds as originating from inside the aviary.

<AudioStream id="signal:aviaryHOA"

aepInputChannels="1:16"

file="aviaryHOA.wav"/>

<HOASource id="src:aviary"

signal="signal:aviaryHOA"

orientation="30, 0, 0"

position="0, 0, 0"

cspace="relative"

mode="continuous"

play="true"

extent="geom:aviary\_extent"

extentTransform="True"

transitionDistance="2"

representation="0"

is6DoF="False"/>

## ChannelSource

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <ChannelSource> | | | | |
| Declares a source which renders multi-channel signals, belonging to a loudspeaker setup. The ChannelSource has a position/orientation in space. The signal ~~component~~ of the ChannelSource must contain the same number of waveforms as indicated by the *inputLayout* attribute. | | | | |
| Child node | Count | Description | | |
| <Loudspeaker> | >=1 | Virtual loudspeaker (see below) | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Identifier |
| position | Position | R, M |  | Position |
| orientation | Rotation | O, M | (0° 0° 0°) | Orientation |
| cspace | Coordinate space | O, M | relative | Spatial frame of reference |
| inputLayout | CICP layout | R |  | Speaker layout of the original audio signal |
| active | Boolean | O, M | true | If true, then render this source |
| gainDb | Gain | O, M | 0 | Gain (dB) |
| refDistance | Float > 0 | O | 1 | Reference distance (m) (see comment below) |
| signal | AudioStream ID | R, M |  | Audio stream |
| aparams | Authoring parameters | O | none | Authoring parameters (see 4.13) |
| mode | Playback mode | O | continuous | Playback mode {“continuous”, “event”} |
| play | Boolean | O, M | False | Playback enabled? |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <Loudspeaker> | | | | |
| Inside of an <ChannelSource> node, it declares single virtual loudspeaker. The position and orientation are specified relative to the parent ChannelSource’s position/orientation. Loudspeakers are not modifiable at runtime. | | | | |
| Attribute | Type | Flags | Default | Description |
| position | Position | R |  | Position |
| orientation | Rotation | O | (0° 0° 0°) | Orientation |
| channel | Integer | R |  | Channel index in the associated AudioStream |
| directivity | Directivity ID | O | none | Associated loudspeaker radiation pattern |
| directiveness | Value | O | 1 | Directiveness (see 3.5) |

**Example:**  
<!-- Example of a five speaker layout according to CICP layout 5 -->

<ChannelSource id=”src1” position=”1.2 1.65 -2.10” signal=”sig:spk” inputLayout=”5”>

<Loudspeaker position=" 0 0 -1.8" orientation=” 180 0 0” channel=”0”/>

<Loudspeaker position="-1 0 -1.4" orientation=”-145 0 0” channel="1"/>

<Loudspeaker position=" 1 0 -1.4" orientation=” 145 0 0” channel="2"/>

<Loudspeaker position="-1.4 0 1.4" orientation=” -45 0 0” channel="3"/>

<Loudspeaker position=" 1.4 0 1.4" orientation=” 45 0 0” channel="4"/>

</ChannelSource>

**Remarks**

* When the signal of an AudioElement is modified, the new AudioStream must have the same numbers of channels as the original AudioStream.

## Reference distance and playback control

**Reference distance**

Certain audio elements are rendered in dependence on the listener’s distance to the audio source (“distance attenuation”). The reference distance field allows to specify a distance at which – independently from the distance attenuation law used – the computed attenuation of the audio element is 0 dB (see Fig. 9). Thus, different renderer implementations will deliver the same level balance between audio elements for points within a reference distance and thus sound comparable (assuming free-field conditions and no room reflections). Note that the reference distance is independent of the gain. For a ChannelSource, the same reference distance is used for all virtual loudspeakers.

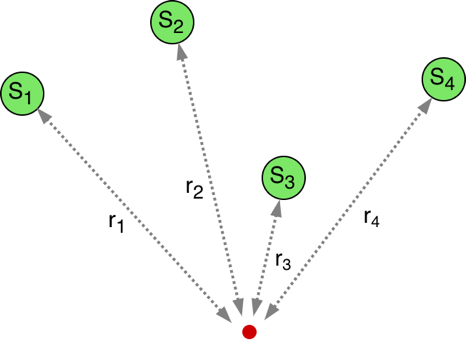


Figure 9: Reference distances of AudioElements to an imaginary reference location (red)

### Playback control

Audio Elements feature two attributes that allow controlling the playback of sound. The mode attribute controls the way the playback behaves: It can either be event-driven or continuous. Both modes have a different behavior:

* In mode=”continuous” the sound is continuously playing from the beginning of the scene.  
  In this mode, the play attribute has no effect.
* In mode=”event” the sound playback is triggered by setting the play attribute to true. When the playback is finished, the play attribute is automatically reset to false. The playback always starts from the beginning of the signal.

The following example illustrates how a sound is played triggered by an event. The playback control is formulated as an update:

<AudioStream id=”sig:doorHandle1” file=”doorHandle1.wav” />

<AudioStream id=”sig:doorHandle2” file=”doorHandle2.wav” />

<ObjectSource id=”src:doorHandle” signal=”sig:doorHandle1” mode=”event” />

<Update id=”upd:doorOpen”>

<!-— Start playback of sound 1 -->

<Modify id=”src:doorHandle” play=”true” />

</Update>

<Update id=”upd:doorOpen2”>

<!-— Start playback of sound 2 -->

<Modify id=”src:doorHandle” signal=”sig:doorHandle2” play=”true” />

</Update>

The following example illustrates how an ObjectSource is enabled/disabled when the listener enters/leaves its proximity:

<!—- Define a proximity controlled playback -->

<ListenerProximityCondition id=”cond:listenerInside” region=”geo:region1” />  
<Box id=”geo:region1” position=”1 0 4” size=”10 5 50” />

<Update condition=”cond:listenerInside”>

<Modify id=”src:musician” active=”true” />

</Update>

<Update condition=”cond:listenerInside” fireOn=”false”>

<Modify id=”src:musician” active=”false” />

</Update>

## Spatial transforms

Multiple spatial entities (see Fig. 3) can be grouped and be translated/rotated together. Such spatial parenting (as found in classical scene graphs) is realized using a <Transform> node. In order to limit the complexity, nesting of transforms is not allowed.

Note that the geometry specification using voxels <Voxels> as specified in Section 3.7 is not applicable to <Transform> node.

A transform defines a frame of reference, called transform coordinate system (see Fig. 10). It is derived from the global coordinate system by translation and rotation, specified for the transform. Elements placed within the transform, having cspace=”relative” selected, expresses their position and orientation within this transform’s system.

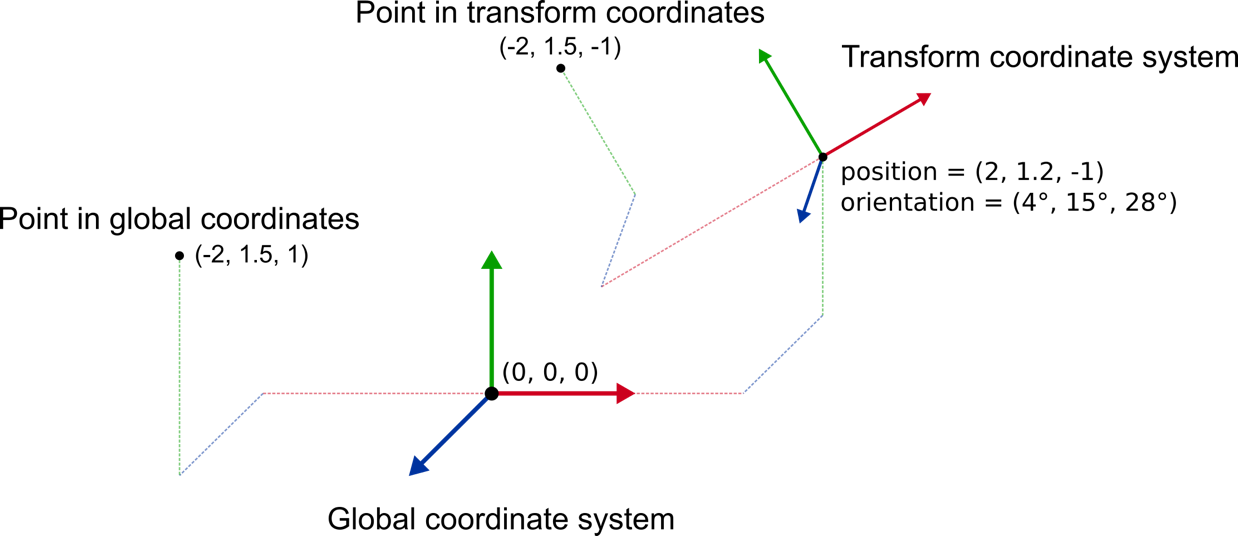


Figure 10: Global coordinates and transform coordinates

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <Transform> | | | | |
| Declares a spatial transform node that translates and/or rotates child entities.  Note that position or orientation or both must be specified.  Selecting cspace=’listener’ attaches the transform to the listener. This allows positioning entities shifted/rotated relative to the listener’s head. | | | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Identifier |
| position | Position | O, M | (0 0 0) | Position |
| orientation | Rotation | O, M | (0° 0° 0°) | Orientation |
| cspace | Coordinate space | O | relative | Spatial frame of reference |

<Transform id=”tf:bus” position=”12.3 0 -50.2” orientation=”90 0 0”>

<ObjectSource id=”src:tire1” position=”...” ... />

<ObjectSource id=”src:tire2” position=”...” ... />

<ObjectSource id=”src:engine” position=”...”

extent=”ext.engine” ... />

<Box id=”ext:engine” ... />

<Box id=”geo:bus\_occluder” material=”mat:bus\_occluder” ... />

</Transform>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <Anchor> | | | | |
| Implements a spatial transform used for placing objects in an Augmented Reality (AR) setting. The position and orientation are fixed and can only be defined once, before the scene is started.  The audioSourceRescale factor is applied to the position of child audio sources (object, Channel, HOA Source, HOA Group), The order of application is translation, rotation and then rescaling. No audio source may have extent when used with rescaling. | | | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Identifier |
| lsdf\_ref | ID | R |  | Identifier of the corresponding LSDF anchor |
| audioSourceRescale | Rescale Factor | O, M | (1 1 1) | Rescale factor to be applied to the position and dimensions of all audio sources (Object, Channel, HOA Source, HOA Group) nodes of the transform, in x-, y-, z- dimensions. |
| autoRescale | Bool | O | False | Enable the renderer to determine the rescale factor by comparing the Acoustic Environment region to the one defined in the LSDF. |

<Anchor id=”tf:picture\_on\_the\_wall” lsdf\_ref=”picture\_anchor”>

<ObjectSource id=”src:voice” position=”…” ... />

<Box id=”ext:picture” ... />

</Anchor>

The content creator can use the audioSourceRescale attribute to change the scale of the audio sources in the scene for an optimal experience. For example, to shrink a concert experience to fit in a smaller listener space. If the scene is flexible to fit any listener space size then they can choose to set the autoRescale attribute to True.

## Conditions

The following condition can be used to trigger updates inside the renderer:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <ListenerProximityCondition> | | | | |
| This condition expresses if the listener (user) resides within a certain spatial region (volume). The region is specified by a geometric object (see Sec. 3.4).  Note: When a mesh is used to specify this spatial volume, it must be ensured by the author, that the mesh has a closed hull (no openings). | | | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Identifier |
| region | Geometry ID | R |  | Spatial region |

## Audio resources

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <AudioStream> | | | | |
| Declares an audio stream provided by a local audio file. AudioElements can use it as signal source.  The file can have any number of channels but must be at a sampling rate of 48000 Hz. For the evaluation platform, input channels for the renderers have to be specified (indices starting with 0). | | | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Identifier |
| file | String | R |  | Path of the audio file (relative to scene.xml file) |
| aepInputChannels | List of integers or integer range | R |  | List of the channel indices, that the  renderers are supplied with the signal in the Audio Evaluation Platform (AEP) |

### Original signals

The original audio files have to be in Microsoft WAV format, sampled at a rate of 48000 Hz. Each Audio Element shall obtain its audio signal from a single audio file (i.e. interleaved if multichannel). For HOA, the channel order of the original audio stream must be ACN and the normalization method must be SN3D.

### Encoded/decoded signals

The MPEG-H 3D Audio is used to encode and decode all original audio signals. The resulting (uncompressed) WAV files are sampled at a rate of 48000 Hz. Each Audio Element shall obtain its audio signal from a single audio file (i.e. interleaved if multichannel). The HOA audio data in the HOA output interface is provided in the so-called equivalent spatial domain representation. The conversion from the HOA domain into the equivalent spatial domain representation and vice versa is described in Annex C.5.1 of ISO/IEC 23008-3.

### Channel routing in the evaluation

Note, that the aepInputChannels attribute is used for the evaluation only, but not an MPEG-I bitstream. The evaluation platform has to supply all audio streams in the scene as input channels to the audio renderers. In order to match the routing in the AEP, each AudioStream must specify its target input channels as provided to the renderers by the AEP. The Max patch reads this information, loads the audio files, provides them as audio streams and maintains the correct routing for the renderers. aepInputChannels can be a list of channel indices separated by commas. Alternatively, it can be specified as an integer range (format ‘startIndex:endIndex’).

**Example:**

<!-- Stereo signal routed to renderer input channels 0 and 1 -->

<AudioStream id=”signal:piano” file=”ellington.wav”

aepInputChannels =”0, 1” />

<!-- Mono signal routed to renderer input channel 2 -->

<AudioStream id=”signal:trumpet” file=”armstrong.wav”

aepInputChannels =”2” />

<!—- A 5.1 signal routed to renderer input channels 3..8 -->

<AudioStream id=”signal:drums” file=”drums51.wav”

aepInputChannels =”3:8” />

<!-- Mono signal routed to renderer input channel 9 -->

<AudioStream id=”signal:voice” file=”ella.wav”

aepInputChannels =”9” />

<!—- HOA order 2 signal routed to renderer input channels 10..18 -->

<AudioStream id=”signal:ambience” file=”ambienceHOA2.wav”

aepInputChannels =”10:18” />

## Source radiation patterns

Sound radiation patterns (directivities) of ObjectSources are described by a finite number of sampled directions and a range of different frequencies. A directivity is an additional direction-dependent filter applied to the source. Such data is loaded from SOFA files [3] or other equivalent formats. Each data point describes the transmission characteristic in this direction. This can be magnitude responses in octave or 1/3 octave frequency resolution. In the SOFA format, this is specified in the *generalTF* scheme [3]. Whenever possible, the set of directions shall cover the full sphere. Arbitrary spherical grids are supported. When a directivity is assigned to an ObjectSource the following alignment is assumed: The frontal direction of the directivity (0°, 0°) points into -z direction of the source’s local coordinate axes, the left direction (90°, 0°) to -x, and the up direction (0°, 90°) to +y. A local SOFA file representing a radiation patterns shall use a spherical coordinate system following the EIF coordinate convention as described in Annex A.2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <SourceDirectivity> | | | | |
| Declares a source radiation pattern provided by a local file. ObjectSources can be associated such a directivity. | | | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Identifier |
| file | String | R |  | Path of the directivity file (relative to scene.xml file) |

### Directiveness

The directiveness parameter controls the intensity of the directivity over all frequency bands. The directivity factors are altered by the function . According to this equation, the value keeps the original directivity. Values weaken the directiveness, until results at omnidirectional directivity. Values emphasize the directiveness.

### Normalization

It is assumed that when a directivity is assigned to an ObjectSource, that all normalization required is inherent in the directivity file, and that the ObjectSource audio stream has been produced or filtered such that the application of the directivity filters does not produce unwanted audio artifacts or frequency-domain filtering.

## Geometric elements

Geometric information is fundamental for a 6DoF scene in order to realize spatially-dependent room acoustic effects, particularly diffraction and occlusion. Introducing abstract geometric elements enables an elegant solution for aspects beyond merely room models. A scene can contain geometric elements to

* specify the spatial extent of ObjectSources (e.g. the source width)
* describe Acoustic Elements (e.g. occlusion, diffraction and reflection, etc.)
* spatially decompose a scene into sub scenes (e.g. audibility ranges of AudioElements)

Three geometry concepts are supported:

* **Primitives**, such as a box, sphere or cylinder (see Fig. 11), which are mathematically modeled (without a spatial discretization) and thus well suited for bounding volumes.
* **Meshes** can be used to approximate arbitrary shapes (with a spatial discretization). A mesh consists of a list of vertices (3D coordinates) and a number of triangular faces (i.e. the indices of three vertices). Meshes can be for specifying geometrical models of the environment (e.g. a room).
* **Voxels** can be used to approximate arbitrary shapes (with a spatial discretization). A voxel represents a volume referenced by indexes of a regular grid in three-dimensional space. Each voxel is associated with acoustically relevant properties and audio rendering parameters.

Geometric elements can be associated with an acoustic material. Otherwise, they are considered acoustically transparent.

### Primitives

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <Box> | | | | |
| Declares a box with three spatial dimensions (see Fig. 11a). The origin of a box is always in its center. | | | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Identifier |
| position | Position | R, M |  | Position |
| orientation | Rotation | O, M | (0° 0° 0°) | Orientation |
| cspace | Coordinate space | O, M | relative | Spatial frame of reference |
| size | Dimensions | R, M |  | Size (x-, y-, z-direction) |
| material | Material ID | O | none | Associated acoustic material |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <Sphere> | | | | |
| Declares a sphere (see Fig. 11b). Selecting different spatial dimensions (x, y, z) warps the sphere into an ellipsoid.  The origin of a sphere is always in its center. | | | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Identifier |
| position | Position | R, M |  | Position |
| orientation | Rotation | O, M | (0° 0° 0°) | Orientation |
| cspace | Coordinate space | O, M | relative | Spatial frame of reference |
| size | Dimensions | R, M |  | Size (x-, y-, z-direction) |
| material | Material ID | O | none | Associated acoustic material |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <Cylinder> | | | | |
| Declares a cylinder (see Fig. 11c). The rotation axis is y. The size in y-direction specifies the height, whereas the sizes in x- and z-direction define the cross section. Equal sizes for the x- and z-direction result in circular cross section. Selecting different spatial dimensions for x and z warps the cross section to an ellipse. The origin of a cylinder is always in its center. | | | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Identifier |
| position | Position | R, M |  | Position |
| orientation | Rotation | O, M | (0° 0° 0°) | Orientation |
| cspace | Coordinate space | O, M | relative | Spatial frame of reference |
| size | Dimensions | R, M |  | Size (x-, y-, z-direction) |
| material | Material ID | O | none | Associated acoustic material |

When one of the three size dimensions is zero, the element becomes 2D (plane, disc, etc.). The surface of a primitive faces towards the outside. When an acoustic material is assigned to a primitive, it is applied on the outer hull of the primitive. Note: The inside of a shoebox room cannot be defined by a single primitive.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1. Box | 1. Sphere | (c) Cylinder |

Figure 11: Geometric primitives

### Meshes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <Mesh> | | | | |
| Declares a triangle mesh. A mesh consists of a list of vertices (3D coordinates) and a number of triangular faces (i.e. the indices of three vertices). Meshes can be used to describe arbitrary geometry. Any <Mesh> node has to have one or more <Vertex>, <Face> child nodes, defining points and triangles. | | | | |
| Child node | Count | Description | | |
| <Vertex> | >=1 | Vertex (see below) | | |
| <Face> | >=1 | Face (see below) | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Identifier |
| position | Position | O, M | (0, 0, 0) | Position (origin of the mesh) |
| orientation | Rotation | O, M | (0° 0° 0°) | Orientation |
| cspace | Coordinate space | O, M | relative | Spatial frame of reference (of the entire mesh, but not its vertices) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <Vertex> | | | | |
| Inside of a <Mesh> node, it declares a vertex (point) for spanning a triangle. Vertices have an index making them referenceable (e.g. in a Face). Vertices are always expressed in the mesh’s local coordinate system, i.e. relative to the mesh’s position. | | | | |
| Attribute | Type | Flags | Default | Description |
| index | Integer number | R |  | Index of the vertex (unique integer) |
| position | Position | R |  | Position (relative to the mesh’s origin) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <Face> | | | | |
| Inside of a <Mesh> node, it declares a triangle spanning three vertices. Vertices are selected by their indices. The order of vertices matters, as it affects the faces normal, i.e. determining which side is front/back. Faces themselves have an index in order to make them referenceable. | | | | |
| Attribute | Type | Flags | Default | Description |
| index | Integer number | R |  | Index of the face (unique integer) |
| vertices | List of three indices | R |  | Vertex indices |
| material | Material ID | O | none | Associated acoustic material |

Faces of a mesh are (acoustically) one-sided only. The order of the vertices in a triangle is important, as it determines the normal direction (front). Given a triangle its front is considered the side, where its normal vector points to. Matching OpenGL conventions, the normal of a triangle is calculated by the cross product of two edge vectors .

**Examples:**

<!-- Define a box -->

<Box id=”room\_bounds”

position=”2.2 0.0 -3.5”

rotation=”0.0 45.0 0.0”

size=”12.5 3.2 8.4”  
 material=”marble” />

<!—- Define a sphere -->

<Sphere id=”src1\_region”

position=”-2.0 0.6 1.0”

size=”1.0 1.0 1.0”

material=”marble” />

<!—- Define a mesh -->

<Mesh id=”room\_surface”>

<Vertex index=”0” position=”1 2 3” />

<Vertex index=”1” position=”0.2 -1 0” />

<Vertex index=”2” position=”2 2 1” />

<Vertex index=”3” position=”1 0.5 -0.1” />

...

<Face index=”0” vertices=”0 1 2” material=”upholstery” />

<Face index=”1” vertices=”0 2 3” />

...

</Mesh>

### Voxels

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <VoxelScene> | | | | |
| Declares voxel scene dimensions and voxel scene spatial resolution. | | | | |
| Child node | Count | Description | | |
| <VoxelBlock> | >=1 | Voxel block element specification (see below) | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Identifier |
| vox\_resolution | Voxel resolution | R |  | Voxel scene resolution (x-, y-, z-direction) |
| position\_s | Position | R, M |  | Start position (x-, y-, z-coordinates) of the first voxel center defining the scene |
| position\_e | Position | R, M |  | End position (x-, y-, z-coordinates) of the second voxel center defining the scene |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <VoxelBlock> | | | | |
| Declares a voxel block element defining a set of voxels sharing the same acoustic material attribute. A voxel scene is defined by a list of voxel block elements. Sequentially defined block elements override the overlapping voxel data. | | | | |
| Attribute | Type | Flags | Default | Description |
| material | Material ID | O, M | ”” (empty string) | Associated acoustic material |
| grid\_coordinate\_s | Voxel Grid Position | R, M |  | Start grid position (x-, y-, z-indices) of the first voxel defining the block element sharing the same acoustic material attribute (range [1.. vox\_resolution] for each dimension) |
| grid\_coordinate\_e | Voxel Grid Position | R, M |  | End grid position (x-, y-, z-indices) of the second voxel defining the block element sharing the same acoustic material attribute (range [1.. vox\_resolution] for each dimension) |

Engineering drawing

Description automatically generated with medium confidence Company name

Description automatically generated with medium confidence

Figure 1 — Example of   
voxel scene (left): green voxels — scene geometry; red voxel — first voxel and blue voxel — second voxel defining the scene dimensions;voxel block element defining the scene geometry (right): green voxels — block element defining the scene geometry; red voxel — first voxel forming the block element; blue voxel — second voxel forming the block element

Sequentially defined block elements extend the voxel scene. All overlapping voxel data is re-defined by subsequent defined block elements. This enables a voxel scene representation by small number of voxel block elements.

**Example:**

**A screenshot of a computer

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated with medium confidence**

**Figure 2 — Example of voxel scene definition by means of the sequence of overlapping voxel block elements**

<AudioScene id=”Demo” duration=”10”>

<!—Define a voxel scene 🡪

<VoxelScene id="half\_opened\_door" position\_s="-1.5 -1.5 -0.5" position\_e="1.5 1.5 0.5" vox\_resolution="16 16 16">

<!-- The voxel scene is composed of at least one VoxelBlock -->

<VoxelBlock grid\_coordinate\_s="1 1 1" grid\_coordinate\_e="16 16 16" material="" />

<VoxelBlock grid\_coordinate\_s="1 8 1" grid\_coordinate\_e="16 9 16" material="wall" />

<VoxelBlock grid\_coordinate\_s="5 8 1" grid\_coordinate\_e="12 9 13" material="door" />

</VoxelScene>

<!-- Defines an update to open the door in two steps over 1.5sec -->

<Update index="1">

<Modify id="half\_opened\_door" grid\_coordinate\_s="5 8 1" grid\_coordinate\_e="8 9 13" duration="0.5" material="" />

<Modify id="half\_opened\_door" grid\_coordinate\_s="5 8 1" grid\_coordinate\_e="5 10 13" duration="1.0" material="" />

</Update>

</AudioScene>

Objects represented using voxels are addressed by their corresponding material-object IDs. To represent the effect of geometry transformation/animation within the voxel scene, material-object IDs of the corresponding voxels are modified in a “frame-by-frame” manner. It is done by the scene update mechanism (similar to 3D sprite animation), where each update describes all changed voxel elements (or even the entire scene). The corresponding scene changes are defined by external scene authoring tools or voxel scene presentation engine.

Note that the voxel-based geometry specification for the following attributes/nodes cannot be used by the current encoder/renderer state:

extent (in <ObjectSource>, <HOASource> and <HOAGroup>)

region (in <AcousticEnvironment> and <ListenerProximityCondition> and <URR>)

If a scene is authored from “voxel-only” geometry representations, the MPEG-I Audio EIF description should contain the corresponding extent and region specification(s) in mesh format. These mesh parts can be obtained by the application of voxel-to-mesh conversion algorithms (e.g., marching cube algorithm) prior the EIF scene description.

## Acoustic materials

Acoustic materials are characterized by four parameters illustrated in Fig. 12:

* Specular reflected energy is reflected back in a distinct outgoing direction
* Diffuse reflected energy is diffusely scattered back from the material
* Transmitted energy passes through the material without changing the sound’s direction
* Coupled energy excites vibrations in the structure and is reemitted by the entire structure

Each value is a number in the range . The sum of these four values, if specified, must not exceed one:

. The remaining energy is dissipated energy that is absorbed by the material (turned into heat). Materials are assumed to be isotropic (i.e. rotation-invariant). As acoustic materials behave differently depending on the frequency (wavelength), these coefficients are typically measured and tabulated in octave or 1/3-octave resolution.



Figure 12: Acoustic material characterization

Such data can be found in the data sheets of material manufacturers, textbooks [3, 4] and in publicly available databases [5, 6]. The material’s behavior outside its definition ranges (above/below) shall be extrapolated (last known values are repeated). When only a single frequency is specified, the material properties are frequency-independent.

Authors are encouraged to design scenes in a way that geometric objects with acoustic materials do not overlap in space. For sound transmissions (transmission and coupling, see Fig. 12), one has to distinguish between primitives and meshes. A primitive has a common transmission characteristic by definition (only one material can be assigned to the primitive). This transmission characteristic considers the case that sound from outside of the primitive is transmitted through the primitive. The transmission behavior to a listener inside the primitive is undefined.

Meshes having multiple faces with potentially different acoustic materials are regarded as follows: When a sound path intersects the mesh or parts of it (e.g. multiple faces), the transmission is only effective, when the path intersects a face on its frontal side (normal side). Given that a path intersects a face from its backside, the sound is unaffected. When a sound path intersects the same concave shape multiple times, each front-side intersection accumulates the sound transmission.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <AcousticMaterial> | | | | |
| Declares an acoustic material. A material characterizes the acoustic behavior of surfaces in a 3D model. A material’s properties are expressed frequency-dependent by absorption and optionally scattering coefficients.  Any <AcousticMaterial> node has to have one or more <Frequency> child nodes. | | | | |
| Child node | Count | Description | | |
| <Frequency> | >=1 | Data specification (see below) | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Identifier |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <Frequency> | | | | |
| Inside of an <AcousticMaterial> node, it declares the material's acoustic behavior at a specific frequency.  A requirement for the coefficients is that . | | | | |
| Attribute | Type | Flags | Default | Description |
| f | Float | R |  | Frequency in Hertz |
| r | Float | O | 0 | Specular reflection coefficient (range 0..1) |
| s | Float | O | 0 | Diffuse scattering coefficient (range 0..1) |
| t | Float | O | 0 | Transmission coefficient (range 0..1) |
| c | Float | O | 0 | Coupling coefficient (range 0..1) |

**Examples:**

<!—- Frequency-independent material -->

<AcousticMaterial id=”marble”>

<Frequency f="0.0" r=”0.85” s=”0.05” c=”0.05” />

</AcousticMaterial>

<!—- Frequency-dependent material -->

<AcousticMaterial id=”upholstery”>

<Frequency f="125.0" r=”0.5” />

<Frequency f="250.0" r=”0.42” />

<Frequency f="500.0" r=”0.30” />

<Frequency f="1000.0" r=”0.21” />

<Frequency f="2000.0" r=”0.15” />

<Frequency f="4000.0" r=”0.06” />

</AcousticMaterial>

## Acoustic Environments

An <AcousticEnvironment> describes the acoustic conditions within the entire scene or a certain spatial zone by means of room acoustic parameters. The conditions are specified at one or more points in the environment. Some parameters are frequency-independent (e.g. pre-delay), while others are specified for different frequencies (e.g. reverberation time, diffuse-to-direct ratio). The <AcousticEnvironment> provides hints on the envisioned environment’s acoustic behavior. Authors of scenes are encouraged to report how they determined the acoustic parameters of their scene. Hereafter, the different parameters are explained.

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. Please note that the specifics of this definition are clarified in [4].

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 and thus an attribute of <AcousticParameters>. Please note that the specifics of this definition are clarified in [4].

The Diffuse-to-Direct-Ratio (DDR) 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 ddr 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 ddr values do not influence the amplitude of the direct sound in the process of rendering. While DDR is a general description of a room’s acoustic properties, rendering reverberation using DDR requires taking into account the source’s directivity pattern to find the total emitted energy from the PCM signal’s reference level. DDR 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. Please note that the specifics of this definition are clarified in [4].

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <AcousticEnvironment> | | | | |
| Declares an acoustic environment. The environment is characterized by acoustic parameters at a number of points in space. Additionally, a bounding volume can be specified. | | | | |
| Child node | Count | Description | | |
| <AcousticParameters> | >=1 | Acoustic parameters (see below) | | |
| Attribute | Type | Flags | Default | Description |
| id | ID | R |  | Identifier |
| region | Geometry ID | O | none | Region in which properties are specified |

Note that the geometry specifying the region of an acoustic environment must be static to the encoder and may not be changing at presentation time.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <AcousticParameters> | | | | |
| Inside of an <AcousticEnvironment> node, it declares the environment’s acoustic behavior at a specific point in space. | | | | |
| Child node | Count | Description | | |
| <Frequency> | >=1 | Frequency data specification (see below) | | |
| Attribute | Type | Flags | Default | Description |
| position | Position | R |  | Position |
| predelay | Float | O | none | Pre-delay time in seconds |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <Frequency> | | | | |
| Inside of a <AcousticParameters> node inside an <AcousticEnvironment> node, it declares the environment’s acoustic behavior at a specific frequency. | | | | |
| Attribute | Type | Flags | Default | Description |
| f | Float | R |  | Frequency in Hertz |
| rt60 | Float | R |  | Reverberation time (RT60) in seconds |
| ddr | Float | O | none | Diffuse-to-reverberant ratio  Note: ddr can only be specified, if the predelay is defined |

**Example:**

<!—- Region where these conditions shall be simulated (optional) -->

<Box id=”geo:cube1” position=”1 2 -4” size=”10 3.4 12” />

<AcousticEnvironment id=“room1“ region=“geo:cube1”>

<!-— Measurements performed at position 1 -->

<AcousticParameters position=”4 2 6.5” predelay=”0.075”>

<Frequency f="63.0" rt60="0.82" ddr=”0.3” />

<Frequency f="125.0" rt60="1.23" ddr=”0.25” />

<Frequency f="250.0" rt60="1.49" ddr=”0.22” />

<Frequency f="500.0" rt60="1.05" ddr=”0.2” />

</AcousticParameters>

<!-— Measurements performed at position 2 -->

<AcousticParameters position=”-1 0.3 1.2” predelay=”0.062”>

<Frequency f=”500.0” rt60=”0.96” ddr=”0.24” />

<Frequency f=”1000.0” rt60=”0.89” ddr=”0.20” />

<Frequency f=”2000.0” rt60=”0.82” ddr=”0.14” />

</AcousticParameters>

</AcousticEnvironment>

## Acoustic Tuning Parameters

Acoustic tuning parameters control scene wide acoustic effects, and are optional but with default values applied otherwise.

The attribute defaultAE enables a default acoustic environment. With this enabled a default acoustic environment will be applied to all regions of the scene where no other Acoustic Environment is specified. This enriches especially the preception of outdoor scenes.

The attributes voxDirectOcclusionControl and voxDiffractionControl specify the direct occlusion and diffracted source attenuation effect modelling strength parameters.

The attribute earlyTuningGainDb controls the level of attenuation of the Early Reflections.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <AcousticTuningParameters> | | | | |
| An <AcousticTuningParamters> node declares global acoustic behaviours. | | | | |
| Attribute | Type | Flags | Default | Description |
| defaultAE | Boolean | O | False | Enables a default Acoustic Environment. |
| voxDirectOcclusionControl | Float | O | 0 | Direct occlusion strength parameter (range 0..100) |
| voxDiffractionControl | Float | O | 1 | Diffracted source attenuation effect modelling strength parameter (range 0..1) |
| earlyTuningGainDb | Float | O | -8 | Early reflection tuning gain, valid range is -21.5 to +10 dB |

# Property types

## Boolean

|  |  |
| --- | --- |
| Value | Description |
| true | True / yes / on / enabled |
| false | False / no / off / disabled |

## Float

Floating point values are expressed in decimal notation (e.g. “-124.23543”).

## Identifier (ID)

Entities are identified by an ID, which as a unique string. In XML, an entities’ ID is specified by the id attribute. IDs can be used to reference other entities.

**Example:**

id=”piano”

## File path

File system paths are relative to the location of the scene.xml file.  
In order to ensure platform independence, slashes (/) are used exclusively as filesystem separators. On Windows platforms, these are manually translated to backslashes (\).

**Example:**

file=”audio/piano.wav”

## Gain

Gains are expressed in decibels (dB) as float numbers. Decibels refer to the effective value of the signal amplitude, i.e. . 0 dB indicates no gain change relative to the WAV file signal level.

**Example:**

gainDb=”-6.0” denotes a gain of factor

## Coordinate spaces

The specification supports different *frames of reference* in which positions and orientations are expressed.

By default, an element’s position and orientation are expressed in its parent entities’ coordinate space.

The global cartesian coordinate space is defined by the root entity, that is the audio scene itself. Any other entity with a position and an orientation defines its own local coordinate space. The user inside a VR/AR environment also defines his or her own frame of reference, the user coordinate space. The cspace attribute defines for an entity in which its position and orientation is specified.

|  |  |
| --- | --- |
| Value | Description |
| relative | Expresses the position/orientation relative to the local coordinates of the parent spatial entity (i.e. an entity that has a position/orientation). Two cases can occur:   * When the entity is a direct child node of the <AudioScene>, its position/orientation refer to the *global* coordinate system. * When the entity is a child node of a <Transform>, its position/orientation are *relative* to the transforms frame of reference. |
| user | Expresses the position/orientation already relative to the user. |

## Position

Spatial positions are expressed by three spatial coordinates referring to a cartesian coordinate system. In XML, a position is specified by a position attribute, followed by a string with three numbers separated by whitespaces. The unit of coordinates is meters.

**Example:**

position=”1.2 -0.4 2.3” denotes the point at x=1.2, y=-0.4, z=2.3

## Voxel grid positions

Spatial voxel positions are expressed by three spatial indices referring to a regular grid associated with the scene coordinate system. In XML, voxel positions are specified by grid\_coordinate\_s and grid\_coordinate\_e attributes, followed by a string with three positive integer numbers (range [1, vox\_resolution] per grid dimension) separated by whitespaces. The unit of coordinates is grid index.

**Example:**

grid\_coordinate\_s=”1 2 3” denotes the point at grid position ix=1, iy=2, iz=3.

## Dimensions

Spatial dimensions are expressed by three sizes (x, y, z) referring to the entity’s local coordinate system. In XML, the dimensions are specified by a size attribute, followed by a string with three numbers separated by whitespaces. Each value must be a non-negative number. The unit of sizes is meters.

**Example:**

size=”1 0 2.1” denotes spatial dimensions of x=1, y=0 and z=2.1 meters.

## Voxel resolution

Spatial voxel resolution is expressed by three values (Nx, Ny, Nz) referring to the number of voxels per each scene dimension associated with the scene coordinate system. In XML, the voxel resolution specified by a vox\_resolution attribute, followed by a string with three integer numbers separated by whitespaces. Each value must be a positive number. The unit of sizes is number of voxels.

**Example:**

vox\_resolution=”128 64 32” denotes voxel scene resolution of 128 ≥ ix ≥ 1, 64 ≥ iy ≥ 1 and 32 ≥ iz ≥ 1.

## Rotation

The spatial orientation of an entity is expressed by three axis rotations in *yaw-pitch-roll* convention (see Fig. 15). This common convention is used by the majority of 3D modeling programs (here sometimes called *heading-pitch-bank*). A definition of these rotations is provided in annex A.2. In XML, a rotation is described by a rotation attribute, followed by a string with three numbers separated by whitespaces. The unit of angles is degrees.

**Example:**

rotation=”45 -20 5” corresponds to 45° yaw, -20° pitch and 5° roll.

## CICP layout

Loudspeaker layout indicated by a CICP ChannelConfiguration as specified in ISO/IEC 23091-3:2018 CICP-Audio. The layout is described by an CICP loudspeaker layout index.

## Authoring parameters

The auralization of Audio Elements can be controlled using authoring parameters, representing the artistic intent. The following are available:

|  |  |
| --- | --- |
| Value | Description |
| noReverb | Omits reverb simulation (e.g. when source signal is reverberant) |
| noDoppler | Omit the simulation of Doppler shifts (e.g. when these are already contained in the signal) |
| noDistance | Omit distance attenuation (e.g. when it is already contained in the signal) |
| noRescale | Omit audio source rescaling (e.q. when the element’s position should not change w.r.t. an anchor object) |

The parameters are specified in XML by an aparams attribute, whose value is a string with the list of the desired parameters. An empty string identifies no restrictions (default parameters).

**Example:** aparams=”noDoppler,noDistance”

Notes:

* Direct signal output, without rendering and spatialization, is not considered  
  in the evaluation at the moment.

# Conclusions

This document specifies the encoder input format for the MPEG-I Audio 6DoF metadata encoder.

A general scene meta model is introduced, defining static declarations and dynamic updates. Many essential entity and property types are defined, documented and illustrated by the help of examples.

# References

1. N0057 *MPEG-I Immersive Audio Draft Documentation for the Audio Evaluation Platform*
2. N0028 *MPEG-I Immersive Audio Architecture and Requirements*
3. Spatially oriented format for acoustics (SOFA), <https://www.sofaconventions.org>
4. N0059 *DRAFT MPEG-I Immersive Audio CfP Supplemental Information and Clarifications*

# Annex A

A.1 Cartesian coordinate system

MPEG-I uses the OpenGL cartesian coordinate system (see Fig. 13),

which is right-handed and +y points up.

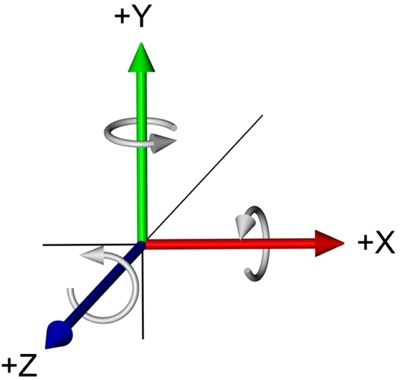


Figure 13: OpenGL coordinate system  
(right-handed, +y points up)

A.2 Spherical coordinate system

Spherical coordinates are defined by a distance , azimuth angle and elevation angle in the following system:

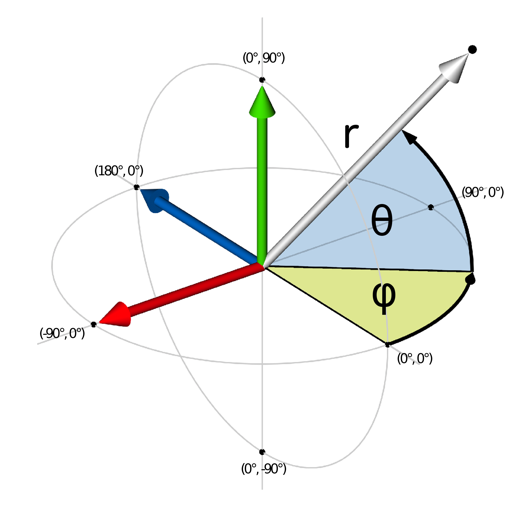


Figure 14: Spherical coordinate system

Positive azimuth angles turn counterclockwise (CCW) with respect to the up axis (y+).  
Points in the upper hemisphere (+y) have positive elevation angles. Radii are always positive.

**Examples:**

* (1, 0°, 0°) spherical matches (0, 0, -1) cartesian
* (1, 90°, 0°) spherical matches (-1, 0, 0) cartesian
* (1, 0°, 90°) spherical matches (0, 1, 0) cartesian

A.3 Yaw-pitch-roll rotations

*Yaw-pitch-roll* rotations are specific convention of Euler angles (i.e. rotations are three axis). Fig. 15 shows the three different axis rotations. *Yaw-pitch-roll* rotations are performed intrinsic, i.e. each rotation is relative to the previously rotated axis. This is achieved by executing the (extrinsic) rotations around the global coordinate axis in the following order

1. roll = rotation around the **global +z axis**
2. pitch = rotation around the **global +x axis**
3. yaw = rotation around the **global +y axis**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1. yaw rotation | 1. pitch rotation | (c) roll rotation |

Figure 15: Yaw, pitch and roll rotations

Written in rotation matrices , , (see [8]) for the three global coordinate axes the total   
yaw-pitch-roll rotation matrix yields: .

Note, that the order of matrix products read right to left.

An object with yaw-pitch-roll angles (0°, 0°, 0°) has the following orientation in the global cartesian coordinate axes:

* Its front direction points into -z
* Its up direction points to +y
* Its right direction points to +x

A.4 Determining of regions by geometric objects

Using geometric objects to specify spatial regions demands a clear semantic whether a point lies inside or outside. For primitives (cube, cylinder, sphere) this is trivial. For most meshes this can be tested as follows (informal mathematical definition):

A point lies inside a given mesh M, when a ray () shot from  
 in an arbitrary direction intersects an odd number of M’s faces.

Different test cases are illustrated in Fig. 16. The point in (a) lies in the mesh as one intersection occurs. In (b) it lies outside, as two intersections occur. Note, that when no intersections occur as in (c), the mesh is *incomplete* and unsuited for defining regions. The content creator must ensure that meshes used for this purpose are complete, i.e. do not have undefined openings.

|  |  |  |
| --- | --- | --- |
| (a) Point inside mesh | (b) Point outside mesh | (c) Incomplete mesh |
| Figure 16: Ray-mesh intersection tests | | |