ETSI TS 103 420 V1.1.1 (2016-07)

TECHNICAL SPECIFICATION

Backwards-compatible object audio carriage using Enhanced AC-3

EBU

OPERATING EUROVISION

Reference

DTS/JTC-035

Keywords

audio, broadcasting, coding, 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

The symbolic source code for tables referenced throughout the present document is contained in archive ts_103420v010101p0.zip which accompanies the present document.

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

In traditional channel-based audio mixing, sound elements are mixed to the fixed speaker channels, i.e. left, right, centre, left surround and right surround. This paradigm is well known and works when the channel configuration at the decoding end can be predetermined, or assumed with reasonable certainty to be 2.0, 5.X, or 7.X.

However, with the popularity of new speaker setups, no assumption can be made about the speaker setup used for playback. Therefore, channel-based audio does not offer a sufficient method for adapting a presentation where the source speaker layout does not match the speaker layout at the decoding end. This presents a challenge when trying to author content that plays back well independently to the speaker configuration.

In object-based audio, individual sound elements are delivered to the playback device, where they are rendered based on the used speaker layout. Because individual sound elements can be associated with a much richer set of metadata, giving meaning to the elements, the method of adaptation to the speaker configuration reproducing the audio can provide better information regarding how to render to fewer speakers.

Enhanced AC-3 (E-AC-3), defined in ETSI TS 102 366 [1], is a widely used format for transmission of channel-based audio content. When the goal is to transport object-based audio in an environment where compatibility with pre-existing devices is paramount, joint object coding (JOC), as specified in the present document, can be used in conjunction with E-AC-3.

Document structure

The present document is structured as follows:

- Clause 4 explains the concept of object-based audio and specifies the decoder interface.
- Clause 5 specifies object audio metadata (OAMD), the object-based audio metadata format.
- Annex B specifies how OAMD can be converted to an audio definition model (ADM), providing an interconnection to the professional metadata generation and monitoring.
- Clause 6 specifies the JOC tool that converts the output of an E-AC-3 decoder to objects, as specified in ETSI TS 102 366 [1].
- Clause 7 specifies the quadrature mirror filter bank (QMF) tool that is used by the JOC tool.

1 Scope

The present document specifies an extension to the E-AC-3 codec.

The extension adds an object-based three-dimensional spatial representation of coded audio information and metadata. It is backward compatible with the one- and two-dimensional channel-based spatial representation of coded audio information as defined in ETSI TS 102 366 [1].

NOTE: In this context, backward compatibility is defined as follows: The three-dimensional spatial representation specified in the present document can be decoded on a device compliant with the syntax and semantics specified in ETSI TS 102 366 [1]. In this case, such a device will output one- or two-dimensional channel-based audio as per the coding algorithm defined in ETSI TS 102 366 [1] alone. Thus, support for decoders specified in ETSI TS 102 366 [1] and associated user experiences are fully maintained with the extension defined herein.

The present document specifies the following:

1) Syntax and semantics of the object-based audio metadata, carried via the extensible metadata delivery format (EMDF), specified in ETSI TS 102 366 [1].

2) Syntax and semantics of metadata to control a tool for conversion of one- or two-dimensional channel-based audio to a higher number of audio signals, part of the three-dimensional spatial representation (JOC).

3) Additional requirements on the E-AC-3 decoder as specified in ETSI TS 102 366 [1].

4) Requirements on the object-based audio interface.

5) Requirements on the JOC tool.

6) Requirements on the QMF tool.

7) Informative guidance for conversion from 1) to the ADM as defined in Recommendation ITU-R BS.2076 [i.1].

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 referenced 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 102 366: "Digital Audio Compression (AC-3, Enhanced AC-3) Standard".

2.2 Informative 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 referenced 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.

[i.1] Recommendation ITU-R BS.2076: "Audio Definition Model".

3 Definitions, symbols, abbreviations and conventions

3.1 Definitions

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

audio definition model: metadata model describing format and content of audio files, specified in Recommendation ITU-R BS.2076 [i.1]

bed instance: group of objects with speaker-anchored coordinates which can be used to carry pre-mixed audio content such as complex ambiance or music

channel: label that is used to assign an audio signal to a dedicated speaker position

channel-based audio: audio content that is presented by one or more audio signals and a corresponding channel configuration

channel configuration: audio content in one channel configuration entirely composed of audio signals in that configuration

coefficient: multiplicative factor in some term of a polynomial, a series or any expression

codec: computer program or device capable of encoding or decoding a digital data stream or signal

decoder interface: interoperatibility point of a decoder, accepting input data, providing output data, or both

dynamic object: object with positional metadata that may vary over time and is described by three coordinates (x, y, z)

enhanced AC-3: format for audio data, specified in ETSI TS 102 366 [1]

extensible metadata delivery format: metadata transmission container format, specified in ETSI TS 102 366 [1]

frame: one decodable unit of audio data, consisting of a sequence of bits

frame rate: rate of frames encoded, decoded or transmitted in real-time operation

intermediate spatial format: format for encoding a scene by hierarchical components

joint object coding: algorithm used to efficiently code object-based audio

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

object audio metadata: format for coding and carrying object audio properties

object: audio signal (the object essence) plus associated object audio properties (carried as object audio metadata)

object-based audio: audio content that is composed of objects

object essence: part of the object that is PCM coded

object property: one of a group of object properties, as specified in the present document, which indicate the content producer's intention of how the object essence should be rendered to a reproduction speaker system

(object) property update: one update of the object properties for one object

(object) property update information: common timing information for a property update for all objects, i.e. a group of property updates

parameter band: grouping of one or more QMF subbands sharing common parameters

pulse code modulation: standard method used to digitally represent sampled analogue signals

quadrature mirror filter bank: process in which a time-domain signal is transformed into the frequency domain and split into a filter bank comprising a number of frequency bands

QMF (sub)band: one row in a QMF matrix, representing a filtered and subsampled signal

QMF timeslot: one column in a QMF matrix, representing QMF subbands sampled at the same point in time

rendering: processing of audio content to adapt it to specific speaker layouts

room-anchored coordinates: coordinates specifying a position relative to a coordinate system that is fixed with respect to a room

screen-anchored coordinates: coordinates specifying a position relative to a coordinate system that is fixed with respect to the display surface in a room

speaker-anchored coordinates: coordinates specifying a position by choosing one speaker in a specific speaker layout

speaker feed: audio signal designated to be played back by a specific speaker

zone: sub-room located inside the listening room

3.2 Symbols

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

$\lceil x \rceil$	round x towards plus infinity
$\lfloor x \rfloor$	round x towards minus infinity
i	the imaginary unit

3.3 Abbreviations

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

ADM	Audio Definition Model
E-AC-3	Enhanced AC-3
EMDF	Extensible Metadata Delivery Format
ISF	Intermediate Spatial Format
JOC	Joint Object Coding
LFE	Low-Frequency Effects (Channel)
LFH	Left Front Height
LRH	Left Rear Height
LRS	Left Rear Surround
LSB	Least Significant Bit
LSS	Left Side Surround
LTM	Left Top Middle
LW	Left Wide
MSB	Most Significant Bit
MULZ	Middle Upper Lower Zenith
OAMD	Object Audio Metadata
PCM	Pulse Code Modulation
QMF	Quadrature Mirror Filter Bank

RFH	Right Front Height
RRH	Right Rear Height
RRS	Right Rear Surround
RSS	Right Side Surround
RTM	Right Top Middle
RW	Right Wide
SR	Stacked Ring

3.4 Conventions

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

Typographic conventions:

- Italic font denotes variables (*n* is a variable).
- Monospace font denotes bitstream elements (`bits` is a bitstream element).

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

min(x)	The minimum value of the elements of x
max(x)	The maximum value of the elements of x
floor(x)	The largest integer(s) less than or equal to the elements of x
mod(a,b)	Denotes the remainder of a after division by b
cos(x)	The cosine of the elements of x
sin(x)	The sine of the elements of x
exp(x)	The exponential value of the elements of x

NOTE 1: The return value of exp() can be complex valued.

NOTE 2: The return value of mod() can have a fractional part.

4 Object-based audio

4.1 Introduction

Object-based audio gives content creators more control over how content is rendered to loudspeakers in consumer homes.

In channel-based audio coding, a set of tracks is implicitly assigned to specific loudspeakers by associating the set of tracks with a channel configuration. If the playback speaker configuration is different from the coded channel configuration, downmixing or upmixing specifications are required to redistribute audio to the available speakers. In object-based audio coding, rendering is applied to objects that comprise the object essence in conjunction with metadata that contains individually assigned object properties. The properties more explicitly specify how the content creator intends the audio content to be rendered (that is, they place constraints on how to render the essence into speakers).

4.2 Coordinate systems

4.2.1 Room-anchored coordinate system

The room-anchored coordinate system is a left-handed Cartesian system normalized to the room cuboid.

Coordinates are specified using three components:

- The x component increases from $x = 0$ at the left wall to $x = 1$ at the right wall.
- The y component increases from $y = 0$ at the front wall to $y = 1$ at the back wall.

- The z component increases from $z = -1$ at the floor to $z = 1$ at the ceiling.

EXAMPLE: An object at the centre of the front wall has coordinates (0,5; 0; 0).

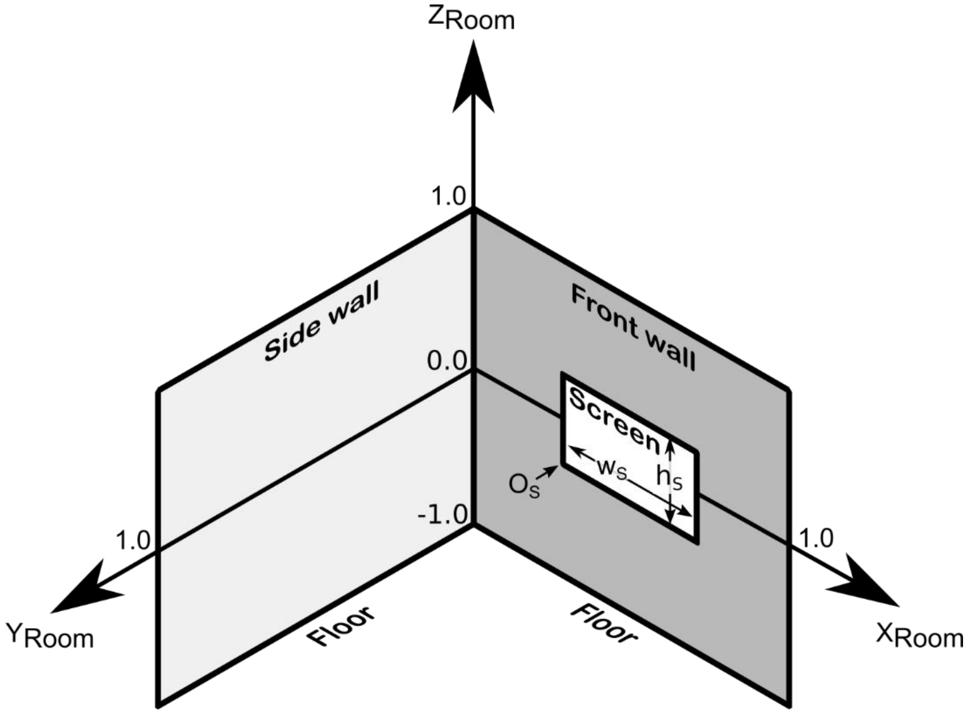


Figure 1: Room-anchored coordinate system

w_S and h_S are the dimensions of the screen, and $\mathbf{O}_S$ is the position of the bottom left screen corner.

Objects can also be outside the room.

NOTE: Object position is not bounded, though the coded representation imposes progressively coarser quantization at greater distances.

4.2.2 Screen-anchored coordinate system

In this coordinate system, the physical location of an object at a given coordinate changes with screen size and screen location.

- The x component increases from x = 0 at the left screen edge to x = 1 at the right screen edge.
- The y component increases from y = 0 at the front wall to y = 1 at the back wall.
- The z component increases from z = -1 at the bottom screen edge to z = 1 at the top screen edge.

The following formula translates screen-anchored coordinates $(X_{\text{screen}}; Y_{\text{screen}}; Z_{\text{screen}})$ to room-anchored coordinates:

$$(X_{\text{room}}; Y_{\text{room}}; Z_{\text{room}}) = \mathbf{O}_S + \left(\frac{w_S}{\text{ref\_screen\_ratio}} \times X_{\text{screen}}; Y_{\text{screen}}; \frac{h_S}{2\times\text{ref\_screen\_ratio}} \times (Z_{\text{screen}}+1)\right)$$

where w_S, h_S and $\mathbf{O}_S$ are specified in clause 4.2.1 and ref_screen_ratio is specified in clause 5.6.1.1.1 and clause 5.6.1.1.2.

EXAMPLE: An object at the centre of the left screen edge has screen-referenced coordinates $(0; 0; 0)_{\text{screen}}$ and room-referenced coordinates $\left(O_{S_x}; O_{S_y}; O_{S_z} + \frac{h_S}{2}\right)_{\text{room}}$.

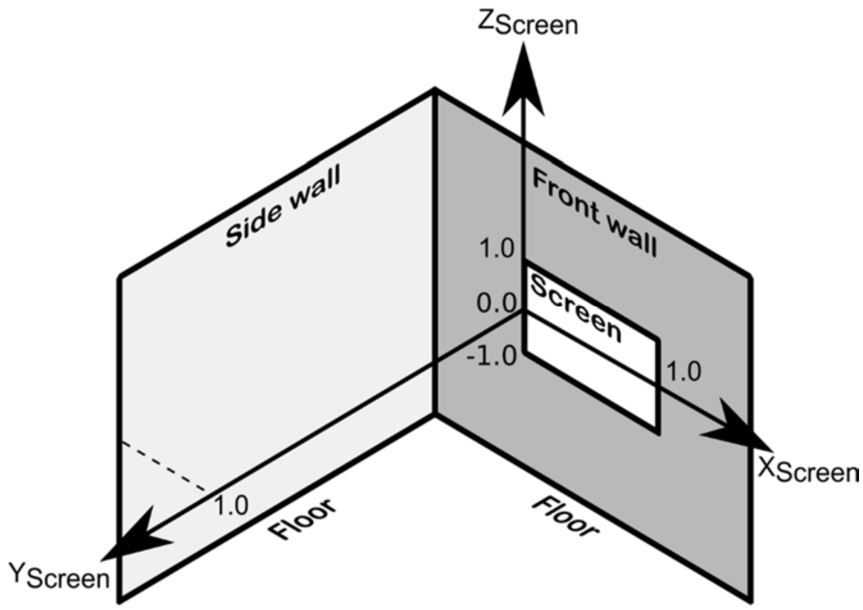


Figure 2: Screen-anchored coordinate system

4.2.3 Speaker-anchored system

In this system, the coordinates of the object are indicated by referring to a specific speaker, meaning the physical location of the object changes with the actual speaker placement.

This system is typically used with objects contained in bed instances.

NOTE: If there is no speaker at the referenced position, the physical object position can be interpolated between nearby speakers in the room.

EXAMPLE: In a 2.0 set-up, an object at "L" is rendered from the left speaker. An object at "C" is located mid-way between the L and R speakers, and may be rendered by panning.

4.3 Decoding of object-based audio content

Object-based audio content is signalled using a specific configuration of the E-AC-3 bitstream.

The decoder shall decode the incoming object-based audio content as specified in the present document, if the following requirements are met:

- The metadata payloads of JOC and OAMD are present as specified in clause 8.2.
- The incoming channel configuration is supported by the JOC tool, as specified in clause 6.3.2.2.

The decoder shall decode the incoming object-based audio content by performing the following steps:

1) Decode the channel-based audio from the incoming E-AC-3 bitstream, as specified in clause 8.1.
2) Transform the time domain signals to the QMF domain, as specified in clause 7.2.
3) Utilize the JOC decoder tool, as specified in clause 6, to reconstruct the object essences from the channel-based downmix.
4) Transform the QMF samples of the object essences to the time domain, as specified in clause 7.3.
5) Decode the object properties as specified in clause 5.2.
6) Provide the pulse code modulation (PCM) object essences and the metadata for the object properties at the decoder interface, specified in clause 4.4.

4.4 Decoder interface

The decoder shall provide an interface for the object-based audio comprising object essence audio data and time-stamped metadata updates for the corresponding object properties.

At the interface the decoder shall provide the decoded per-object metadata in time stamped updates. For each update the decoder shall provide the data specified in the *metadata_update* structure in the following pseudo-code.

Pseudocode 1

```
/// metadata_update data structure ///
struct {
  t_timing           timing_data;
  t_position         position_update;
  t_size             size_update;
  t_priority         priority_update;
  t_gain             gain_update;
  t_channel_lock     channel_lock_update;
  t_zone_constraints zone_constraints_update;
} metadata_update;

/// declaration ///
enum coordinate_system {
  ROOM,
  SCREEN,
  SPEAKER,
}

enum speaker_labels {
  RC_L, // room-centric left
  RC_R, // room-centric right
  RC_C, // room-centric centre
  RC_LFE, // room-centric low frequency effect
  RC_LSS, // room-centric left surround
  RC_RSS, // room-centric right surround
  RC_LRS, // room-centric left rear surround
  RC_RRS, // room-centric right rear surround
  RC_LFH, // room-centric left front height
  RC_RFH, // room-centric right front height
  RC_LTM, // room-centric left top middle
  RC_RTM, // room-centric right top middle
  RC_LRH, // room-centric left rear height
  RC_RRH, // room-centric right rear height
  RC_LW, // room-centric left wide
  RC_RW, // room-centric right wide
  RC_LFE2, // room-centric low frequency effects 2
}

enum zone {
  SCREEN_ZONE,
  SIDE_ZONE,
  SURROUND_ZONE,
  BACK_ZONE,
  CENTRE_AND_BACK_ZONE,
  TOP_AND_BOTTOM_ZONE,
  ZONE_COUNT = 6
}

enum zone_constraints {
  INCLUDE,
  EXCLUDE,
}

/// type definitions ///
typedef struct {
  unsigned start_sample;
  unsigned frame_offset;
  unsigned ramp_duration;
} t_timing;

typedef union {
  struct {
    float x;
    float y;
    float z;
```

```
  } 3D_coordinates;
  unsigned speaker_label; // one of enum speaker_labels
} t_coordinates;

typedef struct {
  unsigned anchor; // one of enum coordinate_system
  t_coordinates coordinates;
  float screen_factor;
  float depth_factor;
  float ref_screen_ratio;
} t_position;

typedef struct {
  float width;
  float depth;
  float height;
} t_size;

typedef float                t_priority;
typedef float                t_gain;
typedef boolean              t_channel_lock;
typedef unsigned[ZONE_COUNT] t_zone_constraints; // each one of enum zone_constraint
```

The decoded per-object metadata corresponds to the object properties, as specified in clause 5.2. The timing of the updates correspond to the temporal context of metadata updates, as specified in clause 5.3.

NOTE: Table B.10 lists room-anchored coordinates of speakers for the *speaker_labels*.

5 Object audio metadata

5.1 Introduction

OAMD is the coded bitstream representation of the metadata for object-based audio processing.

The OAMD bitstream is carried inside an EMDF container, specified in ETSI TS 102 366 [1].

The present document specifies:

- the object properties in clause 5.2;
- the timing concept within the OAMD bitstream in clause 5.3;
- the structure of the OAMD bitstream in clause 5.4;
- the OAMD bitstream syntax in clause 5.5;
- the bitstream elements contained in the OAMD bitstream in clause 5.6.

The OAMD bitstream elements signal:

- object properties metadata, as specified in clause 5.6.1;
- timing metadata, as specified in clause 5.6.2;
- OAMD content description metadata, as specified in clause 5.6.3;
- other metadata, as specified in clause 5.6.4.

5.2 Object properties

5.2.1 Position

5.2.1.1 Introduction

The position property is the data that is provided as *position_update* at the decoder interface.

The position property of an object is specified differently for the following cases:

1) The object is not contained in a bed instance and `b_object_use_screen_ref`=0; position property is specified in clause 5.2.1.2.

2) The object is not contained in a bed instance and `b_object_use_screen_ref`=1; position property is specified in clause 5.2.1.3.

3) The object is contained in a bed instance; position property is specified in clause 5.2.1.4.

NOTE: Clause 5.6.4.12 specifies how to determine if an object is contained in a bed instance.

Table 1 lists the related bitstream elements and their reference.

Table 1: Position related bitstream elements

Bitstream element	Reference
b_use_screen_ref	Clause 5.6.1.1.18
pos3D_X_bits	Clause 5.6.1.1.8
pos3D_Y_bits	Clause 5.6.1.1.9
pos3D_Z_bits	Clause 5.6.1.1.11
pos3D_Z_sign_bits	Clause 5.6.1.1.10
b_differential_position_specified	Clause 5.6.1.1.7
diff_pos3D_X_bits	Clause 5.6.1.1.12
diff_pos3D_Y_bits	Clause 5.6.1.1.13
diff_pos3D_Z_bits	Clause 5.6.1.1.14
b_object_distance_specified	Clause 5.6.1.1.15
b_object_at_infinity	Clause 5.6.1.1.16
distance_factor_idx	Clause 5.6.1.1.17
b_use_screen_ref	Clause 5.6.1.1.18
screen_factor_bits	Clause 5.6.1.1.19
depth_factor_idx	Clause 5.6.1.1.20
b_standard_chan_assign	Clause 5.6.1.1.3
bed_channel_assignment_mask	Clause 5.6.1.1.4
nonstd_bed_channel_assignment_mask	Clause 5.6.1.1.5
b_lfe_only	Clause 5.6.1.1.6

5.2.1.2 Position in room-anchored coordinates

The decoder shall set *anchor* = ROOM.

Let $\mathbf{C} = (\mathrm{pos3D\_X}; \mathrm{pos3D\_Y}; \mathrm{pos3D\_Z})$.

If `b_object_distance_specified` is false, the position $P(x_{\mathrm{Room}}; y_{\mathrm{Room}}; z_{\mathrm{Room}})$ of the object shall be determined as $\mathbf{P} = \mathbf{C}$.

NOTE 1: In this case the object is located inside the room.

If `b_object_distance_specified` is true, the position P of the object shall be determined as follows:

1) $\mathbf{O} = (0{,}5; 0{,}5; 0)$.

2) Let $\mathbf{I}$ be the point where ray $\mathbf{O} - \mathbf{C}$ intersects the room boundaries as shown in figure 3.

3) $\mathbf{P} = \text{distance\_factor} \times \mathbf{I} + (1 - \text{distance\_factor}) \times \mathbf{O}$, where $\text{distance\_factor} = \frac{d_S}{d_r}$.

NOTE 2: In this case the object is located outside the room.

The decoder shall set *3D_coordinates* to the three dimensional coordinates $(x_{\text{Room}}; y_{\text{Room}}; z_{\text{Room}})$ of the position P.

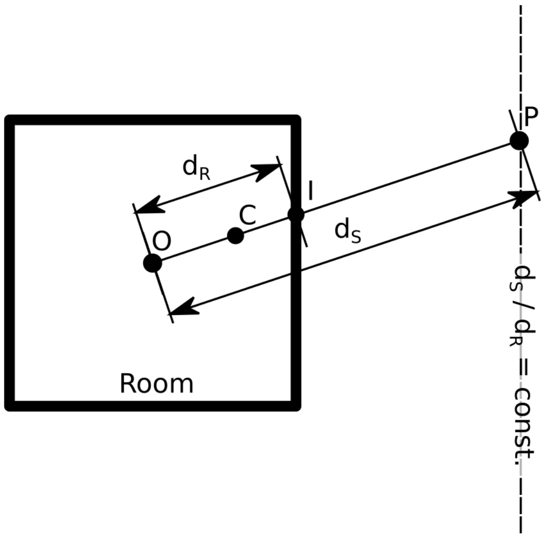


Figure 3: Projection of a room-anchored position to a position outside the room

The room anchored coordinate system is specified in clause 4.2.1.

5.2.1.3 Position in screen-anchored coordinates

The decoder shall set *anchor* = SCREEN.

The position $\mathbf{P}(x_{\text{Screen}}; y_{\text{Screen}}; z_{\text{Screen}})$ shall be determined as $\mathbf{P} = (\text{pos3D\_X}; \text{pos3D\_Y}; \text{pos3D\_Z})$

The decoder shall set *3D_coordinates* at the decoder interface to the three dimensional coordinates $(x_{\text{Screen}}; y_{\text{Screen}}; z_{\text{Screen}})$ of the position $\mathbf{P}$.

Position $\mathbf{P}$ together with *screen_factor* and *depth_factor* can be used to flexibly vary between a screen-anchored and a room-anchored position. A position $\mathbf{P}_{\text{interpolated}}$ can be determined as follows:

1) Let $\mathbf{C}_R = \mathbf{O}_S + \left(\frac{w_S}{\text{ref\_screen\_ratio}} \times X_{\text{screen}}; Y_{\text{screen}}; \frac{h_S}{2 \times \text{ref\_screen\_ratio}} \times (Z_{\text{screen}} + 1)\right)$ be the transformation of screen-anchored coordinates to the room-anchored system as specified in clause 4.2.2.

2) Let $\mathbf{C}_S = P$.

3) Let $\mathbf{M}_1 = \begin{vmatrix} \text{screen\_factor} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \text{screen\_factor} \end{vmatrix}$.

4) Let $\mathbf{M}_2 = \begin{vmatrix} y^{\text{depth\_factor}} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & y^{\text{depth\_factor}} \end{vmatrix}$.

5) Let $\mathbf{I} = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$ be the 3x3 identity matrix.

6) $\mathbf{P}_{\text{interpolated}} = \mathbf{M}_2 \times (\mathbf{M}_1 \times \mathbf{C}_S + (\mathbf{I} - \mathbf{M}_1) \times \mathbf{C}_R) + (\mathbf{I} - \mathbf{M}_2) \times \mathbf{C}_R$.

The screen-anchored coordinate system is specified in clause 4.2.2.

5.2.1.4 Position in speaker-anchored coordinates

The decoder shall set *anchor* = SPEAKER.

For a bed instance the coordinate of each contained object is indicated by the *bed_channel_assignment*.

The *bed_channel_assignment* is a list of speaker labels that indicate a speaker-anchored coordinate. Objects contained in the bed instance are mapped to the speaker labels in the same order. Each object has one coordinate, e.g. one speaker label.

EXAMPLE: For a bed instance containing the objects (*obj0*, *obj1*, *obj2*, *obj3*, *obj4*, *obj5*) the speaker-anchored coordinates are indicated by the *bed_channel_assignment* = (RC_L, RC_R, RC_C, RC_LFE, R_LSS, LC_RLS). In this case the coordinate of *obj0* is 'RC_L', the coordinate of *obj1* is 'RC_R', and so on.

For each object contained in a bed instance, the decoder shall set *speaker_lable* to the coordinate indicated by the corresponding *bed_channel_assignment*.

The speaker-anchored coordinate system is specified in clause 4.2.3.

5.2.2 Size

The size property is the data that is provided as *size* at the decoder interface.

The size property comprises the three values $(\mathrm{width}; \mathrm{depth}; \mathrm{height})$ that indicate the apparent 3-dimensional spatial extent of the object.

Width, depth and height can be controlled independently along the three axes x, y and z of the coordinate system and are bounded by $[0; 1]$, in units of the coordinate system's dimensions.

The size property shall be set as:

$$(\mathrm{width}; \mathrm{depth}; \mathrm{height}) = (\text{object\_width}; \text{object\_depth}; \text{object\_height}).$$

EXAMPLE 1: If all size properties are one, the object is as wide as the room.

EXAMPLE 2: If all size properties are zero, the object is a point source.

EXAMPLE 3: If two of the size properties are zero, it extends along a line.

EXAMPLE 4: If only one of the size properties is zero, it is a rectangle.

EXAMPLE 5: If all sizes are non-zero and equal, the object is a cube; otherwise it is a rectangular cuboid.

Table 2 lists the related bitstream elements and their reference.

Table 2: Object size related bitstream elements

Bitstream element	Reference
`object_size_idx`	Clause 5.6.1.2.1
`object_size_bits`	Clause 5.6.1.2.2
`object_width_bits`	Clause 5.6.1.2.3
`object_depth_bits`	Clause 5.6.1.2.4
`object_height_bits`	Clause 5.6.1.2.5

5.2.3 Priority

The priority property is the data that is provided as *priority* at the decoder interface.

The priority property shall be set to *object_priority*. It imposes an ordering by importance where higher priority indicates higher importance and is bounded by $[0; 1]$.

Table 3 lists the related bitstream elements and their reference.

Table 3: Object priority related bitstream elements

Bitstream element	Reference
`b_default_priority`	Clause 5.6.1.3.1
`object_priority_bits`	Clause 5.6.1.3.2

5.2.4 Gain

The gain property is the data that is provided as *gain* at the decoder interface.

The gain property value shall bet set to *object_gain*. The gain property can be used to apply a custom gain value to an object. It is bounded by $[-\infty\text{dB}; +15\text{dB}]$.

Table 4 lists the related bitstream elements and their reference.

Table 4: Object gain related bitstream elements

Bitstream element	Reference
`object_gain_idx`	Clause 5.6.1.4.1
`object_gain_bits`	Clause 5.6.1.4.2

5.2.5 Channel lock

The channel lock property is the data that is provided as *channel_lock* at the decoder interface.

Channel lock shall be set to `b_object_snap`. Channel lock is a Boolean property that can be used to constrain rendering of an object to a single speaker, providing a non-diffuse, timbre-neutral reproduction. True indicates that the object rendering is constrained to a single speaker; false indicates that panning may be used.

Table 5 lists the related bitstream element and its reference.

Table 5: Object snap related bitstream elements

Bitstream element	Reference
`b_object_snap`	Clause 5.6.1.5.1

5.2.6 Zone constraints

The zone constraints property is the data that is provided in *zone_constraints_update* at the decoder interface.

The list *zone_constraints* specifies zone constraints (include or exclude) for the following zones:

- Screen zone.
- Side zone.
- Surround zone.
- Back zone.
- Centre-and-back zone.
- Top-Bottom zone.

NOTE: Zones may be defined by a list of speakers as listed in table A.7, or by a sub-volume of the room, as specified in table B.19.

Based on the values of `zone_constraints_idx` and `b_enable_elevation`, the decoder shall create a *zone_constraints* list that indicates the zone constraints (include or exclude) for each zone.

EXAMPLE: If `zone_constraints_idx`=2 and `b_enable_elevation` is false, the side zone and the Top-Bottom zone are excluded. The *zone_constraints* list for this case is listed in table 6.

Table 6: zone_constraints list when `zone_constraints_idx`=2 and `b_enable_elevation` is false

Zone	Zone constraints
Screen zone	include
Side zone	exclude
Surround zone	include
Back zone	include
Centre-and-back zone	include
Top-Bottom zone	exclude

Table 7 lists the related bitstream elements and their reference.

Table 7: Object zone constraints related bitstream elements

Bitstream element	Reference
`zone_constraints_idx`	Clause 5.6.1.6.1
`b_enable_elevation`	Clause 5.6.1.6.2

5.3 Property update information

5.3.1 Introduction

The Property Update Information contains the data that is provided as *timing_data* at the decoder interface.

The OAMD bitstream supports multiple updates of the object properties per codec frame. The bitstream contains timing information corresponding to these updates. In the context of this clause, *previous* and *subsequent* refer to the temporal dimension (i.e. the "subsequent update" is an update for the same object, at a later time).

EXAMPLE: Three sequential update blocks *previous*, *current* and *subsequent* have the start times: $\text{start\_time}_{\text{previous}} < \text{start\_time}_{\text{current}} < \text{start\_time}_{\text{subsequent}}$.

5.3.2 Timing of property updates

The timing of one transmitted property update specifies its start time, along with that its context with preceding or subsequent updates and the temporal duration for an interpolation process between successive updates.

The OAMD bitstream syntax supports up to eight property updates per object in each codec frame. The timing of one update is identical for all objects, i.e. the number of signalled updates or the start and stop time of each property update is identical for all objects. The metadata contained in the `md_update_info` block carries the signalled timing data applicable to all updates for all transmitted objects. The contained instances of the `block_update_info` block contain metadata that signals the timing data of the corresponding specific update.

In the following paragraphs the *ramp_duration* value for smoothing is introduced first, the values of *sample_offset* and *block_offset_factor* are introduced afterward for calculating the value of *start_sample*. Finally the calculation of *frame_offset* is specified.

The metadata contained in the `md_update_info` block indicates the value of *ramp_duration* that specifies a time period in audio samples for an interpolation from signalled object property values of the previous property update to values of the current update.

Figure 4 shows the ramp duration for three sequential property updates *n*, *n+1* and *n+2* for one object. The three property updates have the *start_sample* values t_n, t_{n+1} and t_{n+2}. The corresponding *ramp_duration* values are rd_n, rd_{n+1} and rd_{n+2}. The points of time md_n, md_{n+1} and md_{n+2} are the end points of the interpolation process where the signalled values for the object properties are reached after the time period of *ramp_duration*.

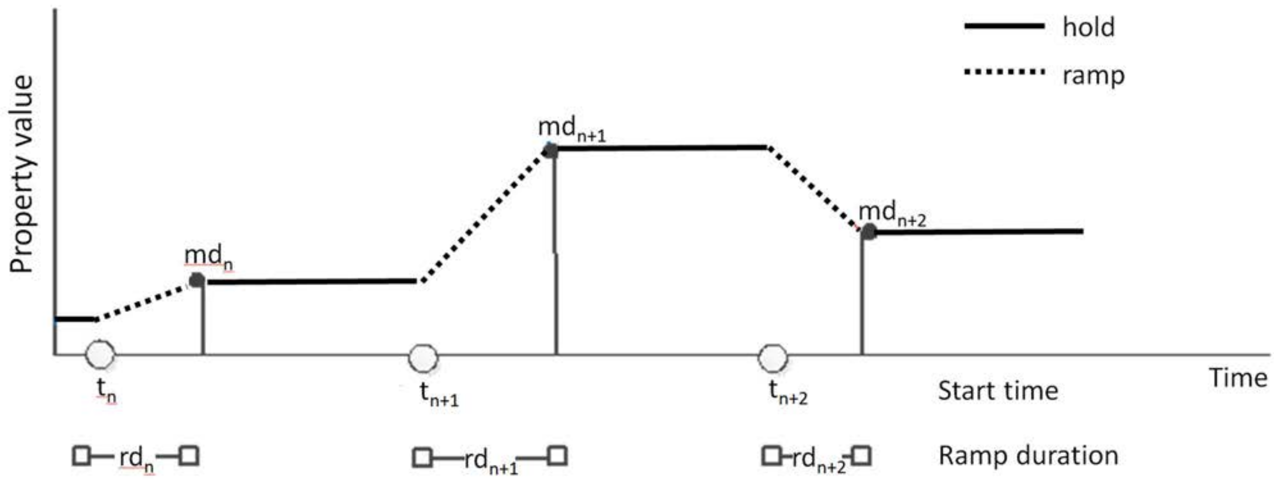


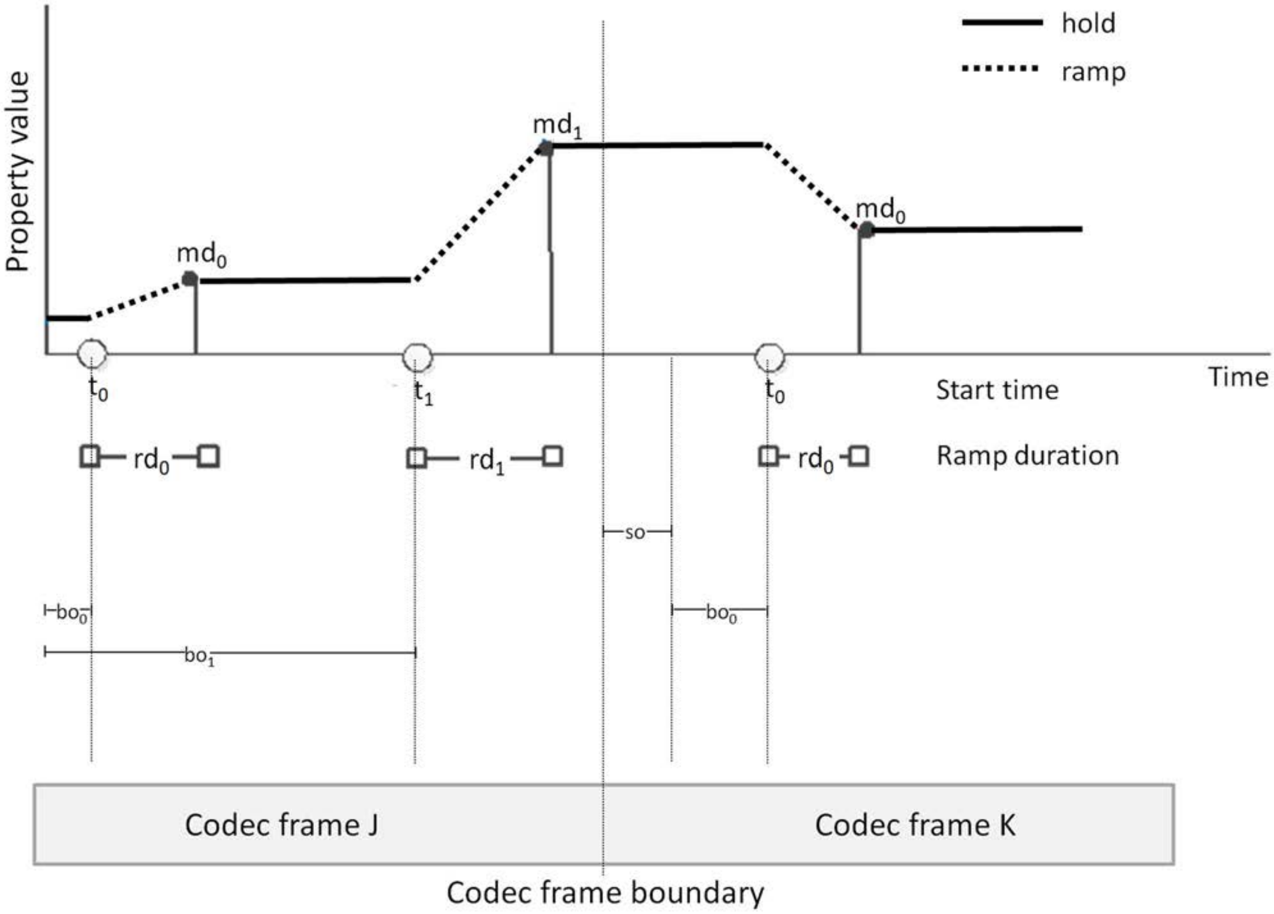
Figure 4: Temporal context of the ramp duration for three sequential property updates

For each property update the timing metadata contains one value for *block_offset_factor* that indicates a time period in samples as offset from a time point *sample_offset* common for all property updates. The value of *sample_offset* is a temporal offset in samples to the first PCM audio sample that the data in the OAMD payload applies to, as specified in ETSI TS 102 366 [1], clauses H.2.2.3.1 and H.2.2.3.2.

Figure 5 shows the temporal context of the start time for two sequential property updates md_0 and md_1 in a codec frame *J* and one property update md_0 in the following codec frame *K*. The *start_sample* values t_n shall be determined by the following equation:

$$t_n = \text{so} + 32 \times \text{bo}_n.$$

In this equation *so* is the value of *sample_offset* and bo_n is the value of *block_offset_factor* for the corresponding property update.



NOTE: For codec frame *J* the value *so* for *sample_offset* is zero.

Figure 5: Temporal context of the start time for three sequential property updates

In the processing of each codec frame the decoder shall add the value 1 536 (the codec transform size in samples) to *frame_offset*.

Table 8 lists the related bitstream elements and their reference.

Table 8: Property update timing related bitstream elements

Bitstream element	Reference
`sample_offset_code`	Clause 5.6.2.1
`sample_offset_idx`	Clause 5.6.2.2
`num_obj_info_blocks_bits`	Clause 5.6.2.4
`block_offset_factor_bits`	Clause 5.6.2.5
`ramp_duration_code`	Clause 5.6.2.5
`b_use_ramp_duration_idx`	Clause 5.6.2.6
`ramp_duration_idx`	Clause 5.6.2.8
`ramp_duration_bits`	Clause 5.6.2.9

5.4 Object audio metadata structure

The OAMD bitstream contains metadata updates. Each update contains the metadata is updated; timing information is shared between objects and updates.

Figure 6 shows the OAMD bitstream structure.

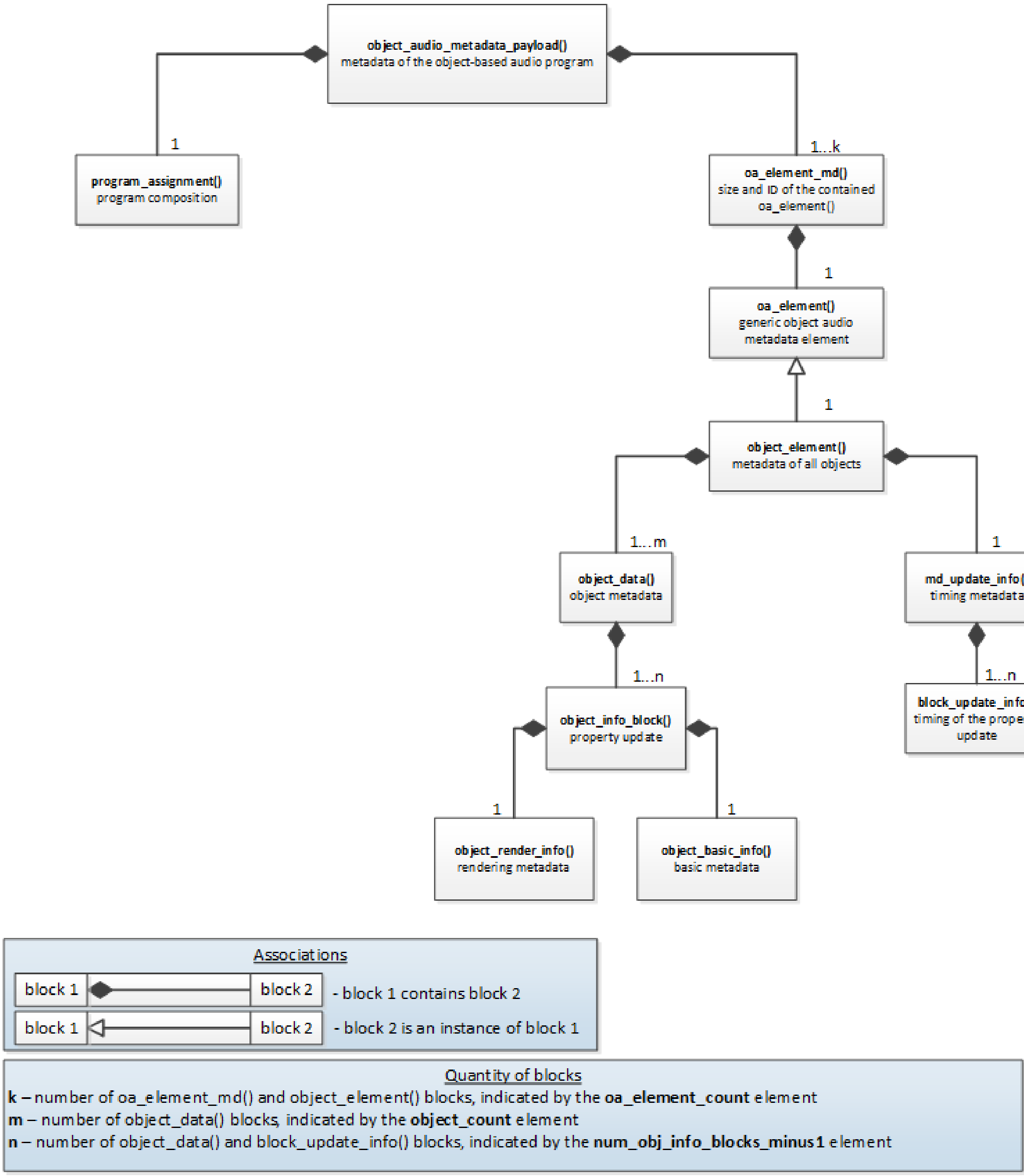


Figure 6: OAMD bitstream structure

Table 9 lists the blocks contained in the OAMD bitstream.

Table 9: OAMD bitstream blocks

Block	Contents	See also
`object_audio_metadata_payload`	The OAMD payload (this is the top level block)	
`program_assignment`	Metadata indicating the OAMD content description	
`oa_element_md`	Metadata indicating the size and identification of the `oa_element`	
`oa_element`	A generic object-based audio metadata element	
`object_element`	Property updates for all objects	
`object_data`	All property updates for one object	
`md_update_info`	Sample offset information common to all contained `block_update_info` elements	Clause 5.3
`block_update_info`	Timing information for each property update	Clause 5.3
`object_info_block`	One property update for one object	Clause 5.3
`object_basic_info`	Gain and priority	Clauses 5.2.4 and 5.2.3
`object_render_info`	Position, size, channel lock and zone constraints	Clauses 5.2.1, 5.2.2, 5.2.5 and 5.2.6

5.5 Bitstream syntax

5.5.1 variable_bits_max

```
Syntax                                                                        No of bits
variable_bits_max(n, max_num_groups)
{
  value = 0;
  num_group = 1;
  read; ....................................................................... n
  value += read;
  b_read_more; ................................................................ 1
  if (max_num_groups > num_group) {
    if (b_read_more) {
      value <<= n;
      value += (1 << n);
    }
    while (b_read_more) {
  read; ....................................................................... n
      value += read;
      b_read_more; ............................................................ 1
      if (num_group >= max_num_groups) {
        break();
      }
      if (b_read_more) {
        value <<= n;
        value += (l << n);
        num_group += 1;
      }
    }
  }
  return value;
}
```

5.5.2 object_audio_metadata_payload

Syntax	No of bits
object_audio_metadata_payload()	
{	
oa_md_version_bits;	2
if (oa_md_version_bits == 0x3) {	
oa_md_version_bits_ext;	3
oa_md_version_bits += oa_md_version_bits_ext;	
}	
object_count_bits;	5
if (object_count_bits == 0x1F) {	
object_count_bits_ext;	7
object_count_bits += object_count_bits_ext;	
}	
program_assignment();	
b_alternate_object_data_present;	1
oa_element_count_bits;	4
if (oa_element_count_bits == 0xF) {	
oa_element_count_bits_ext;	5
oa_element_count_bits += oa_element_count_bits_ext;	
}	
for (i = 0; i < oa_element_count_bits; i++) {	
oa_element_md();	
}	
padding;	VAR
}	

5.5.3 program_assignment

Syntax | **No of bits**

```
program_assignment()
{
  b_dyn_object_only_program; ........................................ 1
  if (b_dyn_object_only_program) {
    b_lfe_present; .................................................. 1
  }
  else {
    content_description_mask; ....................................... 4
    if (content_description_mask & 0x1) {
      b_bed_object_chan_distribute; ................................. 1
      b_multiple_bed_instances_present; ............................. 1
      if (b_multiple_bed_instances_present) {
        num_bed_instances_bits; ..................................... 3
      }
      for (beds = 0; beds < num_bed_instances; beds++) {
        b_lfe_only; ................................................. 1
        if (!b_lfe_only) {
          b_standard_chan_assign; ................................... 1
          if (b_standard_chan_assign) {
            bed_channel_assignment_mask; ........................... 10
          }
          else {
            nonstd_bed_channel_assignment_mask; .................... 17
          }
        }
      }
    }
    if (content_description_mask & 0x2) {
      intermediate_spatial_format_idx; .............................. 3
    }
    if (content_description_mask & 0x4) {
      num_dynamic_objects_bits; ..................................... 5
      if (num_dynamic_objects_bits == 0x1F) {
        num_dynamic_objects_bits_ext; ............................... 7
        num_dynamic_objects_bits += num_dynamic_objects_bits_ext;
      }
    }
    if (content_description_mask & 0x8) {
      reserved_data_size_bits; ...................................... 4
      reserved_data();
      padding; ...................................................... VAR
    }
  }
}
```

5.5.4 oa_element_md

Syntax | **No of bits**

```
oa_element_md()
{
  oa_element_id_idx; ................................................ 4
  oa_element_size_bits = variable_bits_max(4,4);
  if (b_alternate_object_data_present) {
    alternate_object_data_id_idx; ................................... 4
  }
  b_discard_unknown_element; ........................................ 1
  oa_element();
  padding; .......................................................... VAR
}
```

5.5.5 object_element

Syntax	No of bits
object_element()	
{	
md_update_info();	
b_default_screen_size_ratio;	1
if (!b_default_screen_size_ratio) {	
ref_screen_ratio_bits;	5
}	
for (j = 0; j < object_count; j++) {	
object_data();	
}	
}	

5.5.6 md_update_info

Syntax	No of bits
md_update_info()	
{	
sample_offset_code;	2
if (sample_offset_code == 0b01) {	
sample_offset_idx;	2
}	
else {	
if (sample_offset_code == 0b10) {	
sample_offset_bits;	5
}	
}	
num_obj_info_blocks_bits;	3
for (blk = 0; blk < num_obj_info_blocks; blk++) {	
block_update_info(blk);	
}	
}	

5.5.7 block_update_info

Syntax	No of bits
block_update_info()	
{	
block_offset_factor_bits;	6
ramp_duration_code;	2
if (ramp_duration_code == 0b11) {	
b_use_ramp_duration_idx;	1
if (b_use_ramp_duration_idx) {	
ramp_duration_idx;	4
}	
else {	
ramp_duration_bits;	11
}	
}	
}	

5.5.8 object_data

Syntax	No of bits
object_data()	
{	
for (blk = 0; blk < num_obj_info_blocks; blk++) {	
object_info_block(blk);	
}	
}	

5.5.9 object_info_block

Syntax	No of bits
object_info_block(blk)	
{	
b_object_not_active;	1
if (b_object_not_active) {	
object_basic_info_status_idx = 0b00;	
}	
else {	
if (blk == 0) {	
object_basic_info_status_idx = 0b01;	
}	
else {	
object_basic_info_status_idx;	2
}	
}	
if ((object_basic_info_status_idx == 0b01) || (object_basic_info_status_idx == 0b11)) {	
object_basic_info();	
}	
if (b_object_not_active) {	
object_render_info_status_idx = 0b00;	
}	
else {	
if (!b_object_in_bed_or_isf) {	
if (blk == 0) {	
object_render_info_status_idx = 0b01;	
}	
else {	
object_render_info_status_idx;	2
}	
}	
else {	
object_render_info_status_idx = 0b00;	
}	
}	
if ((object_render_info_status_idx == 0b01) || (object_render_info_status_idx == 0b11)) {	
object_render_info();	
}	
b_additional_table_data_exists;	1
if (b_additional_table_data_exists) {	
additional_table_data_size_bits;	4
additional_table_data();	
padding;	VAR
}	
}	

5.5.10 object_basic_info

Syntax	No of bits
object_basic_info()	
{	
if (object_basic_info_status_idx == 0b01) {	
object_basic_info_mask = 0b11;	
}	
else {	
object_basic_info_mask;	2
}	
if (object_basic_info_mask & 0x1) {	
object_gain_idx;	2
if (object_gain_idx == 0b10) {	
object_gain_bits;	6
}	
}	
if (object_basic_info_mask & 0x2) {	
b_default_object_priority;	1
if (!b_default_object_priority) {	
object_priority_bits;	5
}	
}	
}	

5.5.11 object_render_info

```
Syntax                                                              No of bits
object_render_info()
{
  if (object_render_info_status_idx == 0b01) {
    obj_render_info_mask = 0b1111;
  }
  else {
    obj_render_info_mask; ............................................... 4
  }
  if (obj_render_info_mask & 0x1) {
    if (blk == 0) {
      b_differential_position_specified = FALSE;
    }
    else {
      b_differential_position_specified; ................................ 1
    }
    if (b_differential_position_specified) {
      diff_pos3D_X_bits; ................................................ 3
      diff_pos3D_Y_bits; ................................................ 3
      diff_pos3D_Z_bits; ................................................ 3
    }
    else {
      pos3D_X_bits; ..................................................... 6
      pos3D_Y_bits; ..................................................... 6
      pos3D_Z_sign_bits; ................................................ 1
      pos3D_Z_bits; ..................................................... 4
    }
    b_object_distance_specified; ........................................ 1
    if (b_object_distance_sepcified) {
      b_object_at_infinity; ............................................. 1
      if (b_object_at_infinity) {
        object_distance = inf;
      }
      else {
        distance_factor_idx; ............................................ 4
      }
    }
  }
  if (obj_render_info_mask & 0x2) {
    zone_constraints_idx; ............................................... 3
    b_enable_elevation; ................................................. 1
  }
  if (obj_render_info_mask & 0x4) {
    object_size_idx; .................................................... 2
    if (object_size_idx == 0b01) {
      object_size_bits; ................................................. 5
    }
    else {
      if (object_size_idx == 0b10) {
        object_width_bits; .............................................. 5
        object_depth_bits; .............................................. 5
        object_height_bits; ............................................. 5
      }
    }
  }
  if (obj_render_info_mask & 0x8) {
    b_object_use_screen_ref; ............................................ 1
    if (b_object_use_screen_ref) {
      screen_factor_bits; ............................................... 3
      depth_factor_idx; ................................................. 2
    }
    else {
      screen_factor_bits = 0;
    }
  }
  b_object_snap; ........................................................ 1
}
```

5.6 Bitstream description

5.6.1 Object properties

5.6.1.1 Position

5.6.1.1.1 b_default_screen_size_ratio

The `b_default_screen_size_ratio` flag indicates whether *ref_screen_ratio* shall default to the value of 1,0. If `b_default_screen_size_ratio` is false, the value of *ref_screen_ratio* is indicated by the `ref_screen_ratio_bits` element.

5.6.1.1.2 ref_screen_ratio_bits

The `ref_screen_ratio_bits` codeword indicates the value of *ref_screen_ratio*.

The value of *ref_screen_ratio* indicates the ratio of the reference screen width to the distance between the L and R speakers in the mastering studio.

The value of *ref_screen_ratio* shall be determined by the following equation:

$$\text{ref\_screen\_ratio} = {}^{(\text{ref\_screen\_ratio\_bits}+1)}\!/_{33}$$

In this equation, the bits of the `ref_screen_ratio_bits` element are interpreted as an unsigned integer.

5.6.1.1.3 b_standard_chan_assign

The `b_standard_chan_assign` flag indicates whether the *bed_channel_assignment* is indicated by the `bed_channel_assignment_mask` element (if true) or by the `nonstd_bed_channel_assignment_mask` element (when false).

5.6.1.1.4 bed_channel_assignment_mask

The `bed_channel_assignment_mask` codeword indicates the *bed_channel_assignment*.

The *bed_channel_assignment* is a list of speaker labels. Each bit of the `bed_channel_assignment_mask` element is a flag that indicates whether the corresponding speaker labels or pair of speaker labels shall be contained in the *bed_channel_assignment*.

Table 10 lists how bits of the `bed_channel_assignment_mask` element relate to speaker labels.

Speaker labels in the *bed_channel_assignment* shall be ordered like the `bed_channel_assignment_mask` bits, starting with bit 0. If the number of speaker labels is 2, the speaker labels shall be ordered as shown in column two.

Table 10: Speaker labels contained in the *bed_channel_assignment* indicated by the `bed_channel_assignment_mask` element

`bed_channel_assignment_mask` bit	Speaker label(s)	Number of speaker labels
0	RC_L/RC_R	2
1	RC_C	1
2	RC_LFE	1
3	RC_LSS/RC_RSS	2
4	RC_LRS/RC_RRS	2
5	RC_LFH/RC_RFH	2
6	RC_LTM/RC_RTM	2
7	RC_LRH/RC_RRH	2
8	RC_LW/RC_RW	2
9	RC_LFE2	1

5.6.1.1.5 nonstd_bed_channel_assignment_mask

The `nonstd_bed_channel_assignment_mask` codeword indicates the *bed_channel_assignment*.

The *bed_channel_assignment* is a list of speaker labels. Each bit of the `nonstd_bed_channel_assignment_mask` element is a flag that indicates whether the corresponding speaker label shall be contained in the *bed_channel_assignment*.

Table 11 lists how bits of the `bed_channel_assignment_mask` element relate to speaker labels.

Speaker labels in the *bed_channel_assignment* shall be ordered like the `bed_channel_assignment_mask` bits, starting with bit 0.

Table 11: Speaker labels contained in the *bed_channel_assignment* indicated by the `nonstd_bed_channel_assignment_mask` element

`non_std_channel_assignment` bit	Speaker label(s)
0	RC_L
1	RC_R
2	RC_C
3	RC_LFE
4	RC_LSS
5	RC_RSS
6	RC_LRS
7	RC_RRS
8	RC_LFH
9	RC_RFH
10	RC_LTM
11	RC_RTM
12	RC_LRH
13	RC_RRH
14	RC_LW
15	RC_RW
16	RC_LFE2

5.6.1.1.6 b_lfe_only

The `b_lfe_only` flag indicates whether the *bed_channel_assignment* contains one LFE channel and no other channels.

5.6.1.1.7 b_differential_position_specified

The `b_differential_position_specified` flag indicates how the position values *pos3D_X*, *pos3D_Y* and *pos3D_Z* are signalled in the bitstream.

Table 12 specifies how the values of *pos3D_X*, *pos3D_Y* and *pos3D_Z* are signalled in the bitstream.

Table 12: Signalling of *pos3D_X*, *pos3D_Y* and *pos3D_Z*

`b_differential_position_specified`	signalling
0	*pos3D_X*, *pos3D_Y* and *pos3D_Z* are indicated by `pos3D_X_bits`, `pos3D_Y_bits`, `pos3D_Z_sign_bits` and `pos3D_Z_bits` (no differential coding)
1	*pos3D_X*, *pos3D_Y* and *pos3D_Z* are indicated by `diff_pos3D_X_bits`, `diff_pos3D_Y_bits` and `diff_pos3D_Z_bits` (differential coding)

5.6.1.1.8 pos3D_X_bits

The `pos3D_X_bits` codeword indicates the value of *pos3D_X*.

pos3D_X indicates the x-coordinate value for the position of the object.

The value of *pos3D_X* shall be determined by the following equation:

$$\mathrm{pos3D\_X} = \min\left(1; {}^{\mathrm{pos3D\_X\_bits}}/_{62}\right).$$

In this equation, the bits of `pos3D_X_bits` are interpreted as an unsigned integer.

5.6.1.1.9 pos3D_Y_bits

The `pos3D_Y_bits` codeword indicates the value of *pos3D_Y*.

pos3D_Y indicates the y-coordinate value for the position of the object.

The value of *pos3D_Y* shall be determined by the following equation:

$$\mathrm{pos3D\_Y} = \min\left(1; {}^{\mathrm{pos3D\_Y\_bits}}/_{62}\right),$$

where the bits of `pos3D_Y_bits` are interpreted as an unsigned integer.

5.6.1.1.10 pos3D_Z_sign_bits

The `pos3D_Z_sign_bits` codeword indicates the value of *pos3D_Z_sign*.

pos3D_Z_sign is used to determine whether the z-coordinate for the position of the object is positive or negative.

The *pos3D_Z_sign* value shall be determined by the following equation:

$$\mathrm{pos3D\_Z\_sign} = \begin{cases} -1, & when \quad \mathrm{pos3D\_Z\_sign\_bits} = 0 \\ +1, & else \end{cases}$$

5.6.1.1.11 pos3D_Z_bits

The `pos3D_Z_bits` codeword indicates the value of *pos3D_Z*.

pos3D_Z indicates the z-coordinate value for the position of the object.

The value of *pos3D_Z* shall be determined by the following equation:

$$\mathrm{pos3D\_Z} = \mathrm{pos3D\_Z\_sign} \times {}^{\mathrm{pos3D\_Z\_bits}}/_{15}$$

The bits of `pos3D_Z_bits` are interpreted as an unsigned integer and the value of *pos3D_Z_sign* is specified in clause 5.6.1.1.10.

5.6.1.1.12 diff_pos3D_X_bits

The `diff_pos3D_X_bits` codeword indicates the value of *pos3D_X*, taking into account the previous value of *pos3D_X* (differential coding).

The *pos3D_X* value shall be determined by the following equation:

$$\mathrm{pos3D\_X} = \max\left(0; \min\left(1; \mathrm{pos3D\_X}_{\mathrm{prev}} + \left({}^{\mathrm{diff\_pos3D\_X\_bits}}/_{62}\right)\right)\right)$$

In this equation, the bits of `diff_pos3D_X_bits` are interpreted as a signed integer.

The $pos3D\_X_{prev}$ value is the *pos3D_X* value determined by decoding of the previous metadata update block.

NOTE: The previous metadata update block is defined in clause 5.3.

5.6.1.1.13 diff_pos3D_Y_bits

The `diff_pos3D_Y_bits` codeword indicates the value of *pos3D_Y*, taking into account the previous value of *pos3D_Y* (differential coding).

The *pos3D_Y* value shall be determined by the following equation:

$$\mathrm{pos3D\_Y} = \max\left(0; \min\left(1; \mathrm{pos3D\_Y}_{\mathrm{prev}} + \left(\mathrm{diff\_pos3D\_Y\_bits}/_{62}\right)\right)\right).$$

In this equation, the bits of `diff_pos3D_Y_bits` are interpreted as a signed integer.

The *pos3D_Y*$_{\mathrm{prev}}$ value is the *pos3D_Y* value determined by decoding of the previous metadata update block.

NOTE: The previous metadata update block is defined in clause 5.3.

5.6.1.1.14 diff_pos3D_Z_bits

The `diff_pos3D_Z_bits` codeword indicates the value of *pos3D_Z*, taking into account the previous value of *pos3D_Z* (differential coding).

The *pos3D_Z* value shall be determined by the following equation:

$$\mathrm{pos3D\_Z} = \max\left(-1; \min\left(1; \mathrm{pos3D\_Z}_{\mathrm{prev}} + \left(\mathrm{diff\_pos3D\_Z\_bits}/_{15}\right)\right)\right).$$

In this equation, the bits of `diff_pos3D_Z_bits` are interpreted as a signed integer.

The *pos3D_Z*$_{\mathrm{prev}}$ value is the *pos3D_Z* value determined by decoding of the previous metadata update block.

NOTE: The previous metadata update block is defined in clause 5.3.

5.6.1.1.15 b_object_distance_specified

The `b_object_distance_specified` flag indicates whether the position of an object is outside of the room boundaries.

5.6.1.1.16 b_object_at_infinity

The `b_object_at_infinity` flag indicates whether the object distance from the listening position is equal to infinity (when true) or is indicated by the `distance_factor_idx` element (when false).

5.6.1.1.17 distance_factor_idx

The `distance_factor_idx` index indicates the value of *distance_factor*.

The value of *distance_factor* shall be determined as specified in table 13.

Table 13: *distance_factor* value indicated by the `distance_factor_idx` element

`distance_factor_idx`	*distance_factor*
0	1,1
1	1,3
2	1,6
3	2,0
4	2,5
5	3,2
6	4,0
7	5,0
8	6,3
9	7,9
10	10,0
11	12,6
12	15,8
13	20,0
14	25,1
15	50,1

5.6.1.1.18 b_object_use_screen_ref

The `b_object_use_screen_ref` flag indicates whether the signalled values of *pos3D_X*, *pos3D_Y* and *pos3D_Z* are related to the screen (when true) or to the room (when false).

If the `b_object_use_screen_ref` flag is true, the `screen_factor_bits` element and the `depth_factor_idx` element follow in the bitstream. If the `b_object_use_screen_ref` flag is false, *screen_factor* = 0.

5.6.1.1.19 screen_factor_bits

The `screen_factor_bits` unsigned integer indicates the value of *screen_factor*.

The value of *screen_factor* shall be determined by the following equation:

$$\text{screen\_factor} = {}^{(\text{screen\_factor\_bits}+1)}/_{8}.$$

5.6.1.1.20 depth_factor_idx

The `depth_factor_idx` index indicates the value of *depth_factor*.

The value of the *depth_factor* shall be determined as specified in table 14.

Table 14: The value of *depth_factor* indicated by the `depth_factor_idx` element

`depth_factor_idx`	*depth_factor*
0	0,25
1	0,5
2	1
3	2

5.6.1.2 Size

5.6.1.2.1 object_size_idx

The `object_size_idx` index indicates the values of *object_width*, *object_depth* and *object_height*.

Table 15 lists the values of *object_width*, *object_depth*, and *object_height*.

Table 15: Values of *object_width*, *object_depth* and *object_height*

object_size_idx	*object_width*	*object_depth*	*object_height*
0	0	0	0
1	$\frac{\text{object\_size\_bits}}{31}$	$\frac{\text{object\_size\_bits}}{31}$	$\frac{\text{object\_size\_bits}}{31}$
2	$\frac{\text{object\_width\_bits}}{31}$	$\frac{\text{object\_depth\_bits}}{31}$	$\frac{\text{object\_height\_bits}}{31}$
3	Reserved		

5.6.1.2.2 object_size_bits

The `object_size_bits` codeword indicates the value of *object_width*, *object_depth* and *object_height*.

The values shall be determined by the following equation:

$$\text{object\_width} = \text{object\_depth} = \text{object\_height} = {}^{\text{object\_size\_bits}}\!/_{31}.$$

In this equation, the bits of `object_size_bits` are interpreted as an unsigned integer.

5.6.1.2.3 object_width_bits

The `object_width_bits` codeword indicates the value of *object_width*.

The value of *object_width* shall be determined by the following equation:

$$\text{object\_width} = {}^{\text{object\_width\_bits}}\!/_{31}.$$

In this equation, the bits of `object_width_bits` are interpreted as an unsigned integer.

5.6.1.2.4 object_depth_bits

The `object_depth_bits` codeword indicates the value of *object_depth*.

The value of *object_depth* shall be determined by the following equation:

$$\text{object\_depth} = {}^{\text{object\_depth\_bits}}\!/_{31}.$$

In this equation, the bits of `object_depth_bits` are interpreted as an unsigned integer.

5.6.1.2.5 object_height_bits

The `object_height_bits` codeword indicates the value of *object_height*.

The value of *object_height* shall be determined by the following equation:

$$\text{object\_height} = {}^{\text{object\_height\_bits}}\!/_{31}.$$

In this equation, the bits of `object_height_bits` are interpreted as an unsigned integer.

5.6.1.3 Priority

5.6.1.3.1 b_default_object_priority

The `b_default_object_priority` flag indicates whether the value of *object_priority* shall default to 1,0 (when true) or is indicated by the `object_priority_bits` element (when false).

The value of *object_priority* is in [0; 1], where *object_priority* in [0; 1[is signalled as specified in clause 5.6.1.3.2.

5.6.1.3.2 object_priority_bits

The object_priority_bits codeword indicates the value of the *object_priority*.

The value of *object_priority* shall be determined by the following equation:

$$\text{object\_priority} = {}^{\text{object\_priority\_bits}}/_{32}.$$

In this equation, the bits of object_priority_bits are interpreted as an unsigned integer. The value of *object_priority* is in [0; 1], where *object_priority* = 1 is signalled as specified in clause 5.6.1.3.1.

5.6.1.4 Gain

5.6.1.4.1 object_gain_idx

The object_gain_idx index indicates the value of *object_gain*.

object_gain shall be derived from object_gain_idx as specified in table 16.

Table 16: Value of *object_gain*

object_gain_idx	*object_gain*
0	0 dB
1	-∞ dB
2	indicated by the object_gain_bits element
3	*object_gain* of the previous object in this metadata update block, or a default of 0 dB if the current object is the first object

5.6.1.4.2 object_gain_bits

The object_gain_bits unsigned integer indicates the value of *object_gain*.

The value of *object_gain* shall be determined as specified in table 17.

Table 17: Determination of *object_gain* indicated by object_gain_bits

object_gain_bits	*object_gain* [dB]
0-14	15 – object_gain_bits; *object_gain* is in [1; 15]
15-63	14 – object_gain_bits; *object_gain* is in [-49; -1]

NOTE: The value *object_gain* = 0 dB can't be signalled by object_gain_bits, but can be signalled by object_gain_idx = 0.

5.6.1.5 Channel lock

5.6.1.5.1 b_object_snap

The b_object_snap flag indicates the channel lock property.

5.6.1.6 Zone constraints

5.6.1.6.1 zone_constraints_idx

The zone_constraints_idx index indicates zone constraints for the horizontal zones.

Horizontal zones are marked to be included or excluded as listed in table 18.

NOTE: Constraints for the Top-Bottom zone are documented in clause 5.6.1.6.2.

Table 18: Zone constraints indicated by the `zone_constraints_idx` element

zone_constraints_idx	Zone constraints
0	No constraints
1	Back zone excluded (all other horizontal zones included)
2	Side zone excluded (all other horizontal zones included)
3	Centre-and-back zone included (all other horizontal zones excluded)
4	Only screen zone included (all other horizontal zones excluded)
5	Only surround zone included (all other horizontal zones excluded)
6-7	Reserved

5.6.1.6.2 b_enable_elevation

The `b_enable_elevation` flag indicates the zone constraints for the Top-Bottom zone.

The Top-Bottom zone is marked to be included or excluded as listed in table 19.

Table 19: Top-Bottom zone constraints indicated by `b_enable_elevation`

b_enable_elevation	Top-Bottom zone
true	included
false	excluded

5.6.2 Timing metadata

5.6.2.1 sample_offset_code

The `sample_offset_code` index indicates the value of *sample_offset*.

Table 20 lists the value of *sample_offset* indicated by the `sample_offset_code` element.

Table 20: Signalling of *sample_offset*

`sample_offset_code`	*sample_offset*
0	0
1	indicated by the following `sample_offset_idx` element
2	indicated by the following `sample_offset_bits` element
3	Reserved

5.6.2.2 sample_offset_idx

The `sample_offset_idx` index indicates the value of *sample_offset*.

Table 21 lists the value of *sample_offset* indicated by the `sample_offset_idx` element.

Table 21: Value of *sample_offset*

`sample_offset_idx`	*sample_offset* (in samples)
0	8
1	16
2	18
3	24

5.6.2.3 sample_offset_bits

The `sample_offset_bits` unsigned integer determines the value of *sample_offset*.

sample_offset is in a range [0; 31].

5.6.2.4 num_obj_info_blocks_bits

The num_obj_info_blocks_bits codeword indicates the value of *num_obj_info_blocks*.

The *num_obj_info_blocks* value is determined by the following equation:

$$\text{num\_obj\_info\_blocks} = \text{num\_obj\_info\_blocks\_bits} + 1.$$

In this equation, the bits of num_obj_info_blocks_bits are interpreted as an unsigned integer.

5.6.2.5 block_offset_factor_bits

The block_offset_factor_bits unsigned integer indicates the *block_offset_factor* value.

5.6.2.6 ramp_duration_code

The ramp_duration_code index indicates the value of *ramp_duration*.

Table 22 lists the value of *ramp_duration* indicated by the ramp_duration_code element:

Table 22: Value of *ramp_duration*

ramp_duration_code	*ramp_duration*
0	0
1	512
2	1 536
3	indicated by the b_use_ramp_duration_idx element

5.6.2.7 b_use_ramp_duration_idx

The b_use_ramp_duration_idx flag indicates whether the value of *ramp_duration* is indicated by the ramp_duration_idx element (when true) or by the ramp_duration_bits element (when false).

5.6.2.8 ramp_duration_idx

The ramp_duration_idx index indicates the value of *ramp_duration*.

Table 23 lists the value of *ramp_duration* indicated by the corresponding table index.

Table 23: Value of *ramp_duration*

ramp_duration_idx	***ramp_duration*** **(in samples)**
0	32
1	64
2	128
3	256
4	320
5	480
6	1 000
7	1 001
8	1 024
9	1 600
10	1 601
11	1 602
12	1 920
13	2 000
14	2 002
15	2 048

5.6.2.9 ramp_duration_bits

The `ramp_duration_bits` unsigned integer determines the value of *ramp_duration*.

ramp_duration is in the range of [0; 2 047].

5.6.3 Object audio metadata content description

5.6.3.1 b_dyn_object_only_program

The `b_dyn_object_only_program` flag indicates whether the bitstream contains one or more dynamic objects plus an optional object positioned at the LFE speaker (when true), or it contains a combination of one or more bed instances, intermediate spatial format and one or more dynamic objects (when false).

5.6.3.2 b_lfe_present

The `b_lfe_present` flag indicates whether the bitstream contains an object positioned at the LFE speaker.

5.6.3.3 content_description_mask

The `content_description_mask` codeword indicates the object types present in the bitstream.

Each bit of the `content_description_mask` element is a flag that indicates whether one or more objects of the assigned object type are present in the bitstream. Table 24 lists the object types for the `content_description_mask` bits.

Table 24: Object types for `content_description_mask` bits

`content_description_mask` bit	assigned object type
0	object(s) with speaker-anchored coordinate(s) (one or more bed instances)
1	intermediate spatial format (ISF)
2	object(s) with room-anchored or screen-anchored coordinates
3	reserved

5.6.3.4 b_bed_object_chan_distribute

The `b_bed_object_chan_distribute` flag indicates whether an object with a speaker-anchored position is intended to be spread to multiple appropiate loudspeakers.

5.6.3.5 b_multiple_bed_instances_present

The `b_multiple_bed_instances_present` flag indicates whether the bitstream contains multiple bed instances.

If the `b_multiple_bed_instances_present` flag is false, it indicates that *num_bed_instances* = 1.

5.6.3.6 num_bed_instances_bits

The `num_bed_instances_bits` codeword indicates the value of *num_bed_instances*.

The value of *num_bed_instances* is determined by the following equation:

$$\text{num\_bed\_instances} = \text{num\_bed\_instances\_bits} + 2.$$

In this equation, the bits of `num_bed_instances_bits` are interpreted as an unsigned integer.

5.6.3.7 intermediate_spatial_format_idx

The `intermediate_spatial_format_idx` index indicates the intermediate spatial format (ISF) type of the associated objects.

The ISF type indicated by `intermediate_spatial_format_idx` is shown in table 25.

Table 25: ISF

intermediate_spatial_format_idx	ISF type (SR notation)	objects present (MULZ order)	Number of objects
0	SR3.1.0.0	M1 M2 M3 U1	4
1	SR5.3.0.0	M1 M2 M3 M4 M5 U1 U2 U3	8
2	SR7.3.0.0	M1 M2 M3 M4 M5 M6 M7 U1 U2 U3	10
3	SR9.5.0.0	M1 M2 M3 M4 M5 M6 M7 M8 M9 U1 U2 U3 U4 U5	14
4	SR7.5.3.0	M1 M2 M3 M4 M5 M6 M7 U1 U2 U3 U4 U5 L1 L2 L3	15
5	SR15.9.5.1	M1 M2 M3 M4 M5 M6 M7 M8 M9 M10 M11 M12 M13 M14 M15 U1 U2 U3 U4 U5 U6 U7 U8 U9 L1 L2 L3 L4 L5 Z1	30
6, 7	reserved		

NOTE 1: SR refers to a stacked ring format used for intermediate spatial format objects.

NOTE 2: MULZ refers to the order in which audio signals for ISF objects occur.

5.6.3.8 num_dynamic_objects_bits

The `num_dynamic_objects_bits` codeword indicates the value of *num_dynamic_objects*.

The value of *num_dynamic_objects* is determined by the following equation:

$$num\_dynamic\_objects = num\_dynamic\_objects\_bits + 1.$$

In this equation, the bits of `num_dynamic_objects_bits` are interpreted as unsigned integer.

5.6.4 Additional metadata

5.6.4.1 oa_md_version_bits

`oa_md_version_bits` is an unsigned integer that determines the OAMD syntax version.

5.6.4.2 object_count_bits

The `object_count_bits` codeword indicates the value of *object_count*.

The value of *object_count* is determined by the following equation:

$$object\_count = object\_count\_bits + 1.$$

In this equation, the bits of the `object_count_bits` element are interpreted as an unsigned integer. *object_count* indicates the total number of objects in the bitstream.

5.6.4.3 b_alternate_object_data_present

The `b_alternate_object_data_present` flag indicates whether each `oa_element` block contains an `alternate_object_data_id_idx` element.

5.6.4.4 oa_element_count_bits

The oa_element_count_bits unsigned integer determines the number of oa_element blocks contained in the object_audio_metadata_payload block.

5.6.4.5 reserved_data_size_bits

The reserved_data_size_bits codeword indicates the value of *reserved_data_size*.

The *reserved_data_size* indicates the length of the *reserved_data* element plus the padding element in bytes and can be calculated by the following equation:

$$\text{reserved\_data\_size} = \text{reserved\_data\_size\_bits} + 1.$$

In this equation, the bits of the reserved_data_size_bits element are interpreted as an unsigned integer.

5.6.4.6 oa_element_id_idx

The oa_element_id_idx index indicates the type of the following oa_element element.

Table 26 lists the type of the oa_element element, indicated by the corresponding table index.

Table 26: The type of the oa_element indicated by oa_element_id_idx

oa_element_id_idx	Type of the oa_element element
0	Reserved
1	object_element
2-15	Reserved

5.6.4.7 oa_element_size_bits

The oa_element_size_bits codeword indicates the value of *oa_element_size*.

The *oa_element_size* indicates the total length in bytes of the following subsequent elements:

- alternate_object_data_id_idx
- b_discard_unknown_element
- oa_element
- padding

The value of *oa_element_size* is determined by the following equation:

$$\text{oa\_element\_size} = \text{oa\_element\_size\_bits} + 1.$$

In this equation, the bits of the oa_element_size_bits element are interpreted as an unsigned integer.

5.6.4.8 alternate_object_data_id_idx

The alternate_object_data_id_idx index indicates the type of the alternative data for the corresponding oa_element.

The type of the alternative data for the corresponding oa_element element depends on the type of the oa_element element (indicated by the oa_element_id_idx element). For oa_element_id_idx = 1, table 27 lists the type of the alternative data for the object_element element.

Table 27: The type of the alternative data for the `object_element` element indicated by `alternate_object_data_id_idx`

`alternate_object_data_id_idx`	Type of the alternative data for the `object_element` element
0	Default
1-15	Reserved
NOTE: The type of the alternative data for the `object_element` element currently supports one default value only, all other values are reserved for future use cases.	

5.6.4.9 b_discard_unknown_element

The `b_discard_unknown_element` flag indicates whether the intention is to discard the corresponding `oa_element` during a transcoding process, if the value of the `oa_element_id_idx` element is unknown.

5.6.4.10 b_object_not_active

The `b_object_not_active` flag indicates whether the object's audio essence is silent.

5.6.4.11 object_basic_info_status_idx

The `object_basic_info_status_idx` index indicates how the values of the elements contained in the *object_basic_info* block are determined.

Table 28 lists how the values of the elements contained in the *object_basic_info* block shall be determined.

Table 28: Values of the elements contained in the *object_basic_info* block

`object_basic_info_status_idx`	Action	Element values
0	Default	• *object_priority*=0 • *object_gain*=-∞ dB
1	Full update	All values are signalled in the `object_basic_info` block
2	Full reuse	No values signalled, all values are equal to the corresponding values of the previous metadata update
3	Mixed update/reuse	Some values are signalled in the `object_basic_info` block and some values are equal to the corresponding values of the previous metadata update (see clause 5.6.4.16)

5.6.4.12 b_object_in_bed_or_ISF

The `b_object_in_bed_or_ISF` helper flag shall be set if the corresponding object is contained in a bed instance or is an ISF object.

Whether an object is contained in a bed instance or is and ISF object can be determined from the order of objects and from the OAMD content description, specified in clause 5.6.3.

The objects are ordered in the bitstream as follows:

1) Objects contained in a bed instance.

2) ISF objects.

3) Dynamic objects (not contained in a bed instance nor ISF related).

The OAMD content description specifies the number of bed instances as *num_bed_instances*, the number of ISF objects `intermediate_spatial_format_idx`, and the number of dynamic objects in *num_dynamic_objects*. Objects in the first two classes shall be marked with `b_object_in_bed_instance_or_ISF` = true.

5.6.4.13 object_render_info_status_idx

The `object_render_info_status_idx` index indicates how the values of the *object_render_info* elements are determined.

Table 29 lists how the values of the elements contained in the *object_render_info* block shall be determined.

Table 29: Values of *object_render_info* elements

object_render_info_status_idx	Action	Values *object_render_info* elements
0	Default	• *pos3D_X*=0,5 • *pos3D_Y*=0,5 • *pos3D_Z*=0 • *width*=0 • *depth*=0 • *height*=0 • `zone_constraints_idx`=0 • `b_enable_elevation`=true • `b_use_screen_ref`=false • `b_snap`=false
1	Full update	All values are signalled in the `object_render_info` block
2	Full reuse	No values signalled, all values are equal to the corresponding values of the previous metadata update
3	Mixed update/reuse	As indicated by clause 5.6.4.17: some values are signalled in the `object_render_info` block and some values are equal to the corresponding values of the previous metadata update

5.6.4.14 b_additional_table_data_exists

The `b_additional_table_data_exists` flag indicates whether the `additional_table_data_size_bits` element and the `additional_table_data` block follow in the bitstream.

5.6.4.15 additional_table_data_size_bits

The `additional_table_data_size_bits` codeword indicates the value of *additional_table_data_size*.

The value of *additional_table_data_size* is determined by the following equation:

$$\text{additional\_table\_data\_size} = \text{additional\_table\_data\_size\_bits} + 1.$$

In this equation, the bits of the `additional_table_data_size_bits` element are interpreted as an unsigned integer. The value of *additional_table_data_size* indicates the total length of the following `additional_table_data` block and the `padding` element in bytes.

5.6.4.16 obj_basic_info_mask

The `obj_basic_info_mask` codeword indicates whether the `object_gain_idx` and `b_default_object_priority` elements are present in the bitstream.

Each bit of the `obj_basic_info_mask` element is a flag that indicates that a corresponding element is present in the bitstream. Table 30 lists the correspondence between bitstream element and bits of the `obj_basic_info_mask` element.

Table 30: Bitstream elements indicated by the `obj_basic_info_mask` element

obj_basic_info_mask bit	Corresponding bitstream element
0	`object_gain_idx`
1	`b_default_object_priority`

5.6.4.17 obj_render_info_mask

The obj_render_info_mask codeword indicates the bitstream elements contained in the object_render_info block.

Each bit of the obj_render_info_mask codeword is a flag that indicates whether corresponding elements are present in the bitstream. Table 31 lists the correspondence between bitstream elements and bits of the obj_basic_info_mask element.

Table 31: Bitstream elements indicated by the obj_render_info_mask element

obj_render_info_mask bit	Corresponding bitstream elements
0 (least-significant bit (LSB))	b_differential_position_specified, pos3D_X_bits, pos3D_Y_bits, pos3D_Z_sign_bits, pos3D_Z_bits, diff_pos3D_X_bits, diff_pos3D_Y_bits, diff_pos3D_Z_bits, b_object_distance_specified, b_object_at_inifity, and distance_factor_idx
1	zone_constraints_idx and b_enable_elevation
2	object_size_idx, object_size_bits, object_width_bits, object_depth_bits, and object_height_bits
3	b_object_use_screen_ref,screen_factor_bits, and depth_factor_idx

5.6.4.18 padding

The padding element is a sequence of zero bits. The number of padding bits corresponds to the size value of the block.

6 Joint object coding

6.1 Introduction

The JOC tool is a post-processor to the E-AC-3 decoder. It enables decoding of up to 16 object-based audio essences from a channel-based E-AC-3 bitstream.

The JOC decoder performs a QMF domain matrix operation to reconstruct the output objects. The coefficients of the reconstruction matrix are controlled by the JOC side information in the bitstream.

NOTE: Clause 7.1 specifies the QMF domain.

6.2 Bitstream syntax

6.2.1 joc

Syntax	No of bits
```
joc()
{
  joc_header();
  joc_info();
  joc_data();
  if (joc_ext_config > 0) {
    joc_ext_data();
  }
  padding_bits; ........................................ 0...7
}
```

6.2.2 joc_header

Syntax	No of bits
joc_header()	
{	
joc_dmx_config_idx;	3
joc_num_objects_bits;	6
joc_ext_config_idx;	3
}	

6.2.3 joc_info

Syntax	No of bits
joc_info()	
{	
joc_clipgain_x_bits;	3
joc_clipgain_y_bits;	5
joc_seq_count_bits;	10
for (obj = 0; obj < joc_num_objects; obj++) {	
b_joc_obj_present[obj];	1
if (b_joc_obj_present[obj]) {	
joc_num_bands_idx[obj];	3
b_joc_sparse[obj];	1
joc_num_quant_idx[obj];	1
joc_data_point_info(obj);	
}	
}	
}	

6.2.4 joc_data_point_info

Syntax	No of bits
joc_data_point_info(obj)	
{	
joc_slope_idx[obj];	1
joc_num_dpoints_bits[obj];	1
if (joc_slope_idx[obj] == 1) {	
for (dp = 0; dp < joc_num_dpoints[obj]; dp++) {	
joc_offset_ts_bits[obj,dp];	5
}	
}	
}	

6.2.5 joc_data

Syntax	No of bits
joc_data()	
{	
for (obj = 0; obj < joc_num_objects; obj++) {	
if (b_joc_obj_present[obj]) {	
for (dp = 0; dp < joc_num_dpoints[obj]; dp++) {	
if (b_joc_sparse[obj] == 1) {	
joc_channel_idx[0];	3
joc_huff_code = get_joc_huff_code(joc_num_channels, IDX);	
for (pb = 1; pb < joc_num_bands; pb++) {	
joc_hcw;	VAR
joc_channel_idx[pb] = huff_decode(joc_huff_code, joc_hcw);	
}	
joc_huff_code = get_joc_huff_code(joc_num_quant_idx[obj], VEC);	
for (pb = 0; pb < joc_num_bands; pb++) {	
joc_hcw;	VAR
joc_vec[pb] = huff_decode(joc_huff_code, joc_hcw);	
}	
}	
else {	
joc_huff_code = get_joc_huff_code(joc_num_quant_idx[obj], MTX);	
for (ch = 0; ch < joc_num_channels; ch++) {	
for (pb = 0; pb < joc_num_bands; pb++) {	

Syntax	No of bits
`` joc_hcw; ```	VAR

```
            joc_mtx[pb] = huff_decode(joc_huff_code, joc_hcw);
          }
        }
      }
    }
  }
 }
}
```

6.3 Bitstream description

6.3.1 joc

6.3.1.1 joc overview

The joc block is the top-level element that contains all bitstream elements of the JOC side information.

6.3.2 joc_header

6.3.2.1 joc_header overview

The joc_header block contains bitstream elements that indicate the JOC downmix configuration, the number of processed audio objects and the type of extensional JOC configuration data (if present in the bitstream).

6.3.2.2 joc_dmx_config_idx

The joc_dmx_config_idx index indicates the JOC downmix configuration.

Table 32 specifies the JOC downmix configuration indicated by joc_dmx_config_idx.

Table 32: JOC downmix channel configuration

joc_dmx_config_idx	Downmix configuration	Downmix channel
0	5.X	L, R, C, (LFE), Ls, Rs
1	7.X	L, R, C, (LFE), Ls, Rs, Lrs, Rrs
2	5.X + 2	L, R, C, (LFE), Ls, Rs, Lvh, Rvh
3	5.X with 90 degree phase shift	L, R, C, (LFE), Ls, Rs
4	5.X + 2 with 90 degree phase shift	L, R, C, (LFE), Ls, Rs, Lvh, Rvh
5...7	Reserved	
NOTE: If the LFE channel is present, this channel is not processed by the JOC decoder, but bypassed only.		

6.3.2.3 joc_num_channels

The joc_num_channels helper variable indicates the number of channels in the JOC downmix.

The value of joc_num_channels depends on the joc_dmx_config_idx index as listed in table 33.

Table 33: Number of downmix channels

joc_dmx_config_idx	joc_num_channels
0	5
1	7
2	7
3	5
4	7
5...7	reserved

6.3.2.4 joc_num_objects_bits

The `joc_num_objects_bits` unsigned integer indicates the value of *joc_num_objects*.

The value of *joc_num_objects* is determined by the following equation:

$$\text{joc\_num\_objects} = \text{joc\_num\_objects\_bits} + 1$$

6.3.2.5 joc_ext_config_idx

The `joc_ext_config_idx` index indicates the type of extensional configuration data that is present in the bitstream.

Table 34 specifies the extensional configuration data indicated by `joc_ext_config_idx`.

Table 34: JOC extensional configuration data

joc_ext_config_idx	Extensional configuration data type
0	no extensional configuration data present
1 ... 7	reserved

NOTE: Currently all extensional configuration data types are reserved for future use.

6.3.3 joc_info

6.3.3.1 joc_info overview

The `joc_info` block contains several bitstream elements per object that indicate the presence, quantization, frequency- and time resolution of the coded JOC parameters for the corresponding object.

6.3.3.2 joc_clipgain_x_bits, joc_clipgain_y_bits

The `joc_clipgain_x_bits` and `joc_clipgain_y_bits` unsigned integers indicate the value of *joc_clipgain*.

The value of *joc_clipgain* is determined by the following equation:

$$\text{joc\_clipgain} = \left(1 + \frac{\text{joc\_clipgain\_y\_bits}}{32} \times 2^{(\text{joc\_clipgain\_x\_bits} - 4)}\right).$$

The value of *joc_clipgain* is in [1; 8,75].

6.3.3.3 joc_seq_count_bits

The `joc_seq_count_bits` unsigned integer determines the value *joc_seq_count* that is the state of a frame sequence counter.

The frame sequence counter is used for splice detection in the decoder. It is incremented by one every consecutive frame (up to 1 023). When the maximum value is reached, the counter restarts with the value 1 in the following frame. The value 0 indicates the first frame in the bitstream or the first frame after a splice.

6.3.3.4 b_joc_object_present

The `b_joc_obj_present` flag indicates whether JOC side information is present in the bitstream for the corresponding audio object.

6.3.3.5 joc_num_bands_idx

The `joc_num_bands_idx` index indicates the value of *joc_num_bands*.

Table 35 specifies the value of *joc_num_bands* indicated by `joc_num_bands_idx`.

Table 35: Number of frequency bands for the JOC side information

joc_num_bands_idx	*joc_num_bands*
0	1
1	3
2	5
3	7
4	9
5	12
6	15
7	23

6.3.3.6 b_joc_sparse

The b_joc_sparse flag indicates whether the JOC side information is coded in sparse mode.

6.3.3.7 joc_num_quant_idx

The joc_num_quant_idx index indicates the value of *joc_num_quant*.

The value of *joc_num_quant* is the number of quantization steps, used for the coding of JOC side information parameters.

Table 36 specifies the value of *joc_num_quant* indicated by joc_num_quant_idx.

Table 36: Number of quantization steps for the JOC side information

joc_num_quant_idx	*joc_num_quant*
0	96
1	192

6.3.4 joc_data_point_info

6.3.4.1 joc_data_point_info overview

The joc_data_point_info block contains bitstream elements that indicate the temporal extension of the JOC side information. The side information comprises the number of transmitted data points and the temporal interpolation processing.

6.3.4.2 joc_slope_idx

The joc_slope_idx index indicates an interpolation type.

The description of the interpolation type indicated by joc_slope_idx is listed in table 37.

Table 37: Description of the interpolation type signalled by joc_slope_idx

joc_slope_select	Interpolation type description
0	smooth (linear interpolation)
1	steep (transition, no interpolation)

NOTE: Clause 6.6.5 specifies smooth and steep interpolation processing.

6.3.4.3 joc_num_dpoints_bits

The `joc_num_dpoints_bits` unsigned integer indicates the number of JOC data points *joc_num_dpoints*.

The value of *joc_num_dpoints* is determined by the following equation:

$$\text{joc\_num\_dpoints} = \text{joc\_num\_dpoints\_bits} + 1$$

6.3.4.4 joc_offset_ts_bits

The `joc_offset_ts_bits` unsigned integer indicates the QMF timeslot offset value *joc_offset_ts*.

The offset value *joc_offset_ts* for the corresponding data point is determined by the following equation:

$$\text{joc\_offset\_ts} = \text{joc\_offset\_ts\_bits} + 1$$

6.3.5 joc_data

6.3.5.1 joc_data overview

The `joc_data` block contains bitstream elements that indicate the value of `joc_idx`, `joc_vec` or `joc_mtx`.

6.3.5.2 joc_channel_idx

The `joc_channel_idx` index indicates the input channel for the corresponding parameter band.

Table 38 lists the input channel indicated by `joc_channel_idx` for the two different cases `joc_num_channels` = 5 and `joc_num_channels` = 7.

Table 38: JOC input channel for `joc_num_channels` = 5 or `joc_num_channels` = 7

`joc_channel_idx`	Input channel if `joc_num_channels` = 5	Input channel if `joc_num_channels` = 7
0	Left	Left
1	Right	Right
2	Centre	Centre
3	Left Surround	Left Side
4	Right Surround	Right Side
5	n.a.	Left Back
6	n.a.	Right Back

NOTE: The `joc_channel_idx` index is only present when `joc_sparse_idx` = 1.

6.3.5.3 joc_hcw

The `joc_hcw` Huffman codeword indicates the value of `joc_channel_idx`, `joc_vec` or `joc_mtx`, when decoded using the corresponding code.

6.4 Joint object coding decoder interfaces

6.4.1 Input

The *joc_num_channels* QMF matrices are input to the JOC decoder.

$\mathbf{Qin}_{\mathrm{JOC},(1,\,2,\,\ldots,\,\mathrm{joc\_num\_channels})}$

joc_num_channels complex valued QMF matrices corresponding to the number of input channels to be processed by the JOC decoder.

NOTE: The $\mathbf{Qin}_{\mathrm{JOC}}$ matrices each consist of *num_qmf_timeslots* columns and *num_qmf_subbands* rows.

6.4.2 Output

The *joc_num_objects* QMF matrices are output of the JOC decoder.

$\mathbf{Qout}_{JOC,(1,\,2,\,...,\,joc\_num\_objects)}$

joc_num_objects complex valued QMF matrices corresponding to the number of output objects to be reconstructed by the JOC decoder.

NOTE: The $\mathbf{Qout}_{JOC}$ matrices each consist of *num_qmf_timeslots* columns and *num_qmf_subbands* rows.

6.4.3 Control

Control parameters of the JOC decoder are coded in the JOC side information.

The parameters for the JOC reconstruction process are coefficients of a reconstruction matrix. To calculate these coefficients the decoder has an interface for temporal data points of coded matrix coefficients and additional parameters to control the decoding, dequantization and the interpolation process.

6.5 Parameter band mapping

For the JOC reconstruction process, the parameter bands of the JOC side information are mapped to the QMF subbands.

The parameters contained in the JOC side information are transmitted per parameter band. The number of parameter bands is indicated by *joc_num_bands*, as specified in clause 6.3.3.5.

The mapping of grouped QMF subbands to JOC parameter bands is listed in table 39. The columns specify different mappings depending on the value of `joc_num_bands` (1 to 23).

Table 39: Mapping of QMF subbands to JOC parameter bands

QMF subband	**JOC parameter band mapping dependent on *joc_num_bands***							
	23	**15**	**12**	**9**	**7**	**5**	**3**	**1**
0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	0	0
2	2	2	2	2	2	1	0	0
3	3	3	3	3	2	2	1	0
4	4	4	4	3	3	2	1	0
5	5	5	4	4	3	2	1	0
6	6	6	5	4	3	2	1	0
7	7	7	5	5	3	2	1	0
8	8	8	6	5	4	2	1	0
9	9	9	6	6	4	3	1	0
10	10	9	6	6	4	3	1	0
11	11	10	7	6	4	3	1	0
12 - 13	12	10	7	6	4	3	1	0
14 - 15	13	11	8	7	5	3	2	0
16 - 17	14	11	8	7	5	3	2	0
18 - 19	15	12	9	7	5	3	2	0
20 - 22	16	12	9	7	5	3	2	0
23 - 25	17	13	10	8	6	4	2	0
26 - 29	18	13	10	8	6	4	2	0
30 - 34	19	13	10	8	6	4	2	0
35 - 40	20	14	11	8	6	4	2	0
41 - 47	21	14	11	8	6	4	2	0
48 - 63	22	14	11	8	6	4	2	0

6.6 Joint object coding decoder

6.6.1 Introduction

The JOC decoder consecutively decodes, dequantizes and interpolates the JOC side information, and reconstructs the output objects using the processed side information.

The consecutive processing steps are specified as follows:

1) Decoding of the JOC bitstream elements as specified in clause 6.6.2.

2) Dequantization of the JOC parameters as specified in clause 6.6.4.

3) Interpolation of JOC parameters as specified in clause 6.6.5.

4) Reconstruction of the output objects as specified in clause 6.6.6.

6.6.2 Differential decoding of side information

For each object *obj* in [0; `joc_num_objects`-1], the JOC decoder shall decode the quantized coefficients for the reconstruction matrix *joc_mix_mtx_q* from the JOC bitstream elements.

In pseudo code notation, the decoding process for `b_joc_sparse[obj] == 1` is specified as:

Pseudocode 2

```
// input: arrays joc_channel_idx[obj][dp][pb] and joc_vec[obj][dp][pb]
// output: array joc_mix_mtx_q[obj][dp][ch][pb]
if (b_joc_obj_present[obj]) {
  nquant = (joc_num_quant_idx[obj] == 0) ? 96 : 192;
  for (dp = 0; dp < joc_num_dpoints; dp++) {
      offset = (joc_num_quant_idx[obj] == 0) ? 50 : 100;
      for (pb = 0; pb < joc_num_bands; pb++) {
        if (pb == 0) {
          joc_channel_idx_mod = joc_channel_idx[obj][dp][pb];
        }
        else {
          joc_channel_idx_mod = (joc_channel_idx[obj][dp][pb-
1] + joc_channel_idx[obj][dp][pb]) % joc_num_channels;
        }
        for (ch = 0; ch < joc_num_channels; ch++) {
          if (ch == joc_channel_idx_mod) {
            if (pb == 0) {
              joc_mix_mtx_q[obj][dp][ch][pb] = (offset + joc_vec[obj][dp][pb]) % nquant;
            }
            else {
              joc_mix_mtx_q[obj][dp][ch][pb] = (joc_mix_mtx_q[obj][dp][ch][pb-1] +
                                                 joc_vec[obj][dp][pb]) % nquant;
            }
          }
          else {
            joc_mix_mtx_q[obj][dp][ch][pb] = offset;
          }
        }
      }
    }
  }
}
```

In pseudo code notation, the decoding process for `b_joc_sparse[obj] == 0` is specified as:

Pseudocode 3

```
// input: array joc_mtx[obj][dp][ch][pb]
// output: array joc_mix_mtx_q[obj][dp][ch][pb]
if (b_joc_obj_present[obj]) {
  nquant = (joc_num_quant_idx[obj] == 0) ? 96 : 192;
  for (dp = 0; dp < joc_num_dpoints; dp++) {
    offset = (joc_num_quant_idx[obj] == 0) ? 48 : 96;
    for (ch = 0; ch < joc_num_channels; ch++) {
      joc_mix_mtx_q[obj][dp][ch][0] = (offset + joc_mtx[obj][dp][ch][pb]) % nquant;
      for (pb = 1; pb < joc_num_bands[obj]; pb++) {
        joc_mix_mtx_q[obj][dp][ch][pb] = (joc_mix_mtx_q[obj][dp][ch][pb-1] +
        joc_mtx[obj][dp][ch][pb]) % nquant;
      }
    }
  }
}
```

6.6.3 Huffman decoding of side information

The Huffman tables define huffman codewords implicitly.

To decode quantized side information values, the JOC decoder shall decode the huffman codewords into values specified in clause A.1.

The decoding process for `huff_decode(joc_huff_code, joc_hcw)` is specified as:

Pseudocode 4

```
huff_decode(joc_huff_code, joc_hcw)
{
  node = 0;
  next_bit = 0;
  do
  {
    next_bit = get_bit(joc_hcw);
    node = joc_huff_code[node][next_bit];
   }
   while (node > 0);

   return (-node-1);
}
```

The function *get_bit()* shall read one bit out of the codeword given as argument, starting with the MSB and pointing to the next bit for the following call.

The function *joc_get_huff_code(mode, type)* shall get the huffman code table *joc_huff_code[][]* as specified in clause A.1.

6.6.4 Dequantization of side information

The JOC decoder shall calculate the dequantized coefficients of the matrix *joc_mix_mtx_dq* from the quantized coefficients of the matrix *joc_mix_mtx_q*.

In pseudo code notation the calculation is specified as:

Pseudocode 5

```
for (obj=0; obj < joc_num_objects; obj++) {
  nquant = (joc_num_quant_idx[obj] == 0) ? 96 : 192;
  for (dp=0; dp < joc_num_dpoints; dp++) {
    for (ch=0; ch < joc_num_channels; ch++) {
      for (pb=0; pb < joc_num_bands; pb++) {
        joc_mix_mtx_dq[obj][dp][ch][pb] = (joc_mix_mtx_q[obj][dp][ch][pb] – nquant / 2) *
                                          820 / (4096 * (1 + joc_num_quant_idx[obj]));
      }
    }
  }
}
```

NOTE: If `joc_num_quant_idx[obj]` = 0 for one object *obj*, the values corresponding to this object are dequantized into the range [-9,6; 9,4]; otherwise, the range is [-9,6; 9,5].

6.6.5 Temporal interpolation of side information

The JOC decoder shall calculate an interpolated matrix *joc_mix_mtx_interp*.

In pseudo code notation, the calculation of *joc_mix_mtx_interp* is specified as:

Pseudocode 6

```
for (obj=0; obj < joc_num_objects; obj++) {
  for (ch=0; ch < joc_num_channels; ch++) {
    for (sb=0; sb < num_qmf_subbands; sb++) {
      for (ts=0; ts < num_qmf_timeslots; ts++) {
        if (joc_slope_idx[obj] == 0) {
          if (joc_num_dpoints == 1) {
            delta = joc_mix_mtx_dq[obj][0][ch][sb_to_pb(sb)] – joc_mix_mtx_prev[obj][ch][sb];
            joc_mix_mtx_interp[obj][ts][ch][sb] = joc_mix_mtx_prev[obj][ch][sb] +
                                                  (ts+1)*delta/num_qmf_timeslots;
          }
          else {
            ts_2 = floor(num_qmf_timeslots/2);
            if (ts < ts_2) {
              delta = joc_mix_mtx_dq[obj][0][ch][sb_to_pb(sb)] – joc_mix_mtx_prev[obj][ch][sb];
              joc_mix_mtx_interp[obj][ts][ch][sb] = joc_mix_mtx_prev[obj][ch][sb] +
                                                    (ts+1)*delta/ts_2;
            }
            else {
              delta = joc_mix_mtx_dq[obj][1][ch][sb_to_pb(sb)] –
                      joc_mix_mtx_dq[obj][0][ch][sb_to_pb(sb)];
              joc_mix_mtx_interp[obj][ts][ch][sb] = joc_mix_mtx_dq[obj][0][ch][sb_to_pb(sb)] +
                                                    (ts-ts_2+1)*delta/(num_qmf_timeslots-ts_2);
            }
          }
        }
        else {
          if (joc_num_dpoints == 1) {
            if (ts < joc_offset_ts[obj][0]) {
              joc_mix_mtx_interp[obj][ts][ch][sb] = joc_mix_mtx_prev[obj][ch][sb];
            }
            else {
              joc_mix_mtx_interp[obj][ts][ch][sb] = joc_mix_mtx_dq[obj][0][ch][sb];
            }
          }
          else {
            if (ts < joc_offset_ts[obj][0]) {
              joc_mix_mtx_interp[obj][ts][ch][sb] = joc_mix_mtx_prev[obj][ch][sb];
            }
            else if (ts < joc_offset_ts[obj][1]) {
              joc_mix_mtx_interp[obj][ts][ch][sb] = joc_mix_mtx_dq[obj][0][ch][sb];
            }
            else {
              joc_mix_mtx_interp[obj][ts][ch][sb] = joc_mix_mtx_dq[obj][1][ch][sb];
            }
          }
        }
      }
      joc_mix_mtx_prev[obj][ch][sb] =
      joc_mix_mtx_dq[obj][joc_num_dpoints-1][ch][sb_to_pb(sb)];
    }
  }
}
```

The function *sb_to_pb(subband)* shall return the parameter band value corresponding to the QMF subband given as an input argument as specified in clause 6.5.

EXAMPLE: If *joc_num_bands* = 15 and the input to *sb_to_pb(subband)* is the subband value 24, *sb_to_pb(24)* returns the value 13.

Before decoding the first E-AC-3 frame, all elements of the *joc_mix_mtx_prev* matrix shall be 0.

6.6.6 Reconstruction of the output objects

The JOC decoder shall calculate the output objects from the input channels utilizing the matrix *joc_mix_mtx_interp*.

Input and output are specified as follows:

Output objects

$z_{obj}(ts,sb)$ € $\mathbf{Qout}_{JOC,obj}$ for $obj = 0,...,joc\_num\_objects - 1$

NOTE 1: In pseudocode $z_{obj}(ts,sb)$ is addressed as *z[obj][ts][sb]*.

Input channels

$x_{ch}(ts,sb)$ € $\mathbf{Qin}_{JOC,ch}$ for $ch = 0,...,joc\_num\_channels - 1$

NOTE 2: In pseudocode $x_{ch}(ts,sb)$ is addressed as *x[ch][ts][sb]*.

In pseudo code notation, the calculation of the output objects $z_{obj}(ts,sb)$ from the input channels $x_{ch}(ts,sb)$ is specified as:

Pseudocode 7

```
for (obj=0; obj < joc_num_objects; obj++) {
  for (ch=0; ch < joc_num_channels; ch++) {
    for (ts=0; tp < num_qmf_timeslots; ts++) {
      for (sb=0; sb < num_qmf_subbands; sb++) {
        z[obj][ts][sb] += x[ch][ts][sb] * joc_mix_mtx_interp[obj][ch][ts][sb];
      }
    }
  }
}
```

7 Quadrature mirror filter bank domain processing

7.1 Introduction

To enable alias suppressed frequency-domain audio processing, a complex QMF analysis/synthesis transform pair can be employed.

The QMF analysis transforms a real-valued time-domain input signal into a number of spectral subband signals, each of which represents a decimated signal in that subband.

NOTE: When the transform is complex-valued, the output is oversampled by a factor of two compared to a regular cosine modulated QMF analysis.

The QMF synthesis is the reverse operation of the analysis. It transforms subband signals into a time-domain signal.

Typically, the analysis filter feeds into a history buffer taking the form of a *n_timeslots* × *n_subbands* matrix; processing proceeds on the history buffer which then feeds into the synthesis transform.

7.2 Complex analysis processing

The analysis filter transforms *n* time domain samples into *n* complex frequency-domain subband samples.

The analysis transform maintains a state buffer *qana_filt* of *n_qmf_length* real-valued samples.

NOTE: In the following, it is assumed that real and imaginary parts are interleaved.

1) Update the input buffer *qana_filt* by shifting out *n* old samples.

Pseudocode 8

```
for (j = n_qmf_length-1; j >= n; j--)
{
    qana_filt[j] = qana_filt[j-n];
}
```

2) Update the input buffer *qana_filt* by shifting in *n* new time-domain samples.

Pseudocode 9

```
for (j = n-1; j >= 0; j--)
{
    qana_filt[j] = pcm[n-1-j];
}
```

3) Compute vector *z* by pairwise multiplying the elements of *qana_filt* by window coefficients vector *QWIN[]*.

Pseudocode 10

```
for (j = 0; j < n_qmf_length; j++)
{
    z[j] = qana_filt[j] * QWIN[j];
}
```

4) Compute vector *u* as a summation over *z*

Pseudocode 11

```
for (j = 0; j < 2*n; j++)
{
    u[j] = 0;
    for (k = 0; k <= (n_qmf_length/(2*n))-1; k++)
    {
        u[j] += z[j + k*2*n];
    }
}
```

5) Compute *n* new subband samples of vector Q by the complex-valued matrix multiplication $\mathbf{Q} = \mathbf{M} \cdot \mathbf{u}$, where the following equation applies:

$$M_{k,j} = \exp\left(\frac{\mathrm{i} \times \pi \times (k + 1/2) \times (j - 1/2)}{n}\right), \begin{cases} 0 \le k < n \\ 0 \le j < 2 \times n \end{cases}.$$

Pseudocode 12

```
for (sb = 0; sb < n; sb++)
{
    /* note that Q[sb] is a complex datatype */
    Q[sb] = 0;
    for (j = 0; j < 2*n; j++)
    {
        Q[sb] += u[j] * exp(i*pi*(sb+0.5)*(j-0.5)/n);
    }
}
```

7.3 Complex synthesis processing

The synthesis filter transforms *n* complex frequency-domain subband samples *Q[]* into *n* real-valued time-domain samples.

The synthesis transform maintains a state buffer *qsyn_filt* of *2*n_qmf_length* real-valued samples.

1) Update the state buffer *qsyn_filt* by shifting out 2×*n* old samples.

Pseudocode 13

```
/* shift samples by 2*n */
for (j = 2*n_qmf_length-1; j >= 2*n; j--)
{
    qsyn_filt[j] = qsyn_filt[j-2*n];
}
```

2) Compute *2*n* new real-valued samples by multiplying the complex-valued subband samples *Q[]* by the matrix **N**, where the following equation applies:

$$N_{j,k} = \frac{1}{n} \times \exp\left(\frac{\mathrm{i} \times \pi \times (k + 1/2) \times (j - 2 \times n + 1/2)}{n}\right), \begin{cases} 0 \le k < n \\ 0 \le j < 2 \times n \end{cases}.$$

Store the samples in the state buffer.

Pseudocode 14

```
for (j = 0; j < 2*n; j++)
{
    qsyn_filt[j] = 0;
    for (sb = 0; sb < n; sb++)
    {
        exponent = i*pi/(4*n)*(2*sb + 1)*(2*j - 2*n - 1);
        qsyn_filt[j] += real(Q[sb]/n * exp(exponent));
    }
}
```

3) Extract samples from *qsyn_filt* to create the *n_qmf_length*-element vector *g*.

Pseudocode 15

```
for (j = 0; j < n_qmf_length/(2*n); j++)
{
    for (sb = 0; sb < n; sb++)
    {
        g[2*n*j +     sb] = qsyn_filt[4*n*j       + sb];
        g[2*n*j + n + sb] = qsyn_filt[4*n*j + 3*n + sb];
    }
}
```

4) Compute vector *w* by element-wise multiplication of vector *g* by window *QWIN[]*.

Pseudocode 16

```
for (j = 0; j < n_qmf_length; j++)
{
    w[j] = g[j] * QWIN[j];
}
```

5) Calculate *n* new output samples by summation of samples from vector *w*.

Pseudocode 17

```
for (ts = 0; ts < n; ts++)
{
    pcm[ts] = 0;
    for (j = 0; j < n_qmf_length/n; j++)
    {
        pcm[ts] += w[n*j + ts];
    }
}
```

7.4 Filter bank coefficients

The QMF is configured by the number *n* of QMF (sub)band and the filter length *n_qmf_length*.

The QMF in the present document uses *n* = 64 subbands, and a filter length of *n_qmf_length* = 640 (that is, 10 complex coefficients per subband). Table *QWIN* in the file ts_103420v010101p0.zip contains the coefficients.

8 Enhanced AC-3 decoding

8.1 Requirements on enhanced AC-3 decoding

To decode object-based audio as specified in the present document, an E-AC-3 decoder, as specified in ETSI TS 102 366 [1] shall be used to decode the channel-based downmix from the incoming bitstream. The following additional requirements apply:

- The E-AC-3 decoder shall pass the EMDF payload for JOC metadata to the decoder specified in the present document.
- The E-AC-3 decoder shall pass the EMDF payload for OAMD metadata to the decoder specified in the present document.
- The E-AC-3 decoder shall decode all the dependent substreams, as specified in ETSI TS 102 366 [1], clause E.2.8.2, and pass them to the decoder specified in the present document.

8.2 Requirements on EMDF container

To be backward compatible, the payloads for OAMD and JOC side information shall be carried in the EMDF container.

Table 40 lists normative requirements for the EMDF payloads.

Table 40: Payload restrictions for OAMD and JOC

	OAMD	JOC
payload_id	11	14
frequency	1/frame	1/frame

Table 41 lists normative requirements for the EMDF payload configuration data, as specified in ETSI TS 102 366 [1], clause H.2.2.3.

Table 41: Payload configuration data

Payload configuration data element	Value
duratione	0
groupide	1
groupid	equal for OAMD payload and JOC payload
codecdatae	1
discard_unknown_payload	0
create_duplicate	0
remove_duplicate	0
priority	0
proc_allowed	0

Annex A (normative): Tables

A.1 JOC Huffman code tables

Table A.1: JOC Huffman code joc_huff_tree_coarse_generic

Code table name	joc_huff_code_coarse_generic
Mode	0
Type	MTX

Table A.2: JOC Huffman code joc_huff_tree_fine_generic

Code table name	joc_huff_code_fine_generic
Mode	1
Type	MTX

Table A.3: JOC Huffman code joc_huff_tree_coarse_coeff_sparse

Code table name	joc_huff_code_coarse_coeff_sparse
Mode	0
Type	VEC

Table A.4: JOC Huffman code joc_huff_tree_fine_coeff_sparse

Code table name	joc_huff_code_fine_coeff_sparse
Mode	1
Type	VEC

Table A.5: JOC Huffman code joc_huff_tree_5ch_pos_index_sparse

Code table name	joc_huff_code_5ch_pos_index_sparse
Mode	5
Type	IDX

Table A.6: JOC Huffman code joc_huff_tree_7ch_pos_index_sparse

Code table name	joc_huff_code_7ch_pos_index_sparse
Mode	7
Type	IDX

A.2 Speaker Zones

Table A.7: Speaker to zone assignment

Speaker	Zone
L	Screen
R	Screen
C	Screen
Ls	Surround, Back/Side (see note 2)
Rs	Surround, Back/Side (see note 2)
Lb	Back
Rb	Back
Tfl	Top-Bottom
Tfr	Top-Bottom
Tbl	Top-Bottom
Tbr	Top-Bottom
Tsl	Top-Bottom
Tsr	Top-Bottom
Tl	Top-Bottom
Tr	Top-Bottom
Tfc	Top-Bottom
Tbc	Top-Bottom
Tc	Top-Bottom
Cb	Surround, Back
Lw	Screen
Rw	Screen
Lscr	Screen
Rscr	Screen
NOTE 1: The centre-and-back zone is a union of the back zone with the centre speaker. NOTE 2: Depending on the representation speaker configuration: Side for 5.X.X, Back otherwise.	

Annex B (informative): Conversion to ADM format

B.1 Introduction

The present annex specifies a conversion to ADM [i.1] that can be used as a post processing step to the normative decoding process, specified in the present document.

Metadata updates, specified in clause 4.4, are input to the conversion process.

NOTE: The result of the conversion of one metadata update is the combination of the individual conversions, defined in this annex.

The conversion is specified in tables providing information on how to determine the values of the ADM elements and attributes. These values can either be fixed or calculated using the metadata specified in clause 4.4 as input.

EXAMPLE 1: Table B.1 provides the information on the determination of the value for attributes of *admExampleElement* for the case that the value should be set to a fixed value and for the case that the value should be calculated using the fictive value *input_x* as input. Additionally, the table provides the information on the determination of the value for a sub-element of *admExampleElement* and for an attribute of this sub-element.

Table B.1: Determination of values in the *admExampleElement* element

Element name	Attribute name	Value
admExampleElement	exampleAttributeOne	1,0 (fixed value)
admExampleElement	exampleAttributeTwo	3 × input_x (calculation using one value as input)
admExampleSubElement		12 × input_x (calculation using one value as input)
admExampleSubElement	exampleAttributeThree	3,0 (fixed value)

The values of attributes and sub-elements of the following ADM elements are the output of the conversion process:

- *audioChannelFormat* specified in Recommendation ITU-R BS.2076 [i.1], clause 5.3.
- *audioBlockFormat* specified in Recommendation ITU-R BS.2076 [i.1], clause 5.4.
- *audioProgrammeReferenceScreen* specified in Recommendation ITU-R BS.2076 [i.1], clause 5.8.3.
- *audioPackFormat* specified in Recommendation ITU-R BS.2076 [i.1], clause 5.5.

The number of required ADM elements and sub-elements is as follows:

- The number of *audioChannelFormat* elements is equal to the number of objects. All metadata updates for one object are contained in one *audioChannelFormat* element.

EXAMPLE 2: The conversion of object-audio metadata for 12 objects requires 12 *audioChannelFormat* elements.

- The number of *audioBlockFormat* elements is equal to the number of all updates for all objects. The *audioChannelFormat* element contains one or more *audioBlockFormat* elements, where each contains one update of the properties for the corresponding object.

EXAMPLE 2: If each of the 12 objects has 8 updates of its properties, the conversion requires 96 *audioBlockFormat* elements. Each of the 12 *audioChannelFormat* elements contains references to 8 *audioBlockFormat* elements.

- The number of *audioProgrammeReferenceScreen* elements is one. The specification of the reference screen size is equal for all objects.

EXAMPLE 3: One instance of the *audioProgrammeReferenceScreen* element specifies the reference screen size for all of the 8 updates of the 12 objects.

- The number of *audioPackFormat* elements is in the range [1; number of objects]. The *audioPackFormat* element groups one or more *audioChannelFormat* elements and defines a common value for *importance* for the group (see clause B.2.3). The number of required *audioPackFormat* elements depends on how many different values for *priority* are present.

EXAMPLE 4: In the first update all of the 12 objects have the same value for *priority*, so all *audioChannelFormat* elements can be grouped to one *audioPackFormat* element. On the second update 6 objects have a different value from the other 6 objects, but both values are different from the first update. This means two new *audioPackFormat* elements are required. All together three *audioPackFormat* elements are required for these two updates.

B.2 Object properties

B.2.1 Position

B.2.1.1 Introduction

The conversion of the metadata contained in the *position()* block depends on the value of its element *anchor*.

B.2.1.2 Room related position metadata

Conversion of the position metadata elements if anchor = room.

Table B.2: Determination of values in the *audioChannelFormat* element

Element name	Attribute name	Value
audioChannelFormat	typeDefinition	Objects
audioChannelFormat	typeLabel	0003

Table B.3: Determination of values in the *audioBlockFormat* element

Element name	Attribute name	Value
cartesian		1
position	coordinate="X"	$2 \times \left(\min\left(0; \max(1; x)\right) - 0{,}5\right)$
position	coordinate="Y"	$2 \times \left(0{,}5 - \min\left(0; \max(1; y)\right)\right)$
position	coordinate="Z"	$\min\left(0; \max(1; z)\right)$
NOTE: The values for (x; y; z) are contained in the *coordinates* element of the *position()* block.		

B.2.1.3 Screen related position metadata

Conversion of the position metadata elements if anchor = screen.

Table B.4: Determination of values in the *audioChannelFormat* element

Element name	Attribute name	Value
audioChannelFormat	typeDefinition	Objects
audioChannelFormat	typeLabel	0003

Table B.5: Determination of values in the *audioBlockFormat* element:

Element name	Attribute name	Value
cartesian		1
screenRef		1
position	coordinate="X"	$2 \times \left(\min\left(0; \max(1; x)\right) - 0{,}5\right)$
position	coordinate="Y"	$2 \times \left(0{,}5 - \min\left(0; \max(1; y)\right)\right)$
position	coordinate="Z"	$\min\left(0; \max(1; z)\right)$
NOTE: The values for (x; y; z) are contained in the *coordinates* element of the *position()* block.		

Table B.6: Determination of values in the *audioProgrammeReferenceScreen* element

Element name	Attribute name	Value
aspectRatio		1,78
screenCentrePosition	coordinate="X"	0
screenCentrePosition	coordinate="Y"	0
screenCentrePosition	coordinate="Z"	1
screenWidth	coordinate="X"	2 × ref_screen_ratio

B.2.1.4 Speaker-related position metadata

Conversion of the position metadata elements if anchor = speaker.

Table B.7: Determination of values in the *audioChannelFormat* element

Element name	Attribute name	Value
audioChannelFormat	typeDefinition	DirectSpeakers
audioChannelFormat	typeLabel	0001
audioChannelFormat	audioChannelFormatName	as specified in table B.9
audioChannelFormat	audioChannelFormatID	as specified in table B.9

Table B.8: Determination of values in the *audioBlockFormat* element

Element name	Attribute name	Value
cartesian		1
speakerLabel		as specified in table B.10
position	coordinate="X"	as specified in table B.10
position	coordinate="Y"	as specified in table B.10
position	coordinate="Z"	as specified in table B.10

Table B.9: audioChannelFormatID and audioChannelFormatName for different speaker identifiers

Speaker identifier	AudioChannelFormatID	AudioChannelFormatName
Left (L)	AC_00011001	RoomCentricLeft
Right (R)	AC_00011002	RoomCentricRight
Center (C)	AC_00011003	RoomCentricCenter
Low-frequency Effects (LFE)	AC_00011004	RoomCentricLFE
Left Surround (Ls)	AC_00011005	RoomCentricLeftSideSurround
Right Surround (Rs)	AC_00011006	RoomCentricRightSideSurround
Left Rear Surround (Lrs)	AC_00011007	RoomCentricLeftRearSurround
Right Rear Surround (Rrs)	AC_00011008	RoomCentricRightRearSurround
Left Front High (Lfh)	AC_00011009	RoomCentricLeftFrontHigh
Right Front High (Rfh)	AC_0001100a	RoomCentricRightFrontHigh
Left Top Middle (Ltm)	AC_0001100b	RoomCentricLeftTopMiddle
Right Top Middle (Rtm)	AC_0001100c	RoomCentricRightTopMiddle
Left Rear High (Lrh)	AC_0001100d	RoomCentricLeftRearHigh
Right Rear High (Rrh)	AC_0001100e	RoomCentricRightRearHigh
Left Front Wide (Lw)	AC_0001100f	RoomCentricLeftWide
Right Front Wide (Rw)	AC_00011010	RoomCentricRightWide
Low-frequency Effects 2 (LFE2)	AC_00011011	RoomCentricLFE2

NOTE 1: The speaker identifier is contained in the *coordinates* element of the *position()* block.
NOTE 2: The listed audioChannelFormatID and audioChannelFormatName entries are custom definitions for ADM, as specified in Recommendation ITU-R BS.2076 [i.1], section 6.

Table B.10: speakerLabel and position attributes for different speaker identifiers

Speaker identifier	SpeakerLabel	Position.coordinate="X"	Position.coordinate="Y"	Position.coordinate="Z"
Left (L)	RC_L	-1	1	0
Right (R)	RC _R	1	1	0
Center (C)	RC _C	0	1	0
Low-frequency Effects (LFE)	RC _LFE	-1	1	-1
Left Surround (Ls)	RC _LSS	-1	0	0
Right Surround (Rs)	RC _RSS	1	0	0
Left Rear Surround (Lrs)	RC _LRS	-1	-1	0
Right Rear Surround (Rrs)	RC _RRS	1	-1	0
Left Front High (Lfh)	RC _LFH	-1	1	1
Right Front High (Rfh)	RC _RFH	1	1	1
Left Top Middle (Ltm)	RC _LTM	-1	0	1
Right Top Middle (Rtm)	RC _RTM	1	0	1
Left Rear High (Lrh)	RC _LRH	-1	-1	1
Right Rear High (Rrh)	RC _RRH	1	-1	1
Left Front Wide (Lw)	RC _LW	-1	0,677419	0
Right Front Wide (Rw)	RC _RW	1	0,677419	0
Low-frequency Effects 2 (LFE2)	RC _LFE2	1	1	-1

NOTE: The speaker identifier is contained in the *coordinates* element of the *position()* block.

B.2.2 Size

Table B.11: Requirements on the *audioChannelFormat* element

Element name	Attribute name	Value
audioChannelFormat	typeDefinition	Objects
audioChannelFormat	typeLabel	0003

Table B.12: Requirements on the *audioBlockFormat* element

Element name	Attribute name	Value
cartesian		1
width		width
depth		depth
height		height

B.2.3 Priority

Table B.13: Requirements on the *audioPackFormat* element

Element name	Attribute name	Value
audioPackFormat	importance	round($10 \times$priority)

B.2.4 Gain

The gain value should be applied to the object essence.

B.2.5 Channel lock

Table B.14: Requirements on the *audioChannelFormat* element

Element name	Attribute name	Value
audioChannelFormat	typeDefinition	Objects
audioChannelFormat	typeLabel	0003

Table B.15: Requirements on the *audioBlockFormat* element

Element name	Attribute name	Value
channelLock		channel_lock
channelLock	maxDistance	0,4

B.2.6 Zone constraints

Table B.16: Requirements on the *audioChannelFormat* element

Element name	Attribute name	Value
audioChannelFormat	typeDefinition	Objects
audioChannelFormat	typeLabel	0003

Table B.17: Requirements on the *audioBlockFormat* element

Element name	Attribute name	Value
cartesian		1
zoneExclusion		1

Based on the zone constraints the `zoneExclusion` element should contain one or more `zone` sub-elements as specified in table B.18.

Table B.18: Mapping of zone constraints to ADM Zone elements

Zone constraints	Number of zone elements	Value(s) of zone element(s)
Back zone excluded	1	• 'ZM1'
Side zone excluded	2	• 'ZM2_Left' • 'ZM2_Right'
All horizontal zones excluded, except centre-and-back zone	4	• 'ZM3_ScreenLeft' • ''ZM3_SideLeft' • ''ZM3_ScreenRight' • ''ZM3_SideRight'
All horizontal zones excluded, except screen zone	1	• ''ZM4'
All horizontal zones excluded, except surround zone	1	• ''ZM5'
Top-Bottom zone excluded (independent from horizontal zones)	1	• 'ZU' • 'ZB'

Each zone sub-element should contain the six attributes *minX*, *maxX*, *minY*, *maxY*, *minZ* and *maxZ* as specified in table B.19.

Table B.19: Attributes of ADM zone elements

Zone	minX	maxX	minY	maxY	minZ	maxZ
ZM1	-1	1	-1	-0,41934	-0,49900	0,49900
ZM2_Left	-1	-0,75806	-0,41934	0,83871	-0,49900	0,49900
ZM2_Right	0,75806	1	-0,41934	0,83871	-0,49900	0,49900
ZM3_ScreenLeft	-1	-0,16129	0,5	1	-0,49900	0,49900
ZM3_SideLeft	-1	-0,51611	-0,70700	0,49999	-0,49900	0,49900
ZM3_ScreenRight	0,16129	1	0,5	1	-0,49900	0,49900
ZM3_SideRight	0,5611	1	-0,70700	0,49999	-0,49900	0,49900
ZM4	-1	1	-1	0,83871	-0,49900	0,49900
ZM5	-1	1	0,5	1	-0,49900	0,49900
ZU	-1	1	-1	1	0,49950	1
ZB	-1	1	-1	1	-1	-0,49950

B.3 Timing metadata

Table B.20: Requirements on the *audioBlockFormat* element:

Element name	Attribute name	Value
audioBlockFormat	rtime	HH:MM:SS.sssss, where seconds = $\frac{\text{frame\_offset}+\text{start\_time}}{f_S}$.
audioBlockFormat	duration	HH:MM:SS.sssss, where seconds = $\frac{\text{start\_time}_{\text{subseq}}-\text{start\_time}}{f_S}$.
NOTE 1: The values of HH, MM and SS.sssss are calculated as follows: $\mathrm{HH} = \left\lfloor \frac{\text{seconds}}{3600} \right\rfloor$, $\mathrm{MM} = mod\left(\left\lfloor \frac{\text{seconds}}{60} \right\rfloor, 60\right)$, $\mathrm{SS.sssss} = mod(\text{seconds}, 60)$. NOTE 2: f_S is the sampling frequency of the audio essence. NOTE 3: $\text{start\_time}_{\text{subseq}}$ is the *start_time* value of the subsequent metadata update.		

History

Document history		
V1.1.1	July 2016	Publication