ETSI TS 103 448 V1.1.1 (2016-09)

TECHNICAL SPECIFICATION

AC-4 Object Audio Renderer for Consumer Use

EBU

OPERATING EUROVISION

Reference
DTS/JTC-038

Keywords
audio, broadcasting, digital

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

The present document can be downloaded from:
http://www.etsi.org/standards-search

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the only prevailing document is the print of the Portable Document Format (PDF) version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at
https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx

If you find errors in the present document, please send your comment to one of the following services:
https://portal.etsi.org/People/CommiteeSupportStaff.aspx

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by Joint Technical Committee (JTC) Broadcast of the European Broadcasting Union (EBU), Comité Européen de Normalisation ELECtrotechnique (CENELEC) and the European Telecommunications Standards Institute (ETSI).

NOTE: The EBU/ETSI JTC Broadcast was established in 1990 to co-ordinate the drafting of standards in the specific field of broadcasting and related fields. Since 1995 the JTC Broadcast became a tripartite body by including in the Memorandum of Understanding also CENELEC, which is responsible for the standardization of radio and television receivers. The EBU is a professional association of broadcasting organizations whose work includes the co-ordination of its members' activities in the technical, legal, programme-making and programme-exchange domains. The EBU has active members in about 60 countries in the European broadcasting area; its headquarters is in Geneva.

European Broadcasting Union
CH-1218 GRAND SACONNEX (Geneva)
Switzerland
Tel: +41 22 717 21 11
Fax: +41 22 717 24 81

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Introduction

Motivation

Current industry trends for authoring and reproduction of audio content include immersive audio and support for personalization of the audio, as well as many different speaker setups and layouts.

Different means of immersive audio and personalization are provided in ETSI TS 103 190-2 [1]. Object-based audio is one of the means for supporting these trends.

Objects can be thought of as the input tracks to a mixing console, the mixing console being the renderer. But objects are more than audio tracks. They carry metadata that is authored with the tracks. Contemporary mixing consoles have automated gains. For a renderer accepting object-based audio, those gains are driven by the object's own metadata. Metadata is also used to define object location and size, as well as many other ancillary parameters that control the object presentation.

The final mix output by the contemporary mixing console is targeted at a specific playback system. Other channel configurations can be derived from the mix, but they are not necessarily what is monitored. A renderer, located in a playback device in a consumer's home, acts as the mixing console for that device, with the advantage that the speaker setup is known to the renderer. The renderer can use the location and size metadata defined for each object to produce the playback experience that best matches the content creator's intention, within the possibilities and constraints of the available speaker setup.

The present document specifies an object audio renderer for use with ETSI TS 103 190-2 [1], using the metadata as specified therein.

Structure of the document

The present document is structured as follows.

- Clause 4 specifies the input and output interfaces, and the architecture of the renderer.
- Clause 5 specifies the processing blocks of the renderer. These are:
  - Metadata preprocessing, specified in clause 5.1
  - Source panners, specified in clause 5.2
  - Trim processing, specified in clause 5.3
  - Gain mixing, specified in clause 5.4
  - Ramp mixing, specified in clause 5.5
- Annex A lists the supported loudspeaker configurations and associated parameters that are utilized by the processing blocks of the renderer.

An overview of the incoming metadata and the result of the rendering process is presented in clause 4, which makes it a proper starting point when reading the document.

1 Scope

The present document defines an extension to the AC-4 codec.

The present document specifies a consumer object-based audio renderer for use with the AC-4 codec as specified in ETSI TS 103 190-2 [1], and the object-based audio metadata specified therein. The renderer takes the object audio essence and the corresponding metadata defined in ETSI TS 103 190-2 [1] as inputs, and produces loudspeaker feeds for consumer loudspeaker layouts.

2 References

2.1 Normative References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

Referenced documents that are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: Although any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI TS 103 190-2: "Digital Audio Compression (AC-4) Standard; Part 2: Immersive and personalized audio".

[2] Recommendation ITU-R BS.2051-0: "Advanced sound system for programme production".

2.2 Informative References

References are either specific (identified by date of publication and/or edition number or version number) or nonspecific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document, but they assist the user with regard to a particular subject area.

Not applicable.

3 Definitions, symbols, abbreviations and conventions

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

bitstream: sequence of bits

channel: audio signal intended for playback by one of a set of dedicated loudspeakers with predetermined locations, e.g. Left, Right, and Centre channels

codec: system that consists of an encoder and a decoder

divergence: panning mechanism including a control to balance between rendering the object as a point source and panning the object across a specified horizontal distance

gain: multiplicative factor applied to a signal

immersive audio: multi-channel audio for playback with loudspeaker layouts in more than one plane

EXAMPLE: 3/4/4 or 3/2/2

low-frequency effects: band-limited channel specifically intended for deep, low-pitched sounds

loudspeaker feed: audio signal that the renderer has determined to be played back by a certain loudspeaker

metadata: data about data

object: object audio essence plus associated object-based audio metadata

object audio essence: part of the object that is PCM coded

object-based audio: audio content composed of objects

panner: device or an algorithm that performs panning

panning: distribution of a sound signal into a stereo or multi-channel loudspeaker layout

point source: single localized source of audio with negligible size

(object-based audio) rendering: processing of audio content to adapt it to a specific loudspeaker layout

screen-anchored coordinates: coordinates that specify the position of an object in relation to the size and location of the screen

snap: relocation of an object to minimize the audible result of panning

speaker-anchored coordinates: coordinates that specify the position of an object by associating it to a loudspeaker

surround sound: multi-channel audio content for playback with loudspeaker layouts in a single plane

EXAMPLE: 3/2/0 or 3/4/0

trim: process of signal attenuation to adapt the audio to play back on a loudspeaker layout with fewer loudspeakers than the mastering loudspeaker layout

zone: sub-volume of the listening room

3.2 Symbols

For the purposes of the present document, the following symbols apply:

$\{\mathbf{a}; \mathbf{b}\}$	a list of individual values a and b
$[\mathbf{a}; \mathbf{b}]$	a closed interval between a and b values
$(\mathbf{x}; \mathbf{y}; \mathbf{z})$	a three-dimensional vector, used for specifying a position inside the room
$\lvert\mathbf{a}\rvert$	the absolute value of a

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AC	Audio Codec
LFE	Low-Frequency Effects
PCM	Pulse Code Modulation

3.4 Conventions

Unless otherwise stated, the following conventions are used in the present document.

Typographic convention:

Italic font denotes variables and metadata items (*n* is a variable or a metadata item).

Function prototypes can take scalars, vectors, or matrices as arguments and operate element-wise. The return type is either scalar or vector of the same format as the argument.

abs(x) The absolute value of the elements of x

clamp(x) The clamp function is defined as follows: $\mathrm{clamp}(x) = \begin{cases} 0, & when \quad x < 0 \\ x, & \mathrm{when} \quad x \in [0, 1] \\ 1, & when \quad x > 1 \end{cases}$

db_to_linear(x) The conversion of values of the elements of x from logarithmic to linear scale, defined as follows: $\mathrm{db\_to\_linear}(x) = 10^{(x/_{20})}$

floor(x) The largest integer(s) less than or equal to the elements of x

isempty(x) Returns true if the vector x is empty, false otherwise

max(x) The maximum value of the elements of x

min(x) The minimum value of the elements of x

mod(x, y) The mod function denotes the remainder of x after division by y

pow(x, y) The pow function denotes the power function, where x is a base and y is an exponent

sign(x) The sign function is defined as follows: $\mathrm{sign}(x) = \begin{cases} -1, & when \quad x < 0 \\ 0, & \mathrm{when} \quad x = 0 \\ 1, & when \quad x > 1 \end{cases}$

sort_descending(x) The elements of x sorted in a descending order of value

NOTE 1: The return value of mod() can have a fractional part.

Pseudocode conventions:

y(x) function y with parameter x

y[x] array y with an array index x

{x : C(x)} Select elements of x for which condition C(x) is met

C ? X : Y Check if condition C is met. If true, process the X statement, otherwise, process the Y statement

NOTE 2: In pseudocode, dot is used as a decimal point.

4 System overview

4.1 Architecture

4.1.1 Introduction

The rendering process consists of the following stages:

1) Metadata pre-processing.
2) Source panning.
3) Trimming.
4) Gain mixing.
5) Ramp mixing.

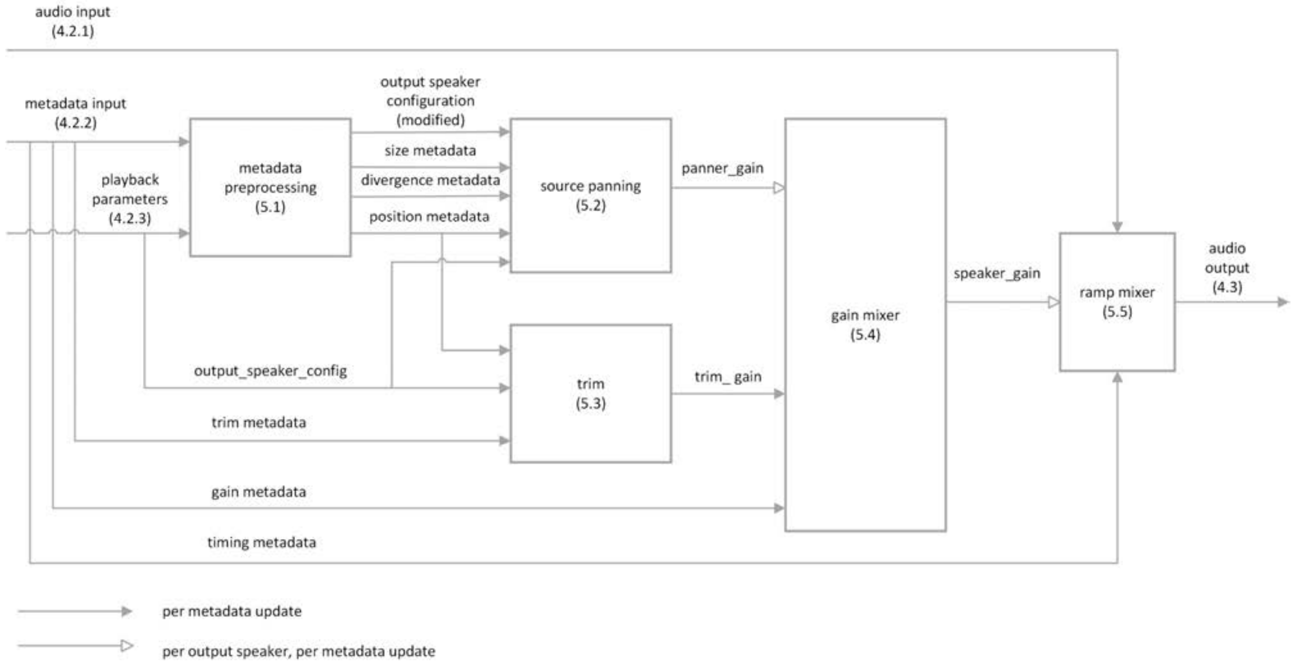

Figure 1

4.1.2 Requirements

The renderer signal processing order shall be as specified in table 1.

Table 1: Renderer signal processing

Order	Processing block	Inputs	Outputs
1	Metadata preprocessing (clause 5.1)	Metadata input (clause 4.2.2) Playback parameters (clause 4.2.3)	*Output loudspeaker configuration (modified)* *object_width_X* *object_width_Y* *object_width_Z* *object_divergence* *object_position_X* *object_position_Y* *object_position_Z*
2	Source panning (clause 5.2)	*Output loudspeaker configuration (modified)* *object_width_X* *object_width_Y* *object_width_Z* *object_divergence* *object_position_X* *object_position_Y* *object_position_Z* *channel*	*panner_gain*
3	Trim (clause 5.3)	*object_position_X* *object_position_Y* *object_position_Z* *output_speaker_config* *trim_mode* *trim_height* *trim_surround* *trim_centre*	*trim_gain*
4	Gain mixer (clause 5.4)	*panner_gain* *trim_gain* *object_gain*	*speaker_gain*
5	Ramp mixer (clause 5.5)	Audio input (clause 4.2.1) *Timing metadata* *speaker_gain*	Audio output (clause 4.3)

4.2 Input

4.2.1 Audio

The audio interface accepts an object audio essence to be rendered to the output loudspeaker layout.

The renderer shall provide an interface for one object audio essence with a length of *frame_size* samples.

4.2.2 Metadata

Metadata listed in this clause are transmitted to the renderer in the AC-4 bitstream. The metadata interface accepts timing metadata and *num_obj_info_blocks* timed updates of the object metadata.

The renderer shall provide an interface for the metadata as specified in table 2.

Table 2: Input metadata

Metadata	Range	Type	Metadata specified in ETSI TS 103 190-2 [1]	Metadata group
num_obj_info_blocks	$[0;7]$	integer	Clause 6.3.9.3	timing metadata
ramp_duration [num_obj_info_blocks]	$[0;2047]$	integer	Clause 6.3.9.3	timing metadata
block_offset_factor [num_obj_info_blocks]	$[0;63]$	integer	Clause 6.3.9.3	timing metadata
sample_offset [num_obj_info_blocks]	$[0;31]$	integer	Clause 6.3.9.3	timing metadata
object_position_X [num_obj_info_blocks]	$[0;1]$	float	Clause 6.3.9.8.4	position metadata
object_position_Y [num_obj_info_blocks]	$[0;1]$	float	Clause 6.3.9.8.4	position metadata
object_position_Z [num_obj_info_blocks]	$[-1;1]$	float	Clause 6.3.9.8.4	position metadata
channel	{*'L'*; *'R'*; *'C'*; *'Ls'*; *'Rs'*; *'Lb'*; *'Rb'*; *'Tfl'*; *'Tfr'*; *'Tl'*; *'Tr'*; *'Tbl'*; *'Tbr'*; *'Lw'*; *'Rw'*; *'LFE'*; *'LFE2'*}	string	Clause 6.3.2.9.5, Clause 6.3.2.9.8, Clause 6.3.2.9.9	position metadata
object_width [num_obj_info_blocks]	$[0;1]$	float	Clause 6.3.9.8.13	size metadata
object_width_X [num_obj_info_blocks]	$[0;1]$	float	Clause 6.3.9.8.14	size metadata
object_width_Y [num_obj_info_blocks]	$[0;1]$	float	Clause 6.3.9.8.15	size metadata
object_width_Z [num_obj_info_blocks]	$[0;1]$	float	Clause 6.3.9.8.16	size metadata
object_divergence [num_obj_info_blocks]	$[0;1]$	float	Clause 6.3.9.8.21	divergence metadata
b_object_snap [num_obj_info_blocks]	{true; false}	boolean	Clause 6.3.9.8.9	snap metadata
zone_mask [num_obj_info_blocks]	$[0;7]$	integer	Clause 6.3.9.8.7	zone metadata
b_enable_elevation [num_obj_info_blocks]	{true; false}	boolean	Clause 6.3.9.8.8	zone metadata
object_gain [num_obj_info_blocks]	$-\infty$ or $[-49;15]$	integer	Clause 6.3.9.7	gain metadata
b_object_not_active [num_obj_info_blocks]	$\{\text{true};\text{false}\}$	boolean	Clause 6.3.9.6.2	gain metadata
object_priority [num_obj_info_blocks]	$[0;1]$	float	Clause 6.3.9.7.6	future use-case metadata
b_ducking_disabled	$\{\text{true};\text{false}\}$	boolean	Clause 6.3.9.4.2	future use-case metadata
object_sound_category	$\{0;1\}$	integer	Clause 6.3.9.4.3	future use-case metadata
master_screen_size_ratio [num_obj_info_blocks]	$[0;1]$	float	Clause 6.3.9.2.3	future use-case metadata
trim_mode [9] [num_obj_info_blocks]	{disable_trim; default_trim; custom_trim}	ternary	Clause 6.3.3.1.18	trim metadata
trim_height [9] [num_obj_info_blocks]	$[-36;0]$	float	Clause 6.3.3.1.18	trim metadata
trim_surround [9] [num_obj_info_blocks]	$[-36;0]$	float	Clause 6.3.3.1.18	trim metadata
trim_centre [9] [num_obj_info_blocks]	$[-36;6]$	float	Clause 6.3.3.1.18	trim metadata
object_screen_factor [num_obj_info_blocks]	$[0;1]$	float	Clause 6.3.9.8.17	screen-anchored position metadata
y_position_exponent [num_obj_info_blocks]	$[0;2]$	float	Clause 6.3.9.8.18	Screen-anchored position metadata

4.2.3 Playback parameters

The playback parameters listed in this clause are set by the playback device to configure the rendering process.

The renderer shall provide an interface for the playback parameters as listed in table 3.

Table 3: Playback parameters

Parameter	Range	Type	Description
output_speaker_config	$[0;8]$	integer	set output loudspeaker configuration as specified in table A.1
target_screen_origin_X	$[0;1]$	float	specified in clause 5.1.2
target_screen_origin_Z	$[-1;1]$	float	specified in clause 5.1.2
target_screen_width_ratio	$[0;1]$	float	specified in clause 5.1.2
target_screen_height_ratio	$[0;1]$	float	specified in clause 5.1.2

4.3 Output

The audio interface provides a loudspeaker feed audio signal for each loudspeaker of the output loudspeaker configuration.

The renderer shall provide an interface for *num_speakers* loudspeaker feeds, where each loudspeaker feed is a sequence of PCM audio samples with a length of *frame_size* samples.

NOTE: The *num_speakers* value depends on the output loudspeaker configuration as specified in table A.1.

5 Functional overview

5.1 Metadata pre-processing

5.1.1 Processing order

5.1.1.1 Introduction

To create a deterministic rendering result, the incoming metadata is pre-processed in a specific order. The output of a metadata pre-processor block may be directed to the input of the following block. Figure 2 shows inputs and outputs of each metadata pre-processor block.

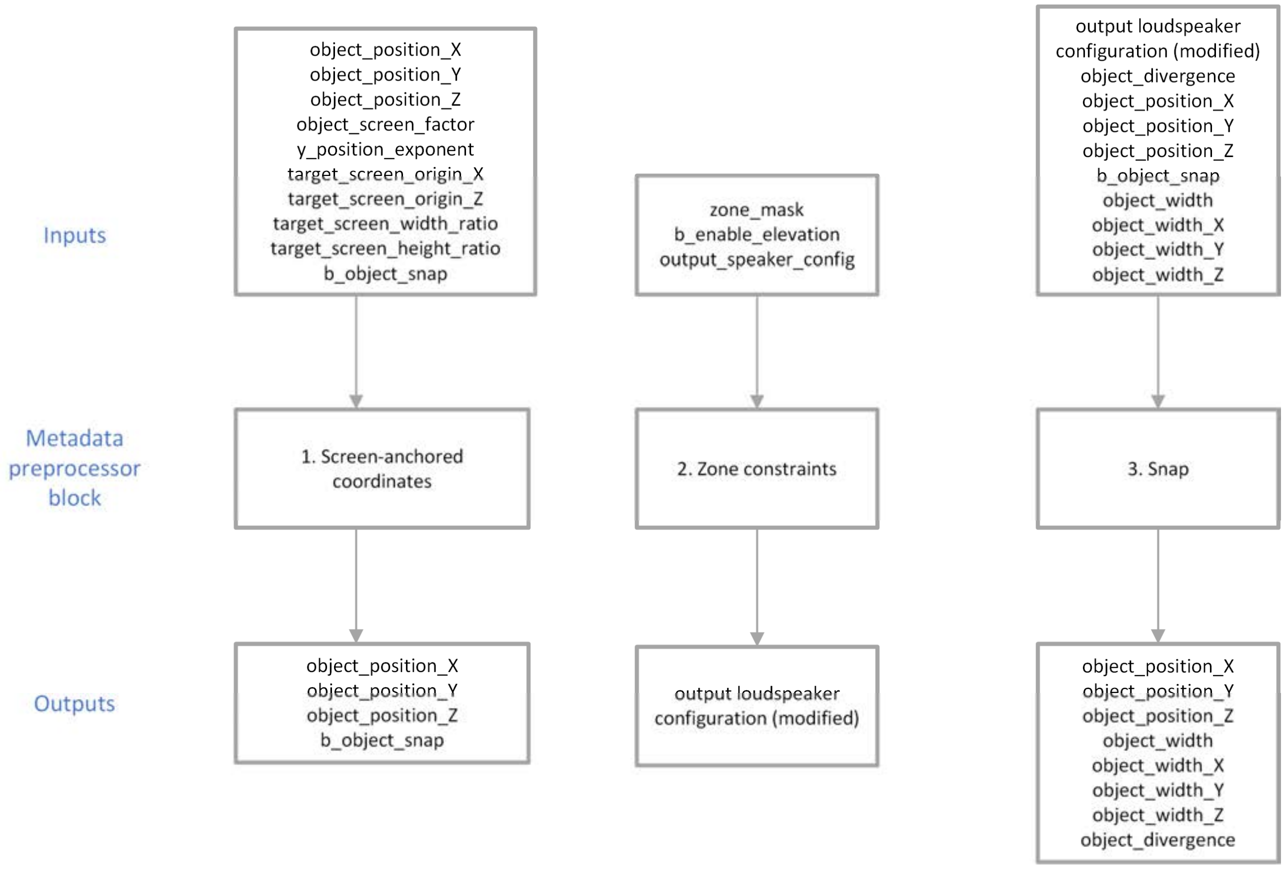

Figure 2

5.1.1.2 Requirements

The metadata shall be pre-processed in the following order:

1) Screen-anchored coordinates
2) Zone constraints
3) Snap

5.1.2 Screen-anchored coordinates

5.1.2.1 Introduction

In consumer playback environments, the position of the Left and Right loudspeakers can vary greatly with respect to the edges of the playback screen. At one extreme, the distance between the Left and Right loudspeakers can be much wider than the screen; at the other extreme, the screen can be very large, such that Left and Right loudspeakers are adjacent to the screen. As a result, audio and visual events that are co-located on the edge of the screen during content creation may not be aligned on different consumer screens.

To address the above issue, content creators can classify the three dimensional coordinates as screen-anchored coordinates, by adding the *object_screen_factor* and *y_position_exponent* metadata items. If these two parameters are available at the metadata interface, the renderer recognizes the relation of the available position coordinates to the screen, and should follow the recommendations specified in clause 5.1.2.2. These algorithms allow the conversion of screen-related position coordinates to position coordinates that can be processed by the panning algorithms specified in the present document.

In addition to the *object_screen_factor* and *y_position_exponent* metadata, information about the playback environment is needed to appropriately implement the processing of screen-anchored coordinates. This information should be computed from the physical location of the screen and the loudspeakers in the playback room, as specified in clause 5.1.2.2.

5.1.2.2 Recommendations

The renderer assumes that the following conditions are met:

- The screen lies entirely between the Left and Right loudspeakers.
- For all loudspeaker configurations in table A.1 with *num_top* > 0, the screen lies entirely between the ceiling and floor loudspeakers (or the floor, if no floor loudspeakers are present).

These conditions can be stated numerically as follows:

- $|\text{target\_screen\_origin\_X} - 0.5| \times 2 + \text{target\_screen\_width\_ratio} \leq 1$
- $|\text{target\_screen\_origin\_Z}| + \text{target\_screen\_height\_ratio} \leq 1$

Let L_x denote the horizontal position of the Left loudspeaker.

Let R_x denote the horizontal position of the Right loudspeaker.

Let S_{cx} denote the horizontal position of the screen centre.

Let S_w denote the horizontal width of the screen.

Let S_{cz} denote the vertical position of the screen centre.

Let S_{tz} denote the vertical position of the top edge of the screen.

Let LR_z denote the vertical position of the Left and the Right loudspeakers.

Let T_z denote the vertical position of the top loudspeaker(s) with minimum *panner_target_Y* value.

The playback device should calculate $\text{target\_screen\_width\_ratio} = \frac{S_w}{R_x - L_x}$.

The playback device should calculate $\text{target\_screen\_origin\_X} = \frac{S_{cx} - L_x}{R_x - L_x}$.

The playback device should calculate $\text{target\_screen\_height\_ratio} = \frac{S_{tz} - S_{cz}}{T_z - LR_z}$.

The playback device should calculate $\text{target\_screen\_origin\_Z} = \frac{S_{cz} - LR_z}{T_z - LR_z}$.

If the output loudspeaker layout does not contain top loudspeakers, i.e. the value of *num_top* specified in table A.1 is 0, the renderer should set:

- *target_screen_height_ratio* = 1
- *target_screen_origin_Z* = 0

If *object_screen_factor[mdu]* and *y_position_exponent[mdu]* are present at the metadata interface, as specified in table 2, the renderer should:

- Set *b_object_snap[mdu]* to false.
- For each *mdu*, recalculate (object_position_X[mdu]; object_position_Y[mdu]; object_position_Z[mdu]), as specified in clause 5.1.2.3.

5.1.2.3 Algorithmic details

Pseudocode 1

```
(object_position_X; object_position_Y; object_position_Z) = screen_scale (object_position_X
                                                                    ,object_position_Y
                                                                    ,object_position_Z
                                                                    ,object_screen_factor
                                                                    ,y_position_exponent
                                                                    ,target_screen_width_ratio
                                                                    ,target_screen_origin_X
                                                                    ,target_screen_height_ratio
                                                                    ,target_screen_origin_Z)
{
  if (object_screen_factor > 0)
  {
    /* X-position on target screen, in room-anchored coordinates */
    x_target_screen = (object_position_X - 0.5) * target_screen_width_ratio
                                                + target_screen_origin_X;

    /* interpolate according object_screen_factor */
    x_target = (object_screen_factor * x_target_screen) + (1 - object_screen_factor) * object_positi
on_X;

    /* interpolate according to y position and y_position_exponent */
    object_position_X = pow(object_position_Y, y_position_exponent) * object_position_X
                      + (1 - pow(object_position_Y, y_position_exponent)) * x_target;

    /* z scaling */
    z_target_screen = object_position_Z * target_screen_height_ratio + target_screen_origin_Z;
    z_target = (object_screen_factor * z_target_screen)
             + (1 - object_screen_factor) * object_position_Z;
    object_position_Z = pow(object_position_Y, y_position_exponent) * object_position_Z
                      + (1 - pow(object_position_Y, y_position_exponent)) * z_target;
  }
}
```

5.1.3 Zone constraints

5.1.3.1 Introduction

Zone constraints provide the means of specifying a spatial region to be excluded from rendering an object. They are indicated by the *zone_mask* and *b_enable_elevation* metadata items.

Zone constraints can reduce the number of available panner targets for the supported loudspeaker configurations that are listed in table A.1. To achieve this, the output loudspeaker configuration and its panner targets are modified by the zone constraints processing, before being used by the source panner.

5.1.3.2 Requirements

The renderer shall modify the output loudspeaker configuration that is sent to the source panner by performing the following steps:

- For the combination of *output_speaker_config* and *zone_mask* value, as listed in table 4, exclude loudspeakers listed in the third column and reduce the *num_speakers* value as listed in the fourth column.
- If *b_enable_elevation* is false, for the *output_speaker_config* indexes listed in table 5, exclude loudspeakers listed in the second column, reduce the *num_speakers* value as listed in the third column, and set *object_position_Z* to 0.

NOTE: The requirements listed in this clause apply to each metadata update *mdu* in [0; *num_object_info_blocks*-1].

Table 4: Loudspeaker exclusions indicated by *zone_mask*

***output_speaker_config* index as specified in table A.1**	***zone_mask* value**	**Loudspeakers to be excluded**	**Reduce *num_speakers* by**
1; 5; 6	3	Left Right	2
1; 5; 6	4	Left surround Right surround	2
1; 5; 6	5	Left Right Centre	3
2; 3; 4; 8	1	Left back Right back	2
2; 3; 4; 8	2	Left side Right side	2
2; 3; 4; 8	3	Left Right Left side Right side	4
2; 3; 4; 8	4	Left side Right side Left back Right back	4
2; 3; 4; 8	5	Left Right Centre	3
7	1	Left back Right back Back centre	3
7	2	Left side Right side Left wide Right wide	4
7	3	Left Right Left wide Right wide Left side Right side	6
7	4	Left side Right side Left wide Right wide Left back Right back Back centre	7
7	5	Left Right Centre Left wide Right wide	5

Table 5: Loudspeaker exclusions indicated by *b_enable_elevation*

***output_speaker_config* index as specified in table A.1**	**Loudspeakers to be excluded**	**Reduce *num_speakers* by**
3; 5	Top front left Top front right Top back left Top back right	4
4; 6	Top left Top right	2
7	Top front left Top front right Top back left Top back right Top side left Top side right Top front centre Top back centre Top centre Bottom front left Bottom front right Bottom front centre	12
8	Top front left Top front right Top back centre	3

5.1.4 Snap

5.1.4.1 Introduction

The snap metadata indicates whether the position of the object is modified ("snapped") and assigned to the closest among a fixed set of locations. This set of locations is intended to coincide with the *panner_target* locations for the largest loudspeaker configuration supported by the renderer.

A weighted Euclidean distance metric is used to determine which location to snap the object to. This distance metric creates rectangular cuboid "snap" regions around each location in Cartesian space. Dividing the "snap" regions in this way improves the intuitive use of the snap feature during content creation.

5.1.4.2 Requirements

The renderer shall create a list of snap target locations, corresponding to the panner target coordinates in table A.9:

- $\mathrm{snap\_target\_X}[k] = \mathrm{panner\_target\_X}[k]$;
- $\mathrm{snap\_target\_Y}[k] = \mathrm{panner\_target\_Y}[k]$; and
- $\mathrm{snap\_target\_Z}[k] = \mathrm{panner\_target\_Z}[k]$;
- where $k \in [0; \mathrm{num\_snap\_targets} - 1]$.

The *num_snap_targets* value shall be equal to the *num_speakers* value for the 22.2 loudspeaker configuration, as listed in table A.1.

The list of snap target locations and the value of *num_snap_targets* shall then be modified according to *zone_mask* and *b_enable_elevation*, as specified in clause 5.1.3 for the 22.2 loudspeaker configuration.

The renderer shall re-order the list of snap target locations based on their coordinates, in the following sequence:

1) Order locations with coordinates (0,5; 0; 0), (0,5; 0; 1), and (0,5; 0; -1) as listed.
2) Sort locations from highest to lowest *snap_target_Z* value.
3) Sort locations with equal *snap_target_Z* value from highest to lowest *snap_target_Y* value.

4) Sort locations with equal *snap_target_Z* and *snap_target_Y* value from lowest to highest *snap_target_X* value.

If *b_object_snap* is true, the renderer shall recalculate the coordinates (*object_position_X*; *object_position_Y*; *object_position_Z*), as specified in Pseudocode 2.

If *b_object_snap* is true, the renderer shall set object_width_X = object_width_Y = object_width_Z = 0

If *b_object_snap* is true, the renderer shall set object_divergence = 0

NOTE: The requirements listed in this clause apply to each metadata update *mdu* in [0; *num_object_info_blocks*-1].

5.1.4.3 Algorithmic details

Pseudocode 2

```
min_dist = Inf;
weight_X = 1/16; weight_Y = 4; weight_Z = 32;

/* find the closest loudspeaker */
for (j = 0; j < num_speaker; j++) /* for each loudspeaker */
{
  /* weighted Euclidean distance using Cartesian object
   * and loudspeaker positions*/
  dist = weight_X * pow((object_position_X - snap_target_X(j)), 2)
       + weight_Y * pow((object_position_Y - snap_target_Y(j)), 2)
       + weight_Z * pow((object_position_Z - snap_target_Z(j)), 2);

  if (dist < min_dist)
  {
    min_dist = dist;
    idx_min = j;
  }
}

object_position_X = snap_target_X(idx_min);
object_position_Y = snap_target_Y(idx_min);
object_position_Z = snap_target_Z(idx_min);
```

5.1.5 Metadata for future use cases

Some of the metadata specified at the metadata interface is reserved for future use cases.

The renderer shall ignore *object_priority*.

The renderer should ignore *b_ducking_disabled*.

The renderer should ignore *object_sound_category*.

The renderer should ignore *master_screen_size_ratio*.

5.2 Source panner

5.2.1 Source panner architecture

5.2.1.1 Introduction

The source panner consists of the following parallel blocks utilizing different panning algorithms:

1) Speaker-anchored coordinates panner.

2) Divergence panner.

3) Size panner.

4) Point source panner.

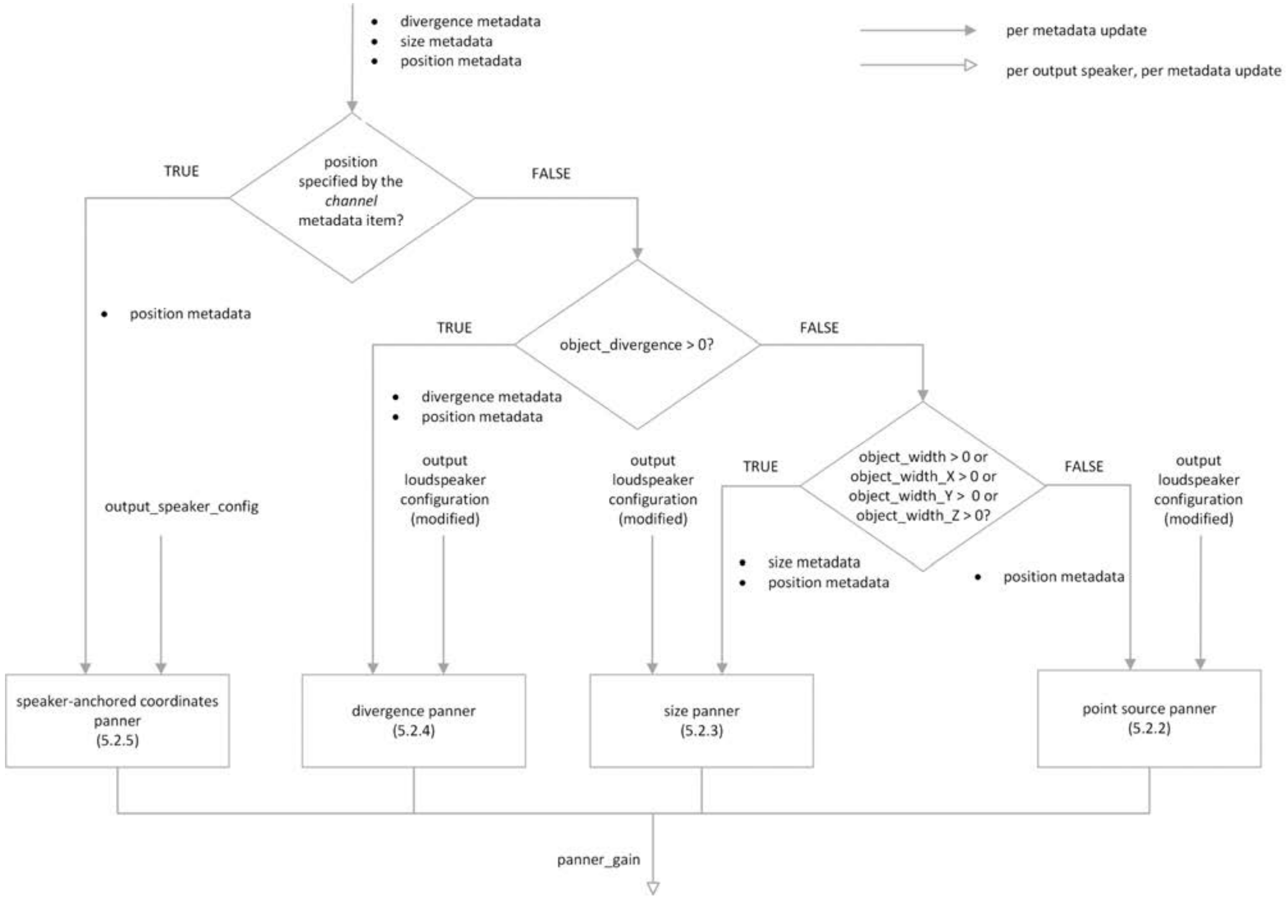

Figure 3

5.2.1.2 Requirements

To choose a panning algorithm, the conditions below shall be evaluated in the following order:

5) If the position of an object is specified by *channel*, use the speaker-anchored coordinates panner, as specified in clause 5.2.5.

6) If object_divergence > 0, use the divergence panner, as specified in clause 5.2.4.

7) If object_width > 0 or object_width_X > 0 or object_width_Y > 0 or object_width_Z > 0, use the size panner, as specified in clause 5.2.3.

8) Otherwise, use the point source panner, as specified in clause 5.2.2.

5.2.2 Rendering point sources

5.2.2.1 Introduction

The point panner can create the impression of an auditory event at any point inside the room. It provides a logical extension to the surround sound production tools used today.

The point panner calculates a gain coefficient for each active loudspeaker in the output loudspeaker layout, given an object position.

5.2.2.2 Requirements

The requirements in this clause apply if

- the object position is specified by *object_position_X*, *object_position_Y* and *object_position_Z*.
- *object_divergence*, *object_width*, *object_width_X*, *object_width_Y* and *object_width_Z* are not present at the metadata interface, or all their values are 0.

Let $\mathrm{mdu} \in [0; \mathrm{num\_obj\_info\_blocks} - 1]$ and $\mathrm{j} \in [0; \mathrm{num\_speakers} - 1]$.

For each metadata update *mdu*, the point source panner shall determine the following lists: *panner_target_X*, *panner_target_Y*, and *panner_target_Z*, as specified in clause A.2.

For each metadata update *mdu*, the point source panner shall calculate the matrix $\mathrm{panner\_gain\_Z[mdu]} = \mathrm{pan\_Z(panner\_target\_Z, object\_position\_Z[mdu])}$, as specified in Pseudocode 4.

For each metadata update *mdu*, the point source panner shall calculate the matrix $\mathrm{panner\_gain\_Y[mdu]} = \mathrm{pan\_Y(panner\_target\_Y, panner\_target\_Z, object\_position\_Y[mdu])}$, as specified in Pseudocode 5.

For each metadata update *mdu*, the point source panner shall calculate the matrix $\mathrm{panner\_gain\_X[mdu]} = \mathrm{pan\_X(panner\_target\_X, panner\_target\_Y, panner\_target\_Z, object\_position\_X[mdu])}$, as specified in Pseudocode 6.

The point source panner shall calculate *panner_gain[mdu][j]*, as specified in Pseudocode 3.

5.2.2.3 Algorithmic details

The source point panning algorithm consists of a 3D extension of the panner concept that is widely used in 5.1- and 7.1-channel surround sound production.

Pseudocode 3

```
for (mdu = 0; mdu < num_obj_info_blocks; mdu++) /* for each metadata update */
{
  for (j = 0; j < num_speakers; j++)
  {
    panner_gain[mdu][j] = panner_gain_X[mdu][j]
                        * panner_gain_Y[mdu][j]
                        * panner_gain_Z[mdu][j];
  }
}
```

Pseudocode 4

```
panner_gain_Z[num_speakers] = pan_Z(panner_target_Z[num_speakers]
                                   ,object_position_Z)
{
  for (j = 0; j < num_speakers; j++)
  {
    /* Z-gain */
    z_this = panner_target_Z[j];

    /* find speakers in other plane, on other side of object */
    if(z_this ≥ object_position_Z)
    {
      z_other = max({panner_target_Z : panner_target_Z < z_this});
    }
    else
    {
      z_other = min({panner_target_Z : panner_target_Z > z_this});
    }

    if(isempty(z_other))
    {
      panner_gain_Z[j] = 1.0;
    }
    else if (sign(z_other - object_position_Z) == sign(z_this - object_position_Z))
    {
      panner_gain_Z[j] = 0.0;
    }
```

```
    else
    {
      panner_gain_Z[j] = cos((z_this - object_position_Z) / (z_other - z_this) * pi / 2);
    }
  }
}
```

Pseudocode 5

```
panner_gain_Y[num_speakers] = pan_Y(panner_target_Y[num_speakers]
                                   ,panner_target_Z[num_speakers]
                                   ,object_position_Y)
{
  epsilon = 0.001; /* small positive constant */
  for (j=0; j<num_speakers; j++)
  {
    y_this = panner_target_Y[j];
    z_this = panner_target_Z[j];

    /* Y-gain */
    /* from among speakers in this plane... */
    panner_target_Y_plane = panner_target_Y[{idx:abs(panner_target_Z(idx)
                          - z_this) < epsilon}];

    /* ...find speakers in closest row, on other side of object */
    if(y_this ≥ object_position_Y)
    {
      y_other = max({panner_target_Y_plane : panner_target_Y_plane < y_this});
    }
    else
    {
      y_other = min({panner_target_Y_plane: panner_target_Y_plane > y_this});
    }

    if (isempty(y_other))
    {
      panner_gain_Y[j] = 1.0;
    }
    else if (sign(y_other - object_position_Y) == sign(y_this - object_position_Y))
    {
      panner_gain_Y[j] = 0.0;
    }
    else
    {
      panner_gain_Y[j] = cos((y_this - object_position_Y) / (y_other - y_this) * pi / 2);
    }
  }
}
```

Pseudocode 6

```
panner_gain_X[num_speakers] = pan_X(panner_target_X[num_speakers]
                                   ,panner_target_Y[num_speakers]
                                   ,panner_target_Z[num_speakers]
                                   ,object_position_X)
{
  epsilon = 0.001; /* small positive constant */

  for (j = 0; j < num_speaker; j++)
  {
    x_this = panner_target_X[j];
    y_this = panner_target_Y[j];
    z_this = panner_target_Z[j];

    /* X-gain */
    /* Among speakers in this plane and row... */
    panner_target_X_plane = panner_target_X[{idx:abs(panner_target_Z(idx)
                          - z_this) < epsilon}];
    panner_target_Y_plane = panner_target_Y[{idx:abs(panner_target_Z(idx)
                          - z_this) < epsilon}];
    panner_target_X_row = panner_target_X_plane[{idx:abs(panner_target_Y_plane(idx)
                        - y_this) < epsilon}];


    /* find closest speaker on other side of the object */
    if(x_this ≥ object_position_X)
```

```
    {
      x_other = max({panner_target_X_row : panner_target_X_row < x_this});
    }
    else
    {
      x_other = min({panner_target_X_row: panner_target_X_row > x_this});
    }

    if(isempty(x_other))
    {
      panner_gain_X[j] = 1.0;
    }
    else if (sign(x_other - object_position_X) == sign(x_this - object_position_X))
    {
      panner_gain_X[j] = 0.0;
    }
    else
    {
      panner_gain_X[j] = cos((x_this - object_position_X) / (x_other - x_this) * pi / 2);
    }
  }
}
```

NOTE 1: A maximum of eight loudspeakers will have non-zero gains.

NOTE 2: The sum of squares of the speaker gains is one, which means that the panning operation is energy preserving.

5.2.3 Rendering objects with size

5.2.3.1 Introduction

Objects with size are rendered by the size panner.

The size panner calculates a gain coefficient for each active loudspeaker in the output loudspeaker layout, given the position and size of an object. The size of an object is in the range [0; 2,8].

To perform the calculation, the size panner considers virtual positions on each of the three room dimension axes, separately. The three lists of one-dimensional positions can have different lengths, i.e. there can be more positions on the X-axis than on the Z-axis.

In general, the size panner processing can be described with the following steps:

1) Calculate point source panner gains per position for each dimension, given the lists of one dimensional positions.

2) Weigh and combine the gains per dimension to produce inside size gains per dimension.

3) Weigh and combine the gains from the one dimensional positions at room boundaries to produce boundary size gains.

4) Combine the inside and boundary size gains to produce the final size gains.

5) Combine the final size gains with the point source panner gains.

5.2.3.2 Requirements

The requirements in this clause apply if

- the object position is specified by *object_position_X*, *object_position_Y* and *object_position_Z*.
- *object_divergence* is not present at the metadata interface or its value is 0.
- one of *object_width*, *object_width_X*, *object_width_Y* or *object_width_Z* is greater than 0.

If *object_width* is present at the metadata interface, as specified in table 2, the renderer shall set object_width_X = object_width_Y = object_width_Z = object_width.

Let $\mathrm{flr} = \begin{cases} -1, & when \quad \text{output\_speaker\_config} = 7 \\ 0, & otherwise \end{cases}$.

The renderer shall create three lists of virtual positions, where each list contains equidistant positions in one dimension. The lists span the range [0; 1] in X and Y dimensions and [*flr*; 1] in Z dimension.

Let N_x be the number of X positions.

Let N_y be the number of Y positions.

Let N_z be the number of Z positions.

Let $N_z \geq \begin{cases} 16, & when \quad \text{output\_speaker\_config} = 7 \\ 8, & otherwise \end{cases}$.

Let $N_x \geq 20$.

Let $N_y \geq 20$.

Let $s_x \in [0; N_x - 1]; s_y \in [0; N_y - 1]; s_z \in [0; N_z - 1]$.

Let $\text{vsource\_X}[s_x]$ be the position on the X-axis at index s_x.

Let $\text{vsource\_Y}[s_y]$ be the position on the Y-axis at index s_y.

Let $\text{vsource\_Z}[s_z]$ be the position on the Z-axis at index s_z.

Let $j \in [0; \text{num\_speakers} - 1]$ and $\text{mdu} \in [0; \text{num\_obj\_info\_blocks} - 1]$.

Let *panner_target_X*, *panner_target_Y* and *panner_target_Z* be the lists of panner targets, as specified in clause A.2.

For each value of s_x, the renderer shall calculate *panner_gain_X* as: $\text{panner\_gain\_X}[s_x] = \text{pan\_X}(\text{panner\_target\_X}, \text{panner\_target\_Y}, \text{panner\_target\_Z}, \text{vsource\_X}[s_x])$, as specified in Pseudocode 6.

For each value of s_y, the renderer shall calculate *panner_gain_Y* as: $\text{panner\_gain\_Y}[s_y] = \text{pan\_Y}(\text{panner\_target\_Y}, \text{panner\_target\_Z}, \text{vsource\_Y}[s_y])$, as specified in Pseudocode 5.

For each value of s_z, the renderer shall calculate *panner_gain_Z* as: $\text{panner\_gain\_Z}[s_z] = \text{pan\_Z}(\text{panner\_target\_Z}, \text{vsource\_Z}[s_z])$, as specified in Pseudocode 4.

For each dimension, the renderer shall calculate the object size as:

$\text{object\_size\_X}[\text{mdu}] = \text{size\_scale}(\text{object\_width\_X}[\text{mdu}])$,

$\text{object\_size\_Y}[\text{mdu}] = \text{size\_scale}(\text{object\_width\_Y}[\text{mdu}])$, and

$\text{object\_size\_Z}[\text{mdu}] = \text{size\_scale}(\text{object\_width\_Z}[\text{mdu}])$,

where the *size_scale()* function is specified in Pseudocode 7.

For each dimension, the renderer shall calculate an object size with lower bounds as:

$$\text{object\_size\_X\_lb}[\text{mdu}] = \max\left(\text{object\_size\_X}[\text{mdu}], \frac{2}{N_x - 1}\right)$$

$$\text{object\_size\_Y\_lb}[\text{mdu}] = \max\left(\text{object\_size\_Y}[\text{mdu}], \frac{2}{N_y - 1}\right)$$

$$\text{object\_size\_Z\_lb}[\text{mdu}] = \max\left(\text{object\_size\_Z}[\text{mdu}], \frac{2 \times (1 - \text{flr})}{N_z - 1}\right)$$

For each dimension, the renderer shall calculate the weight as:

$\text{weight\_X}[s_x][\text{mdu}] = \text{weight}(\text{vsource\_X}[s_x], \text{object\_position\_X}[\text{mdu}], \text{object\_size\_X\_lb}[\text{mdu}])$,

$\text{weight\_Y}[s_y][\text{mdu}] = \text{weight}(\text{vsource\_Y}[s_y], \text{object\_position\_Y}[\text{mdu}], \text{object\_size\_Y\_lb}[\text{mdu}])$, and weight_Z[$s_z$][mdu] = weight(vsource_Z[s_z], max(flr, object_position_Z[mdu]), object_size_Z_lb[mdu]),

where the function y = weight(a, b, c) is specified in Pseudocode 8.

The renderer shall calculate p and size_eff as specified in Pseudocode 9.

The renderer shall calculate size_inside_gain[j][mdu] as specified in Pseudocode 10.

The renderer shall calculate *panner_gain_front* as: panner_gain_front = pan_Y(panner_target_Y, panner_target_Z, 0), as specified in Pseudocode 5.

The renderer shall calculate *panner_gain_back* as: panner_gain_back = pan_Y(panner_target_Y, panner_target_Z, 1), as specified in Pseudocode 5.

The renderer shall calculate *panner_gain_left* as: panner_gain_left = pan_X(panner_target_X, panner_target_Y, panner_target_Z, 0), as specified in Pseudocode 6.

The renderer shall calculate *panner_gain_right* as: panner_gain_right = pan_X(panner_target_X, panner_target_Y, panner_target_Z, 1), as specified in Pseudocode 6.

The renderer shall calculate *panner_gain_ceil* as: panner_gain_ceil = pan_Z(panner_target_Z, 1), as specified in Pseudocode 4.

The renderer shall calculate *panner_gain_flr* as: panner_gain_flr = pan_Z(panner_target_Z, flr), as specified in Pseudocode 4.

The renderer shall calculate size_bound_gain as specified in Pseudocode 11.

The renderer shall calculate size_gain as specified in Pseudocode 12, where the function y = fade(position, size, min_val) is specified in Pseudocode 13.

The renderer shall calculate point_panner_gain as specified in clause 5.2.2.2.

NOTE: The result of the calculation in clause 5.2.2.2 is now assigned to point_panner_gain instead of panner_gain to avoid a symbol clash.

The renderer shall calculate panner_gain as specified in Pseudocode 14.

5.2.3.3 Algorithmic details

Pseudocode 7

```
y = size_scale (x)
{
  epsilon = 1e-5;
  if (x < epsilon)
  {
    y = 0;
  }
  else if (x <= 0.2)
  {
    y = 1.5 * x;
  }
  else if (x <= 0.5)
  {
    y = 7/3 * x - 1/6;
  }
  else if (x <= 0.75)
  {
    y = 3.2 * x - 0.6;
  }
  else
  {
    y = 4 * x - 1.2;
  }
}
```

Pseudocode 8

```
y = weight (a, b, c)
{
  if (c > 0)
  {
    tmp  = 1.5 * (a - b) / c;
    tmp2 = pow(tmp, 4)
    y = pow(10, -tmp2);
  }
  else
  {
    y = 0;
  }
}
```

Pseudocode 9

```
f1 = 2 / 3;
f2 = 2 / 9;
f3 = 1 / 9;
f4 = -4 / 2.3;

for (mdu = 0; mdu < num_obj_info_blocks; mdu++)
{
  {x1, x2, x3} = sort_descending(object_size_X[mdu], object_size_Y[mdu], object_size_Z[mdu]);
  size_eff[mdu] = f1 * x1 + f2 * x2 + f3 * x3;
  if (size_eff[mdu] > 0.5)
  {
    p[mdu] = 6 + (size_eff[mdu] - 0.5) * f4;
  }
  else
  {
    p[mdu] = 6;
  }
}
```

Pseudocode 10

```
tolerance = 0.00001;
w_panner_gain_X[num_obj_info_blocks][num_speakers] = {{0}, {0}};
w_panner_gain_Y[num_obj_info_blocks][num_speakers] = {{0}, {0}};
w_panner_gain_Z[num_obj_info_blocks][num_speakers] = {{0}, {0}};

for (mdu=0; mdu < num_obj_info_blocks; mdu++)
{
  total = 0;

  for (j=0; j < num_speakers; j++)
  {
    for (s=0; s < Nx; s++)
    {
      w_panner_gain_X[mdu][j] += pow(panner_gain_X[s][j] * weight_X[s][mdu], p[mdu]);
    }

    for (s=0; s < Ny; s++)
    {
      w_panner_gain_Y[mdu][j] += pow(panner_gain_Y[s][j] * weight_Y[s][mdu], p[mdu]);
    }

    for (s=0; s < Nz; s++)
    {
      w_panner_gain_Z[mdu][j] += pow(panner_gain_Z[s][j] * weight_Z[s][mdu], p[mdu]);
    }

    size_inside_gain[mdu][j] = pow(w_panner_gain_X[mdu][j] * w_panner_gain_Y[mdu][j]
                                  * w_panner_gain_Z[mdu][j], 1 / p[mdu]);
    total += pow(size_inside_gain[mdu][j], 2);
  }

  total = sqrt(total);
  for (j=0; j < num_speakers; j++)
  {
    size_inside_gain[mdu][j] /= max(tolerance, total);
```

```
  }
}
```

Pseudocode 11

```
tolerance = 0.00001;
for (mdu = 0; mdu < num_obj_info_blocks; mdu++)
{
  total = 0;

  for (j = 0; j < num_speakers; j++)
  {
     sum1 = pow(panner_gain_flr[j] * weight(flr, max(object_position_Z[mdu], flr)
     , object_size_Z_lb[mdu]), p) * w_panner_gain_X[mdu][j] * w_panner_gain_Y[mdu][j];

     sum2 = pow(panner_gain_ceil[j] * weight(1, max(object_position_Z[mdu], flr)
     , object_size_Z_lb[mdu]), p) * w_panner_gain_X[mdu][j] * w_panner_gain_Y[mdu][j];

     sum3 = pow(panner_gain_left[j] * weight(0, object_position_X[mdu], object_size_X_lb[mdu]), p)
            * w_panner_gain_Y[mdu][j] * w_panner_gain_Z[mdu][j];

     sum4 = pow(panner_gain_right[j] * weight(1, object_position_X[mdu], object_size_X_lb[mdu]), p)
            * w_panner_gain_Y[mdu][j] * w_panner_gain_Z[mdu][j];

     sum5 = pow(panner_gain_front[j] * weight(0, object_position_Y[mdu], object_size_Y_lb[mdu]), p)
            * w_panner_gain_X[mdu][j] * w_panner_gain_Z[mdu][j];

     sum6 = pow(panner_gain_back[j] * weight(1, object_position_Y[mdu], object_size_Y_lb[mdu]), p)
            * w_panner_gain_X[mdu][j] * w_panner_gain_Z[mdu][j];

     size_bound_gain[mdu][j] = sum1 + sum2 + sum3 + sum4 + sum5 + sum6;
     total += pow(size_bound_gain[mdu][j], 2);
  }

  total = sqrt(total);
  for (j = 0; j < num_speakers; j++)
  {
    size_bound_gain[mdu][j] /= max(tolerance, total);
  }
}
```

Pseudocode 12

```
tolerance = 0.00001;
for (mdu=0; mdu < num_obj_info_blocks; mdu++)
{
  total = 0;
  out_fact[mdu] = fade(object_position_X[mdu], object_size_X_lb[mdu], 0)
              * fade(object_position_Y[mdu], object_size_Y_lb[mdu], 0)
              * fade(max(object_position_Z[mdu], flr), object_size_Z_lb[mdu], flr);

  for (j = 0; j < num_speakers; j++)
  {
    size_gain[mdu][j] = pow(size_bound_gain[mdu][j] + out_fac[mdu] * size_inside_gain[mdu][j]
                        ,1 / p[mdu]);
    total += pow(size_gain[mdu][j], 2);
  }

  total = sqrt(total[mdu]);
  for (j = 0; j < num_speakers; j++)
  {
    size_gain[mdu][j] /= max(tolerance, total);
  }
}
```

Pseudocode 13

```
y = fade(position, size, min_val)
{
  /* distance of an object from the closest boundary in the room */

  tmp = min(position - min_val, 1 - position);

  if (tmp < size || tmp < 0.2)
  {
    y = tmp / max(0.2, size);
  }
  else
  {
    y = 1;
  }
}
```

Pseudocode 14

```
tolerance = 0.00001;
for (mdu = 0; mdu < num_obj_info_blocks; mdu++)
{
  total = 0;
  if (size_eff[mdu] < 0.2)
  {
    alpha = cos(size_eff[mdu] / 0.2 * pi / 2);
    beta =  sin(size_eff[mdu] / 0.2 * pi / 2);
  }
  else
  {
    alpha = 0;
    beta  = 1;
  }

  for (j = 0; j < num_speakers; j++)
  {
    panner_gain[mdu][j] = alpha * point_panner_gain[mdu][j] + beta * size_gain[mdu][j];
    total += pow(panner_gain[mdu][j], 2);
  }

  total = sqrt(total);
  for (j = 0; j < num_speakers; j++)
  {
    panner_gain[mdu][j] /= max(tolerance, total);
  }
}
```

5.2.4 Rendering objects with divergence

5.2.4.1 Introduction

Divergence converts one object into two objects, with the following three-dimensional position coordinates: $\left(\text{object\_position\_X} \pm {}^{\text{object\_divergence}}/_{2}; \text{object\_position\_Y}; \text{object\_position\_Z}\right)$. These two objects are panned by the divergence panner and the resulting loudspeaker gains are normalized in an energy-preserving way.

When an object is located close to the room boundary and, based on the calculation above, one of the divergent objects is located outside the room, the divergence panner limits the divergence amount to ensure that the divergent objects remain within the room boundaries.

5.2.4.2 Requirements

The requirements in this clause apply if

- the object position is specified by *object_position_X*, *object_position_Y* and *object_position_Z*.
- *object_divergence* > 0.

Let $j \in [0; \text{num\_speakers} - 1]$ and $\text{mdu} \in [0; \text{num\_obj\_info\_blocks} - 1]$.

The source panner shall set $\text{object\_width\_X}[\text{mdu}] = \text{object\_width\_Y}[\text{mdu}] = \text{object\_width\_Z}[\text{mdu}] = 0$.

For each metadata update *mdu*, the divergence panner shall determine the *panner_target_X*, *panner_target_Y*, and *panner_target_Z* lists, as specified in clause A.2.

The divergence panner shall calculate *panner_gain[mdu][j]*, as specified in Pseudocode 15, utilizing the *pan_X*, *pan_Y*, and *pan_Z* functions, as specified in clause 5.2.2.3.

5.2.4.3 Algorithmic details

Pseudocode 15

```
for (mdu = 0; mdu < num_obj_info_blocks; mdu++) /* for each metadata update */
{
  tmp_div = min(object_divergence[mdu], 1  - abs(object_position_X[mdu] – 0.5) * 2);
  x1 = object_position_X[mdu] + tmp_div / 2;
  x2 = object_position_X[mdu] – tmp_div / 2;
  total = 0;

  /* scalar multiplication for the vectors panner_target_* of length num_speakers to
  calculate the matrix panner_gain[num_obj_info_blocks][num_speakers] */
  panner_gain[mdu] = pan_Z(panner_target_Z, object_position_Z[mdu])
                   * pan_Y(panner_target_Y, panner_target_Z, object_position_Y[mdu])
                   * ( pan_X(panner_target_X, panner_target_Y, panner_target_Z, x1)
                     + pan_X(panner_target_X, panner_target_Y, panner_target_Z, x2));

  for (j = 0; j < num_speakers; j++)
  {
    total += pow(panner_gain[mdu][j], 2)
  }
  total = sqrt(total);
  for (j = 0; j < num_speakers; j++)
  {
    panner_gain[mdu][j] /= total;
  }
}
```

5.2.5 Rendering objects with speaker-anchored coordinates

5.2.5.1 Introduction

Speaker-anchored coordinates specify a location in the listening room that is related to the position of a specific loudspeaker. The speaker-anchored coordinates are assigned to a specific loudspeaker and are indicated by a label that specifies a channel corresponding to that loudspeaker. Ideally, the loudspeaker corresponding to the channel is available in the output loudspeaker layout, so the object audio essence can be routed entirely to that loudspeaker. If the loudspeaker is not available in the output loudspeaker layout, the object essence may be routed to a different loudspeaker or the object essence may be panned between two loudspeakers.

5.2.5.2 Requirements

The requirements in this clause apply if the object position is specified by *channel*.

Let *mdu* be the metadata update in range $[0; \text{num\_object\_info\_blocks} - 1]$.

Let $m = \sqrt{\frac{1}{2}}$.

If *Lb* and *Rb* are present, as specified in ETSI TS 103 190-2 [1], clause 6.3.2.9.5, 6.3.2.9.8 or 6.3.2.9.9, let $g = m$; otherwise, let $g = 1$.

Let *j* be in range $[0; \text{num\_speakers} - 1]$, as specified for the set *output_speaker_config* in table A.1.

NOTE: The value of *num_speakers* is determined as listed in table A.1, i.e. not reduced by the zone constraints processing.

For the incoming speaker-anchored position metadata, as indicated by *channel*, and the set *output_speaker_config*, the renderer shall determine $\text{panner\_gain}[\text{mdu}][j] = y$, where the values of *j* and *y* are listed as pair $\{j; y\}$ in table 6.

The value of $\text{panner\_gain}[\text{mdu}][j]$ shall be 0, if the corresponding value is not specified in table 6.

Table 6: Output loudspeaker configuration

Channel	Output_speaker_config								
	0	1	2	3	4	5	6	7	8
L	{0; 1}	{0; 1}	{0; 1}	{0; 1}	{0; 1}	{0; 1}	{0; 1}	{0; 1}	{0; 1}
R	{1; 1}	{1; 1}	{1; 1}	{1; 1}	{1; 1}	{1; 1}	{1; 1}	{1; 1}	{1; 1}
C	{0; m}, {1; m}	{2; 1}	{2; 1}	{2; 1}	{2; 1}	{2; 1}	{2; 1}	{2; 1}	{2; 1}
Ls	{0; g×m}	{3; g}	{3; 1}	{3; 1}	{3; 1}	{3; g}	{3; g}	{3; 1}	{3; 1}
Rs	{1; g×m}	{4; g}	{4; 1}	{4; 1}	{4; 1}	{4; g}	{4; g}	{4; 1}	{4; 1}
Lb	{0; 0,5}	{3; m}	{5; 1}	{5; 1}	{5; 1}	{3; m}	{3; m}	{5; 1}	{5; 1}
Rb	{1; 0,5}	{4; m}	{6; 1}	{6; 1}	{6; 1}	{4; m}	{4; m}	{6; 1}	{6; 1}
Tfl	{0; 0,5}	{3; m}	{3; m}	{7; 1}	{7; m}	{5; 1}	{5; m}	{7; 1}	{7; 1}
Tfr	{1; 0,5}	{4; m}	{4; m}	{8; 1}	{8; m}	{6; 1}	{6; m}	{8; 1}	{8; 1}
Tl	{0; 0,5}	{3; m}	{3; m}	{7; m}, {9; m}	{7; 1}	{5; m}, {7; m}	{5; 1}	{11; 1}	{7; 1}
Tr	{1; 0,5}	{4; m}	{4; m}	{8; m}, {10; m}	{8; 1}	{6; m}, {8; m}	{6; 1}	{12; 1}	{8; 1}
Tbl	{0; 0,5}	{3; m}	{3; m}	{9; 1}	{7; m}	{7; 1}	{5; m}	{9; 1}	{9; m}
Tbr	{1; 0,5}	{4; m}	{4; m}	{10; 1}	{8; m}	{8; 1}	{6; m}	{10; 1}	{9; m}
Lw	{0; 0,5}	{0; m}	{0; m}	{0; m}	{0; m}	{0; m}	{0; m}	{20; 1}	{0; m}
Rw	{1; 0,5}	{1; m}	{1; m}	{1; m}	{1; m}	{1; m}	{1; m}	{21; 1}	{1; m}
LFE	{2; 1}	{5; 1}	{7; 1}	{11; 1}	{9; 1}	{9; 1}	{7; 1}	{22; 1}	{10;1}
LFE2	{2; 1}	{5; 1}	{7; 1}	{11; 1}	{9; 1}	{9; 1}	{7; 1}	{23; 1}	{11,1}

5.3 Trim

5.3.1 Introduction

Trim can be used to lower the level of out-of-screen elements that are included in the mix. This can be desirable when immersive mixes are reproduced in layouts with few loudspeakers (for example, stereo, 5.1- or 7.1-channel).

Trim can preserve intelligibility of on-screen located objects and can avoid the effect of an increased perceived loudness of out-of-screen located objects when rendered to a loudspeaker layout with fewer loudspeakers than the mastering loudspeaker layout.

The trim process supports custom parameters, but it also applies to the content if these custom parameters are not present at the metadata interface. For the latter case, the present document specifies a default trim algorithm that does not use custom parameters.

5.3.2 Requirements

Based on the *num_top* and *num_mid* values for the set *output_speaker_config* specified in table A.1, the renderer shall set *trim_cfg*, as specified in table 7.

If the position metadata is specified by *object_position_X*, *object_position_Y* and *object_position_Z*, the *trim_gain* shall be calculated as follows:

- If $\text{trim\_mode}[\text{trim\_cfg}] = \text{disable\_trim}$, the renderer shall set *trim_gain* to 1.
- If $\text{trim\_mode}[\text{trim\_cfg}] = \text{default\_trim}$, the renderer shall calculate *trim_gain* as specified in Pseudocode 16.
- If $\text{trim\_mode}[\text{trim\_cfg}] = \text{custom\_trim}$, the renderer shall calculate *trim_gain* as specified in Pseudocode 17.

If the position metadata is specified by *channel*, the value of *trim_gain* shall be set to 1.

Table 7: *trim_cfg* values

num_mid	*num_top* = 0	*num_top* = 2	*num_top* > 2
0	*trim_cfg* = 0	*trim_cfg* = 3	*trim_cfg* = 6
2 | 3	*trim_cfg* = 1	*trim_cfg* = 4	*trim_cfg* = 7
≥ 4	*trim_cfg* = 2	*trim_cfg* = 5	*trim_cfg* = 8

5.3.3 Algorithmic details

Pseudocode 16

```
start_effect_Y = 0.0;
full_effect_Y  = 0.6;
start_effect_Z = 0.2;
full_effect_Z  = 1.0;
max_trim = min(0, - 4.5 + 3 * min (1, num_mid / 4)  + 1.5 * min(1, num_top / 4);

for (mdu = 0; mdu < num_obj_info_blocks; mdu++)
{
  gY = clamp((object_position_Y[mdu] - start_effect_Y)
     / (full_effect_Y - start_effect_Y));
  gZ = clamp((abs(object_position_Z[mdu]) - start_effect_Z)
     / (full_effect_Z - start_effect_Z));

  trim_gain_db[mdu] = max_trim * clamp(gY + gZ);
  trim_gain[mdu] = db_to_linear(trim_gain_db[mdu]);
}
```

NOTE: The *num_mid* and *num_top* values depend on the output loudspeaker configuration that is set, as specified in table A.1.

Pseudocode 17

```
siz = 0.05;
tran = 0.1;
for (mdu = 0; mdu < num_obj_info_blocks - 1; mdu++)
{
  trim_Z = trim_height[trim_cfg] * abs(object_position_Z[mdu]);
  trim_Y =  (object_position_Y[mdu] ≥ 0.5) ? trim_surround[trim_cfg]
          : trim_surround[trim_cfg] * (2 * object_position_Y[mdu]);

  if (output_speaker_config != 0)
  {
    trim_C = 0;
  }
  else
  {
    trim_C = (1 - clamp(max(max(abs(object_position_X[mdu] - 0.5) - siz, object_position_Y[mdu] - si
z)
              , abs(object_position_Z[mdu]) - siz) / tran)) * trim_centre[trim_cfg];
  }

  trim_gain[mdu] = db_to_linear(trim_C + trim_Y + trim_Z);
}
```

5.4 Gain mixer

5.4.1 Introduction

The gain mixer creates speaker gain values by combining gain values calculated by the source panner with the trim gain values and the custom gain values. The resulting loudspeaker gain values are input to the ramp mixer.

5.4.2 Requirements

If *b_object_not_active* is true, the renderer shall set $\text{object\_gain} = -\infty$.

The renderer shall calculate the speaker gain values as specified in Pseudocode 18.

5.4.3 Algorithmic details

Pseudocode 18

```
for (mdu=  0; mdu < num_obj_info_blocks; mdu++)
{
  for (j = 0; j < num_speakers; j++)
  {
    speaker_gain[mdu][j] = panner_gain[mdu][j] * trim_gain[mdu] * db_to_linear(object_gain[mdu]);

  }
}
```

5.5 Ramp mixer

5.5.1 Introduction

The ramp mixer creates output loudspeaker feeds *y[j]* from the object audio essence, attenuated with *speaker_gain[j]* values calculated in clause 5.2.2.3.

The effective mixing gains *mix_gain[j]* crossfade to the *speaker_gain[j]* values over a period of time specified by *ramp_duration*.

5.5.2 Requirements

The renderer shall calculate $\text{start\_smp[mdu]} = \text{sample\_offset} + \text{block\_offset\_factor[mdu]} \times 32$ for each incoming metadata update $\text{mdu} \in [0; \text{num\_obj\_info\_blocks} - 1]$.

The renderer shall calculate loudspeaker feeds *y(j)* as specified in Pseudocode 19.

5.5.3 Algorithmic details

Pseudocode 19

```
mdu = 0;
for (smp = 0; smp < frame_size; smp++)
{
  /* check for new metadata update starting at the current sample */
  if (smp == start_smp[mdu])
  {
    /* set new ramp duration */
    ramp_dur = ramp_duration[mdu];
    remain_ramp_dur = ramp_dur;

    /* calculate new ramp delta values */
    for (j = 0; j < num_speakers; j++)
    {
      ramp_delta[j] = (speaker_gain[mdu][j] - mix_gain[j]) / (ramp_dur + 1);
    }
```

```
    /* avoid out of bound for pointer to next update */
    if (mdu < num_obj_info_blocks - 1) mdu++;
  }

  /* calculate output speaker feeds */
  for (j = 0; j < num_speakers; j++)
  {
    /* update mixgain if still ramping */
    if (remain_ramp_dur ≥ 0)
    {
      mix_gain[j] += ramp_delta[j];
      remain_ramp_dur--;
    }

    /* apply mixgain to input object essence */
    y[j][smp] += x[smp] * mix_gain[j];
  }
}
```

Annex A (normative): Loudspeaker configurations and source panner coordinates

A.1 Introduction

Table A.1 lists the supported output loudspeaker configurations for the renderer that is specified in the present document.

Table A.2 through table A.10 list loudspeaker names, abbreviations, and physical locations by referencing the loudspeaker notation as specified in Recommendation ITU-R BS.2051-0 [2]. Table A.2 through table A.10 also list the coordinates (X; Y; Z) for each panner target corresponding to the loudspeaker. These coordinates are used by the source panner algorithm.

NOTE: The coordinates for the panner targets are for internal use of the source panner and are not necessarily meant to reflect the exact position of a loudspeaker in the room.

A.2 Requirements

The renderer shall set *num_speakers*, *num_top* and *num_mid* to the values listed in table A.1 based on:

- the output loudspeaker configuration, as specified in clause 4.2.3; and
- the zone constraints metadata preprocessing, as specified in clause 5.1.3.

For each loudspeaker *j*, ordered top-down in the corresponding table (table A.2 through table A.10), the renderer shall set the panner target coordinates as follows:

- *panner_target_X[j]* to the value listed in column X;
- *panner_target_Y[j]* to the value listed in column Y; and
- *panner_target_Z[j]* to the value listed in column Z;

based on:

- the output loudspeaker configuration, as specified in clause 4.2.3; and
- the zone constraints metadata preprocessing, as specified in clause 5.1.3.

A.3 Tables

Table A.1: Supported loudspeaker configurations

***output_speaker_config* index**	**Loudspeaker configuration name**	***num_speakers* (excluding LFE and LFE2)**	***num_top***	***num_mid***	**See also**
0	2.X or 2/0/0	2	0	0	Table A.2
1	5.X or 3/2/0	5	0	2	Table A.3
2	7.X or 3/4/0	7	0	4	Table A.4
3	3/4/4	11	4	4	Table A.5
4	3/4/2	9	2	4	Table A.6
5	3/2/4	9	4	2	Table A.7
6	3/2/2	7	2	2	Table A.8
7	22.2	22	9	7	Table A.9
8	10.2	10	3	4	Table A.10
NOTE: X can be either 1 or 0.					

Table A.2: 2.1 or 2/0/0 configuration

	Loudspeaker				**Panner target**	
Index	**Loudspeaker name**	**Loudspeaker abbreviation**	**Loudspeaker positions as specified in Recommendation ITU-R BS.2051-0 [2]**	**X**	**Y**	**Z**
0	Left	*L*	*M+030*	0	0	0
1	Right	*R*	*M-030*	1	0	0
2	Low-Frequency Effects	*LFE*	*LFE1*	N/A	N/A	N/A
NOTE: If the LFE loudspeaker is not present, the loudspeaker configuration is 2.0.						

Table A.3: 5.1 or 3/2/0 configuration

Loudspeaker				**Panner target**		
Index	**Loudspeaker name**	**Loudspeaker abbreviation**	**Loudspeaker positions as specified in Recommendation ITU-R BS.2051-0 [2]**	**X**	**Y**	**Z**
0	Left	*L*	*M+030*	0	0	0
1	Right	*R*	*M-030*	1	0	0
2	Centre	*C*	*M+000*	0,5	0	0
3	Left surround	*Ls*	*M+110*	0	0,5	0
4	Right surround	*Rs*	*M-110*	1	0,5	0
5	Low-Frequency Effects	*LFE*	*LFE1*	N/A	N/A	N/A
NOTE: If the LFE loudspeaker is not present, the loudspeaker configuration is 5.0.						

Table A.4: 7.1 or 3/4/0 configuration

Loudspeaker				**Panner target**		
Index	**Loudspeaker name**	**Loudspeaker abbreviation**	**Loudspeaker positions as specified in Recommendation ITU-R BS.2051-0 [2]**	**X**	**Y**	**Z**
0	Left	*L*	*M+030*	0	0	0
1	Right	*R*	*M-030*	1	0	0
2	Centre	*C*	*M+000*	0,5	0	0
3	Left side	*Lss*	*M+090*	0	0,5	0
4	Right side	*Rss*	*M-090*	1	0,5	0
5	Left back	*Lb*	*M+135*	0	1	0
6	Right back	*Rb*	*M-135*	1	1	0
7	Low-Frequency Effects	*LFE*	*LFE1*	N/A	N/A	N/A
NOTE: If the LFE loudspeaker is not present, the loudspeaker configuration is 7.0.						

Table A.5: 3/4/4 configuration

Loudspeaker				Panner target		
Index	**Loudspeaker name**	**Loudspeaker abbreviation**	**Loudspeaker positions as specified in Recommendation ITU-R BS.2051-0 [2]**	**X**	**Y**	**Z**
0	Left	*L*	*M+030*	0	0	0
1	Right	*R*	*M-030*	1	0	0
2	Centre	*C*	*M+000*	0,5	0	0
3	Left side	*Lss*	*M+090*	0	0,5	0
4	Right side	*Rss*	*M-090*	1	0,5	0
5	Left back	*Lb*	*M+135*	0	1	0
6	Right back	*Rb*	*M-135*	1	1	0
7	Top front left	*Tfl*	*U+030*	0,25	0,25	1
8	Top front right	*Tfr*	*U-030*	0,75	0,25	1
9	Top back left	*Tbl*	*U+135*	0,25	0,75	1
10	Top back right	*Tbr*	*U-135*	0,75	0,75	1
11	Low-Frequency Effects	*LFE*	*LFE1*	N/A	N/A	N/A

Table A.6: 3/4/2 configuration

Loudspeaker				Panner target		
Index	**Loudspeaker name**	**Loudspeaker abbreviation**	**Loudspeaker positions as specified in Recommendation ITU-R BS.2051-0 [2]**	**X**	**Y**	**Z**
0	Left	*L*	*M+030*	0	0	0
1	Right	*R*	*M-030*	1	0	0
2	Centre	*C*	*M+000*	0,5	0	0
3	Left side	*Lss*	*M+090*	0	0,5	0
4	Right side	*Rss*	*M-090*	1	0,5	0
5	Left back	*Lb*	*M+135*	0	1	0
6	Right back	*Rb*	*M-135*	1	1	0
7	Top left	*Tl*	*U+090*	0,25	0,5	1
8	Top right	*Tr*	*U-090*	0,75	0,5	1
9	Low-Frequency Effects	*LFE*	*LFE1*	N/A	N/A	N/A

Table A.7: 3/2/4 configuration

Loudspeaker				Panner target		
Index	**Loudspeaker name**	**Loudspeaker abbreviation**	**Loudspeaker positions as specified in Recommendation ITU-R BS.2051-0 [2]**	**X**	**Y**	**Z**
0	Left	*L*	*M+030*	0	0	0
1	Right	*R*	*M-030*	1	0	0
2	Centre	*C*	*M+000*	0,5	0	0
3	Left surround	*Ls*	*M+110*	0	0,5	0
4	Right surround	*Rs*	*M-110*	1	0,5	0
5	Top front left	*Tfl*	*U+030*	0,25	0,25	1
6	Top front right	*Tfr*	*U-030*	0,75	0,25	1
7	Top back left	*Tbl*	*U+135*	0,25	0,75	1
8	Top back right	*Tbr*	*U-135*	0,75	0,75	1
9	Low-Frequency Effects	*LFE*	*LFE1*	N/A	N/A	N/A

Table A.8: 3/2/2 configuration

Loudspeaker				Panner target		
Index	**Loudspeaker name**	**Loudspeaker abbreviation**	**Loudspeaker positions as specified in Recommendation ITU-R BS.2051-0 [2]**	**X**	**Y**	**Z**
0	Left	*L*	*M+030*	0	0	0
1	Right	*R*	*M-030*	1	0	0
2	Centre	*C*	*M+000*	0,5	0	0
3	Left surround	*Ls*	*M+110*	0	0,5	0
4	Right surround	*Rs*	*M-110*	1	0,5	0
5	Top left	*Tl*	*U+090*	0,25	0,5	1
6	Top right	*Tr*	*U-090*	0,75	0,5	1
7	Low-Frequency Effects	*LFE*	*LFE1*	N/A	N/A	N/A

Table A.9: 22.2 configuration

Loudspeaker				Panner target		
Index	**Loudspeaker name**	**Loudspeaker abbreviation**	**Loudspeaker positions as specified in Recommendation ITU-R BS.2051-0 [2]**	**X**	**Y**	**Z**
0	Left	*L*	*M+030*	0	0	0
1	Right	*R*	*M-030*	1	0	0
2	Centre	*C*	*M+000*	0,5	0	0
3	Left side	*Lss*	*M+090*	0	0,5	0
4	Right side	*Rss*	*M-090*	1	0,5	0
5	Left back	*Lb*	*M+135*	0	1	0
6	Right back	*Rb*	*M-135*	1	1	0
7	Top front left	*Tfl*	*U+030*	0,25	0,25	1
8	Top front right	*Tfr*	*U-030*	0,75	0,25	1
9	Top back left	*Tbl*	*U+135*	0,25	0,75	1
10	Top back right	*Tbr*	*U-135*	0,75	0,75	1
11	Top side left	*Tsl*	*U+90*	0,25	0,5	1
12	Top side right	*Tsr*	*U-90*	0,75	0,5	1
13	Top front centre	*Tfc*	*U+000*	0,5	0,25	1
14	Top back centre	*Tbc*	*U+180*	0,5	0,75	1
15	Top centre	*Tc*	*T+000*	0,5	0,5	1
16	Bottom front left	*Bfl*	*B+45*	0	0	-1
17	Bottom front right	*Bfr*	*B-45*	1	0	-1
18	Bottom front centre	*Bfc*	*B+000*	0,5	0	-1
19	Back centre	*Cb*	*M+180*	0,5	1	0
20	Left wide	*Lw*	*M+60*	0	0,2929	0
21	Right wide	*Rw*	*M-60*	1	0,2929	0
22	Low-Frequency Effects	*LFE*	*LFE1*	N/A	N/A	N/A
23	Low-Frequency Effects	*LFE2*	*LFE2*	N/A	N/A	N/A

Table A.10: 10.2 configuration

Loudspeaker				Panner target		
Index	**Loudspeaker name**	**Loudspeaker abbreviation**	**Loudspeaker positions as specified in Recommendation ITU-R BS.2051-0 [2]**	**X**	**Y**	**Z**
0	Left	*L*	*M+030*	0	0	0
1	Right	*R*	*M-030*	1	0	0
2	Centre	*C*	*M+000*	0,5	0	0
3	Left side	*Lss*	*M+090*	0	0,5	0
4	Right side	*Rss*	*M-090*	1	0,5	0
5	Left back	*Lb*	*M+135*	0	1	0
6	Right back	*Rb*	*M-135*	1	1	0
7	Top front left	*Tfl*	*U+030*	0,25	0,25	1
8	Top front right	*Tfr*	*U-030*	0,75	0,25	1
9	Top back centre	*Tbc*	*U+180*	0,5	0,75	1
10	Low-Frequency Effects	*LFE*	*LFE1*	N/A	N/A	N/A
11	Low-Frequency Effects	*LFE2*	*LFE2*	N/A	N/A	N/A

History

Document history		
V1.1.1	September 2016	Publication