# ETSI TS 103 420 V1.2.1 (2018-10)

TECHNICAL SPECIFICATION

**Backwards-compatible object audio carriage using Enhanced AC-3**

EBU
OPERATING EUROVISION

Reference

RTS/JTC-042

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

# Contents

# Intellectual Property Rights

# 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_103420v010201p0.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 representation 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 (OBA) and specifies the decoder interface.
- Clause 5 specifies object audio metadata (OAMD), the OBA 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 tool that is used by the JOC tool.
- Annex C lists requirements on packaging a bitstream as defined in the present document into an ISO based media file format.
- Annex D lists requirements on an MPEG DASH media presentation description signaling a bitstream as defined in the present document.
- Annex E provides requirements on a MPEG CMAF compliant media format containing a bitstream as defined in the present document.

# 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 OBA 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 OBA, the JOC tool, the quadrature mirror filter bank tool, packaging the bitstream into ISO based media file format, the signalling in an MPEG DASH media presentation description, and requirements on a MPEG CMAF compliant media file, containing a bitstream as defined in the present document.

5) 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 https://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] ISO/IEC 23000-19: "Common Media Application Format for Segmented Media".

[3] ISO/IEC 23009-1: 'Dynamic adaptive streaming over HTTP (DASH) -- Part 1: Media presentation description and segment formats'.

[4] ISO/IEC 14496-12: "Information technology -- Coding of audio-visual objects -- Part 12: ISO base media file format".

## 2.2 Informative References

References are either specific (identified by date of publication and/or edition number or version number) or nonspecific. For specific references, only the cited version applies. For non-specific references, the latest version of the 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 of terms, symbols, abbreviations and conventions

## 3.1 Terms

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

**audio channel:** data representing an audio signal designated to a dedicated speaker position

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

**audio object:** data representing an object essence plus corresponding object properties

**bed:** group of multiple bed objects

**bed object:** static object whose spatial position is fixed by an assignment to a speaker

**block:** portion of a frame

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

**channel configuration:** targeted speaker layout

**codec frame:** series of PCM samples or encoded audio data representing the same time interval for all channels in the configuration

NOTE: Decoders usually operate in a codec-frame-by-codec-frame mode.

**common media application format:** MPEG media file format optimized for delivery of a single adaptive multimedia presentation to a large number and variety of devices; compatible with a variety of adaptive streaming, broadcast, download, and storage delivery methods

**DASH descriptor:** DASH element used for property signalling

NOTE: DASH descriptors share a common structure.

**DASH supplemental property descriptor:** DASH descriptor that specifies supplemental information about the containing DASH element that may be used by the DASH client for optimizing the processing

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

**elementary stream:** bitstream that consists of a single type of encoded data (audio, video, or other data)

**Enhanced AC-3:** audio coding system developed by Dolby and specified in ETSI TS 102 366 [1]

**extensible metadata delivery format:** set of rules and data structures that enables robust signaling of metadata in an end-to-end process, involving a container, metadata payloads, and authentication protocols

NOTE: Specified in ETSI TS 102 366 [1].

**intermediate spatial format:** format that defines spatial position by distributing the signal to speakers located in a stacked-ring configuration

**ISO based media file format:** media file format as specified in ISO/IEC 14496-12 [4]

**joint object coding:** algorithms for delivering immersive audio at low bitrates, achieved by conveying a multi-channel downmix of the immersive content using perceptual audio coding algorithms together with parametric side information that enables the reconstruction of the audio objects from the downmix in the decoder

**low-frequency effects:** optional single channel of limited bandwidth (typically less than 120 Hz)

**media presentation description:** formalized XML-based description for a DASH media presentation for the purpose of providing a streaming service

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

**object audio metadata:** metadata used for rendering an audio object. The rendering information may dynamically change (e.g. gain and position)

**object-based audio:** audio comprised of one or more audio objects

**object property:** metadata associated with an object essence, which indicates the author's intention for the rendering process

**pulse code modulation:** digital representation of an analog signal where the amplitude of the signal is sampled at uniform intervals

**QMF subband:** frequency range represented by one row in a QMF matrix, carrying a subsampled signal

**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

**rendering:** processing of audio content to adapt it to a specific loudspeaker layout

**room-anchored coordinates:** coordinates specifying a position relative to a coordinate system which 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

**substream:** part of an AC-4 bitstream, contains audio data and corresponding metadata

**zone:** sub-volume of 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 |
| CMAF | Common Media Application Format |
| DASH | Dynamic Adaptive Streaming over HTTP |
| E-AC-3 | Enhanced AC-3 |
| EMDF | Extensible Metadata Delivery Format |
| HH | Hours |
| HTTP | HyperText Transfer Protocol |
| IDX | Index |
| ISF | Intermediate Spatial Format |
| JOC | Joint Object Coding |
| LFE | Low-Frequency Effects |
| MM | Minutes |
| MPD | Media Presentation Description |
| MPEG | Moving Pictures Expert Group |
| MSB | Most Significant Bit |
| MTX | Matrix |
| MULZ | Mid-Upper-Lower-Zenith |
| OAMD | Object Audio MetaData |
| OBA | Object-Based Audio |
| PCM | Pulse Code Modulation |
| QMF | Quadrature Mirror Filter |
| SR | Stacked Ring |
| SS | Seconds |
| VEC | Vector |
| ZB | Bottom zone |
| ZU | Top zone |

## 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

OBA 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 audio 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).

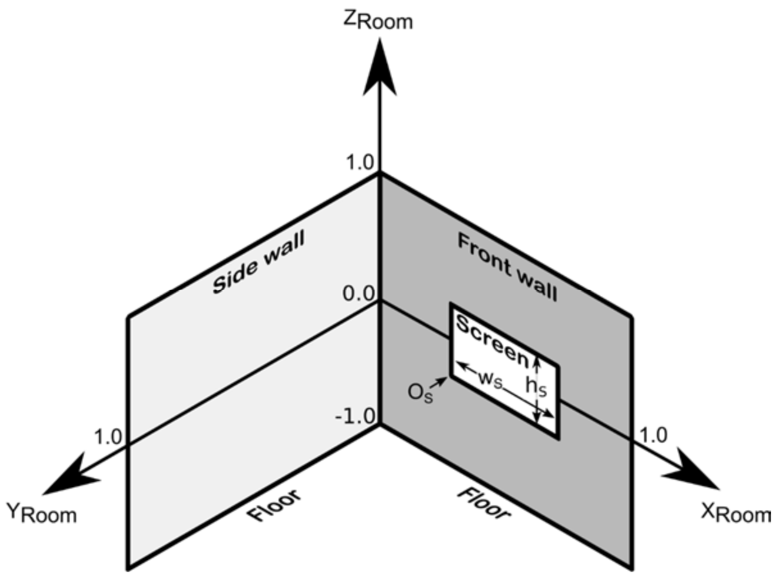


**Figure 1: Room-anchored coordinate system**

$w_S$ and $h_S$ are the dimensions of the screen, and $O_S$ 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:

$(X_{\text{room}}; Y_{\text{room}}; Z_{\text{room}}) = O_S + \left(w_S \times X_{\text{screen}}; Y_{\text{screen}}; \frac{h_S}{2} \times (Z_{\text{screen}}+1)\right)$ where $w_S$, $h_S$ and $O_S$ are specified in clause 4.2.1.

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

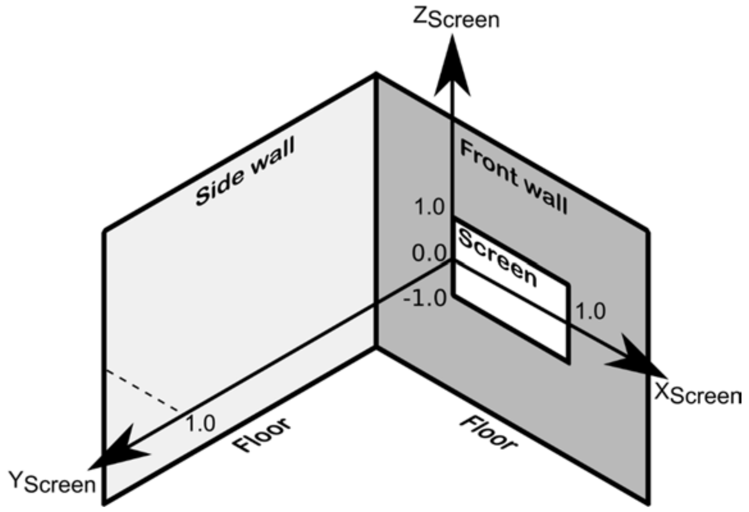


**Figure 2: Screen-anchored coordinate system**

### 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

OBA content is signalled using a specific configuration of the E-AC-3 bitstream.

The decoder shall decode the incoming OBA 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 OBA 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 quadrature mirror filter bank domain, as specified in clause 7.2.
3) Utilize the JOC decoder tool, as specified in clause 6, to reconstruct the audio object (object) essences from the channel-based downmix.
4) Transform the quadrature mirror filter bank 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 OBA comprising object audio 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;
  t_divergence       divergence_update;
  t_trim             trim_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_LS, // room-centric left side/surround
  RC_RS, // room-centric right side/surround
  RC_LB, // room-centric left back
  RC_RB, // room-centric right back
  RC_TFL, // room-centric top front left
  RC_TFR, // room-centric top front right
  RC_TSL, // room-centric top side left
  RC_TSR, // room-centric top side right
  RC_TBL, // room-centric top back left
  RC_TBR, // room-centric top back right
  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 = 8
}

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;
} t_position;

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

typedef struct {
  int   trim_mode;
  float trim_centre;
  float trim_surround;
  float trim_height;
  float bal3D_Y_tb;
  float bal3D_Y_lis;
} t_trim

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
typedef float                 t_divergence;
```

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 OBA processing.

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

The present document specifies:

- the general OBA content description in clause 5.1a
- 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:

- OAMD content description metadata, as specified in clause 5.6.0
- object properties metadata, as specified in clause 5.6.1
- timing metadata, as specified in clause 5.6.2
- other metadata, as specified in clause 5.6.4

## 5.1a OBA content description

The top-level `object_audio_metadata_payload` and the contained `program_assignment` contain several bitstream elements that carry general information about the transmitted OBA content such as:

- version of the OAMD payload syntax
- total number of objects
- type of objects (e.g. dynamic objects or bed instances)
- program composition

For the full list of transmitted information and the detailed semantics see clause 5.6.0.

## 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 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 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; position property is specified in clause 5.2.1.4.

NOTE: Clause 5.6.4.8 specifies how to determine if an object is contained in a bed.

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 |
| ext_prec_pos3D_X | Clause 5.6.6.4.3 |
| ext_prec_pos3D_Y | Clause 5.6.6.4.4 |
| ext_prec_pos3D_Z | Clause 5.6.6.4.5 |

## 5.2.1.2 Position in room-anchored coordinates

The decoder shall set *anchor* = ROOM.

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

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

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

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

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

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

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

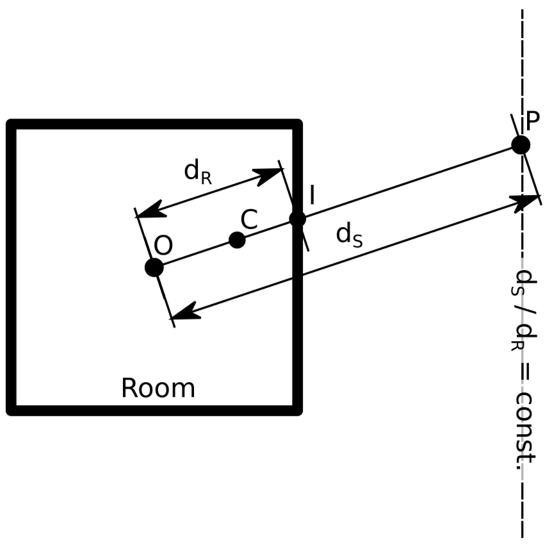


**Figure 3: Projection of a room-anchored position to a position outside the room**

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

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

Position 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 $\mathrm{P}_{\mathrm{interpolated}}$ can be determined as follows:

1) Let $\mathrm{C_R} = \mathrm{O_S} + \left(\mathrm{w_S} \times \mathrm{X_{screen}}; \mathrm{Y_{screen}}; \frac{\mathrm{h_S}}{2} \times (\mathrm{Z_{screen}}+1)\right)$ be the transformation of screen-anchored coordinates to the room-anchored system as specified in clause 4.2.2.

2) Let $\mathrm{C_S} = P$.

3) Let $\mathrm{M_1} = \begin{vmatrix} \mathrm{screen\_factor} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \mathrm{screen\_factor} \end{vmatrix}$

4) Let $\mathrm{M_2} = \begin{vmatrix} y^{\mathrm{depth\_factor}} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & y^{\mathrm{depth\_factor}} \end{vmatrix}$

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

6) $\mathrm{P_{interpolated}} = \mathrm{M_2} \times (\mathrm{M_1} \times \mathrm{C_S} + (\mathrm{I} - \mathrm{M_1}) \times \mathrm{C_R}) + (\mathrm{I} - \mathrm{M_2}) \times \mathrm{C_R}$.

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

The *bed_channel_assignment* is a list of channel labels that indicate a speaker-anchored coordinate. Objects contained in the bed are mapped to the channel labels in the same order. Each object has one coordinate, e.g. one speaker it is intended be played back from.

EXAMPLE: For a bed 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, RC_LS, RC_RS). In this case the channel label of *obj0* is 'RC_L', indicating the coordinate of the loudspeaker "Room Centric Left", the label of *obj1* is 'RC_R', and so on.

For each object contained in a bed, the decoder shall set *speaker_label* 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 $(\text{width}; \text{depth}; \text{height})$ 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 $(\text{width}; \text{depth}; \text{height}) = (\text{object\_width}; \text{object\_depth}; \text{object\_height})$.

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 is a Boolean property that shall be set to `b_object_snap`. Channel lock 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 |

## 5.2.7 Object Divergence

The object divergence property is the data that is provided as *object_divergence* at the decoder interface.

Object divergence is a property that shall be set to the value of *object_divergence*. Divergence is used to convert one object into two objects, where the energy is spread along the X-Axis.

Table 8 lists the related bitstream element and its reference.

**Table 8: Object divergence related bitstream elements**

| bitstream element | reference |
|---|---|
| `object_div_mode` | Clause 5.6.6.3.2 |
| `object_div_table` | Clause 5.6.6.3.3 |
| `object_div_code` | Clause 5.6.6.3.4 |

## 5.2.8 Object Trim

The object trim property is the data that is provided as *object_trim* at the decoder interface.

Object trim is a property that shall be set to the transmitted trim values listed in table 9. Object trim can be used to lower the level of out-of-screen elements that are included in the mix. This can be desirable when immersive mixes are reproduced in layouts with few loudspeakers.

Table 9 lists the related bitstream element and its reference.

**Table 9: Object trim related bitstream elements**

| bitstream element | reference |
|---|---|
| `warp_mode` | Clause 5.6.5.1 |
| `global_trim_mode` | Clause 5.6.5.2 |
| `b_default_trim` | Clause 5.6.5.3 |
| `b_disable_trim` | Clause 5.6.5.4 |
| `trim_balance_presence` | Clause 5.6.5.5 |
| `trim_centre` | Clause 5.6.5.6 |
| `trim_surround` | Clause 5.6.5.7 |
| `trim_height` | Clause 5.6.5.8 |
| `bal3D_Y_sign_tb_code` | Clause 5.6.5.9 |
| `bal3D_Y_amount_tb` | Clause 5.6.5.10 |
| `bal3D_Y_sign_lis_code` | Clause 5.6.5.11 |
| `bal3D_Y_amount_lis` | Clause 5.6.5.12 |

## 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 codec frame (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*.

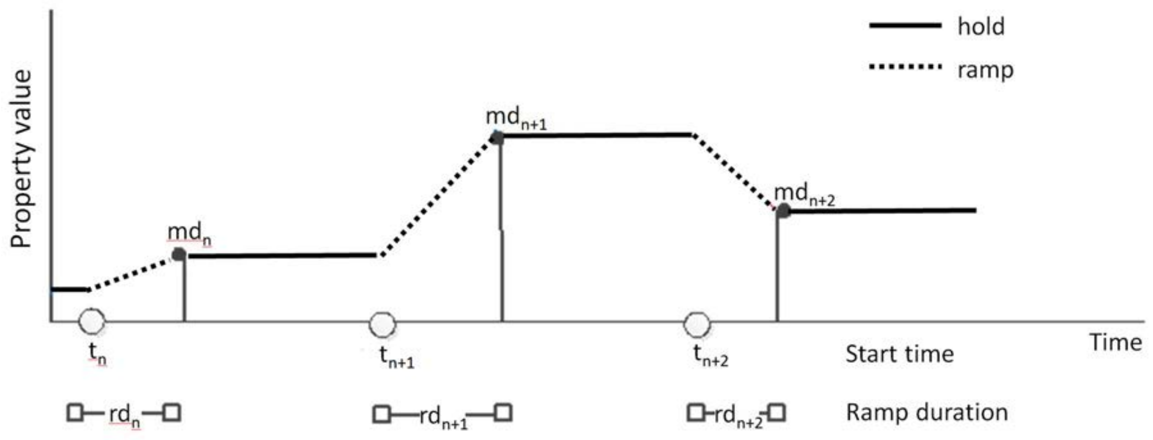
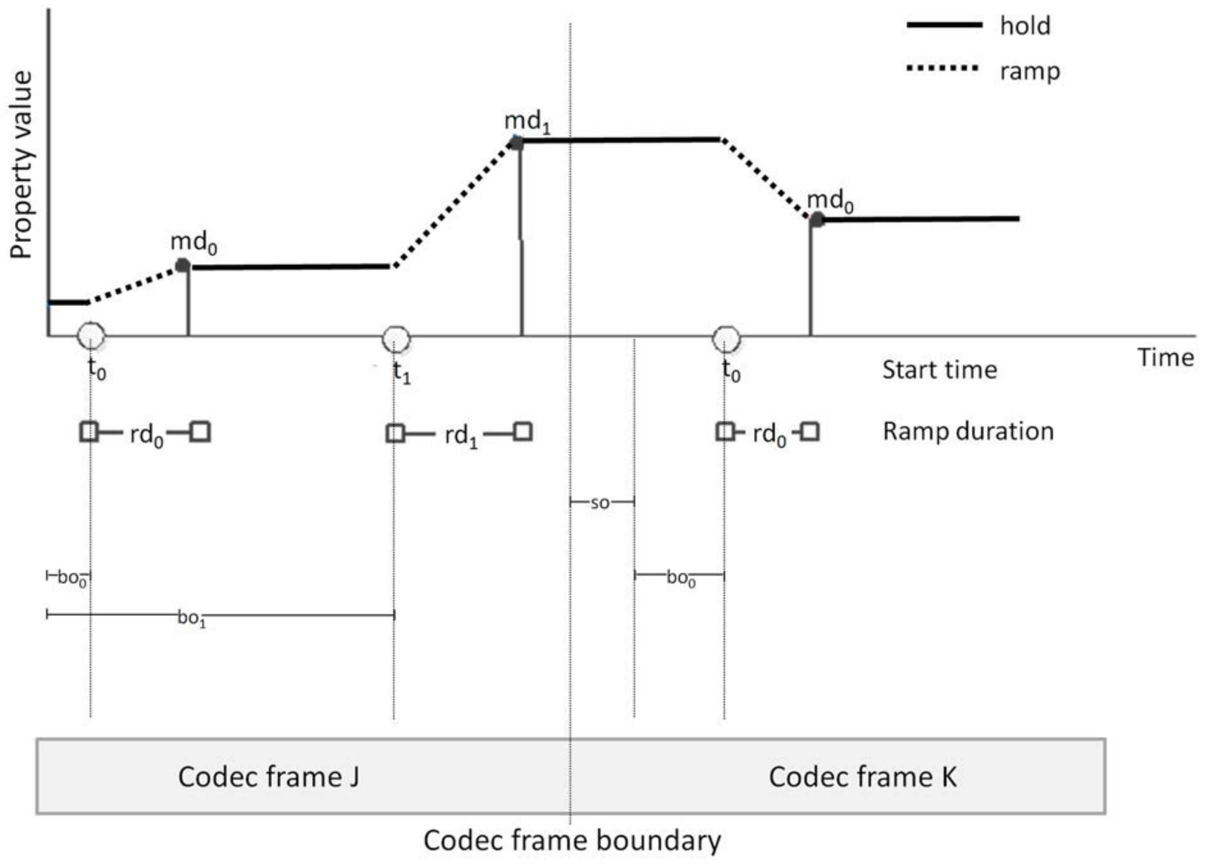


**Figure 4: Temporal context of the ramp duration for three sequential property updates**

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 10 lists the related bitstream elements and their reference.

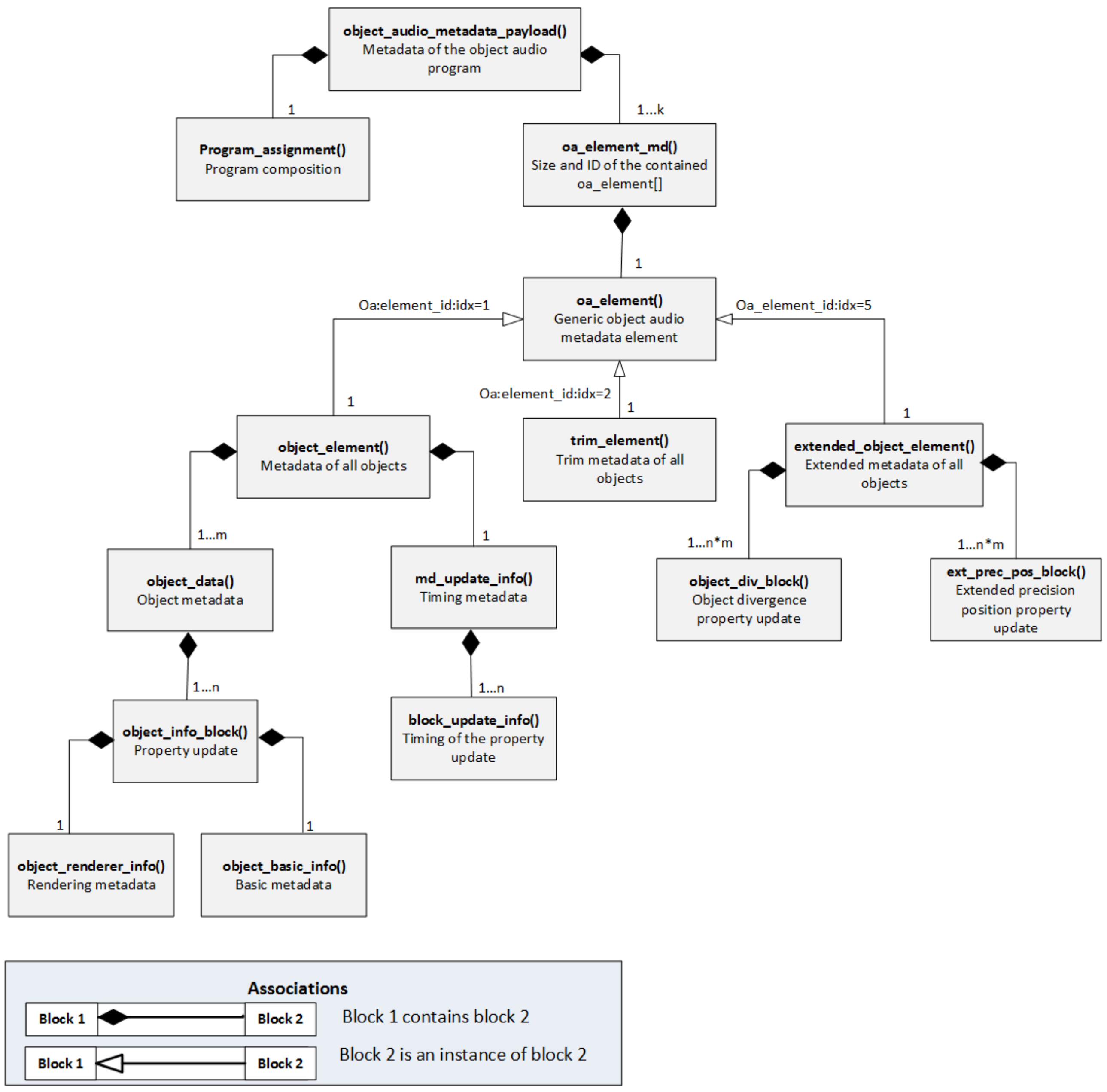
**Table 10: Property update timing related bitstream elements**

| 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.



**Quantity of blocks**

k- number of oa_element_md() and object_element() blocks, indicated by the oa_element_count element
m – number of object_data() blocks, indicated by the object_count element
n- number of object_data() and block_update_info() blocks, indicated by the obj_info_blocks_minus1 element

**Figure 6: OAMD bitstream structure**

Table 11 lists the blocks contained in the OAMD bitstream.

**Table 11: 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 | Clause 5.2.4 and clause 5.2.3 |
| object_render_info | Position, size, channel lock and zone constraints | Clause 5.2.1, clause 5.2.2, clause 5.2.5 and clause 5.2.6 |
| trim_element | Trim metadata | Clause 5.2.8 |
| extended_object_element | Property updates for the extended metadata (divergence and high precision position) | Clause 5.6.6.4.3, clause 5.6.6.4.4, clause 5.6.6.4.5 |
| obj_div_block | Divergence | Clause 5.2.7 |
| ext_prec_pos_block | High precision position | |

## 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 += (1 << n);
        num_group += 1;
      }
    }
  }
Syntax                                                                    No of bits
  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[]; ...................................................... 4
    if (content_description[3]) {
      b_bed_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[]; ........................................... 10
          }
          else {
            nonstd_bed_channel_assignment[]; .................................... 17
          }
        }
      }
    }
Syntax                                                                       No of bits
    if (content_description[2]) {
      intermediate_spatial_format_idx; ..................................... 3
    }
    if (content_description[1]) {
      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[0]) {
      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_reserved_data_not_present; .............................................. 1
  if (!reserved_data_not_present) {
    reserved; ............................................................... 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
  }
Syntax                                                                 No of bits
  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[] = {true, true};
  }
  else {
    object_basic_info[]; ........................................................... 2
  }
  if (object_basic_info[1]) {
    object_gain_idx; ............................................................... 2
    if (object_gain_idx == 0b10) {
      object_gain_bits; ............................................................ 6
    }
Syntax                                                                        No of bits
  }
  if (object_basic_info[0]) {
    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[] = {true, true, true, true};
  }
  else {
    obj_render_info[]; ....................................................... 4
  }
  if (obj_render_info[0]) {
    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[1]) {
    zone_constraints_idx; .................................................... 3
    b_enable_elevation; ...................................................... 1
  }
  if (obj_render_info[2]) {
    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
      }
Syntax                                                                  No of bits
    }
  }
  if (obj_render_info[3]) {
    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
}
```

## 5.5.12 trim_element

```
Syntax                                                                  No of bits
trim_element()
{
  warp_mode; ............................................................... 2
  reserved; ................................................................ 2
  global_trim_mode; ........................................................ 2
  if (global_trim_mode == 0b10) {
    for (cfg = 0; cfg < NUM_TRIM_CONFIGS; cfg++) {
      b_default_trim; ...................................................... 1
      if (b_default_trim == 0) {
        b_disable_trim; .................................................... 1
        if (b_disable_trim == 0) {
          trim_balance_presence[]; ......................................... 5
          if (trim_balance_presence[4]) {
            trim_centre; ................................................... 4
          }
          if (trim_balance_presence[3]) {
            trim_surround; ................................................. 4
          }
          if (trim_balance_presence[2]) {
            trim_height; ................................................... 4
          }
          if (trim_balance_presence[1]) {
            bal3D_Y_sign_tb_code; .......................................... 1
            bal3D_Y_amount_tb; ............................................. 4
          }
          if (trim_balance_presence[0]) {
            bal3D_Y_sign_lis_code; ......................................... 1
            bal3D_Y_amount_lis; ............................................ 4
          }
        }
      }
    }
  }
  b_disable_trim_per_obj; .................................................. 1
  if (b_disable_trim_per_obj) {
    for (obj = 0; obj < object_count; obj++) {
      b_disable_trim; ...................................................... 1
    }
  }
}
```

### 5.5.13 extended_object_element

**Syntax** **No of bits**

```
extended_object_element()
{
  b_obj_div_block; ............................................................ 1
  if (b_obj_div_block) {
    for (obj = 0; obj < object_count; obj++) {
      for (blk = 0; blk < num_obj_info_blocks; blk++) {
        obj_div_block(obj, b_obj_not_active, obj_type);
      }
    }
  }
  b_ext_prec_pos_block; ....................................................... 1
  if (b_ext_prec_pos_block) {
    for (obj = 0; obj < object_count; obj++) {
      for (blk = 0; blk < num_obj_info_blocks; blk++) {
        ext_prec_pos_block(obj, b_obj_not_active, obj_type);
      }
    }
  }
}
```

### 5.5.14 obj_div_block

**Syntax** **No of bits**

```
obj_div_block(obj_id, b_obj_not_active, obj_type)
{
  if (b_obj_not_active) {
    object_divergence = 0;
  }
  else {
    if (obj_type == DYNAMIC) {
      b_object_divergence; ........................................................ 1
      if (b_object_divergence) {
        object_div_mode; .......................................................... 2
        if (object_div_mode == 0) {
          object_div_table; ....................................................... 2
        }
        if (object_div_mode == 2 || object_div_mode == 3) {
          object_div_code; ........................................................ 6
        }
        else {
          object_divergence = 0;
        }
      }
    }
  }
}
```

### 5.5.15 ext_prec_pos_block

**Syntax** **No of bits**

```
ext_prec_pos_block(obj_id, b_obj_not_active, obj_type)
{
  if (b_obj_not_active) {
    b_ext_prec_pos = false;
  }
  else {
    if (obj_type == DYNAMIC) {
      b_ext_prec_pos; ............................................................. 1
```

| Syntax | No of bits |
|---|---|
| `      if (b_ext_prec_pos) {` | |
| `        `**`ext_prec_pos_presence[]`**`;` | 3 |
| `        if (ext_prec_pos_presence[2]) {` | |
| `          `**`ext_prec_pos3D_X`**`;` | 2 |
| `        }` | |
| `        if (ext_prec_pos_presence[1]) {` | |
| `          `**`ext_prec_pos3D_Y`**`;` | 2 |
| `        }` | |
| `        if (ext_prec_pos_presence[0]) {` | |
| `          `**`ext_prec_pos3D_Z`**`;` | 2 |
| `        }` | |
| `      }` | |
| `    }` | |
| `  }` | |
| `}` | |

## 5.6 Bitstream description

### 5.6.0 Object audio metadata content description

#### 5.6.0.1 oa_md_version_bits

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

#### 5.6.0.2 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.

#### 5.6.0.3 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.

#### 5.6.0.4 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.

#### 5.6.0.5 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 LFE channel (when true), or it contains a combination of one or more bed instances, intermediate spatial format and one or more dynamic objects (when false).

#### 5.6.0.6 b_lfe_present

The `b_lfe_present` flag indicates whether a bitstream that otherwise contains only dynamic objects also contains an LFE channel.

#### 5.6.0.7 content_description[]

The `content_description[]` array indicates the object types present in the bitstream.

Each element of the `content_description` array is a flag that indicates whether one or more objects of the assigned object type are present in the bitstream. Table 11a lists the object types for the `content_description[]` array elements.

**Table 11a: Object types for content_description array elements**

| `content_description[]` array element | assigned object type |
|---|---|
| 3 | object(s) with speaker-anchored coordinate(s) (bed objects) |
| 2 | intermediate spatial format (ISF) |
| 1 | object(s) with room-anchored or screen-anchored coordinates |
| 0 | reserved |

### 5.6.0.8 b_bed_chan_distribute

The `b_bed_chan_distribute` flag indicates whether bed channels (i.e. objects with a speaker-anchored position) are intended to be distributed to multiple appropriate loudspeakers.

### 5.6.0.9 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.

### 5.6.0.10 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: num_bed_instances = num_bed_instances_bits + 2. In this equation, the bits of `num_bed_instances_bits` are interpreted as an unsigned integer.

### 5.6.0.11 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 11b.

**Table 11b: Intermediate spatial format**

| `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.<br>NOTE 2: MULZ refers to the order in which audio signals for ISF objects occur. | | | |

#### 5.6.0.12 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.

### 5.6.1 Audio object properties

#### 5.6.1.1 Position

##### 5.6.1.1.1 Void

##### 5.6.1.1.2 Void

##### 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[]` array (if true) or by the `nonstd_bed_channel_assignment[]` array (when false).

##### 5.6.1.1.4 bed_channel_assignment[]

The `bed_channel_assignment[]` array of flags indicates the *bed_channel_assignment*.

The *bed_channel_assignment* is a list of channel labels. Each element of the `bed_channel_assignment[]` array is a flag that indicates whether the corresponding channel labels or pair of channel labels shall be contained in the *bed_channel_assignment* list.

Table 12 lists how elements of the `bed_channel_assignment[]` array relate to channel labels.

Channel labels in the *bed_channel_assignment* list shall be ordered like the `bed_channel_assignment[]` array elements, starting with bit 0 (if present). If the number of channel labels is 2, the channel labels shall be ordered as shown in column two.

**Table 12: Channel labels contained in the bed_channel_assignment list indicated by the bed_channel_assignment[] array elements**

| `bed_channel_assignment[]` array index | Speaker label(s) | Number of speaker labels |
|---|---|---|
| 9 | RC_L/RC_R | 2 |
| 8 | RC_C | 1 |
| 7 | RC_LFE | 1 |
| 6 | RC_LS/RC_RS | 2 |
| 5 | RC_LB/RC_RB | 2 |
| 4 | RC_TFL/RC_TFR | 2 |
| 3 | RC_TSL/RC_TSR | 2 |
| 2 | RC_TBL/RC_TBR | 2 |
| 1 | RC_LW/RC_RW | 2 |
| 0 | RC_LFE2 | 1 |

##### 5.6.1.1.5 nonstd_bed_channel_assignment[]

The `nonstd_bed_channel_assignment[]` array indicates the *bed_channel_assignment*.

The *bed_channel_assignment* is a list of channel labels. Each element of the `nonstd_bed_channel_assignment[]` array is a flag that indicates whether the corresponding channel label shall be contained in the *bed_channel_assignment* list.

Table 13 lists how elements of the `bed_channel_assignment[]` array relate to channel labels.

Channel labels in the *bed_channel_assignment* list shall be ordered like the `bed_channel_assignment[]` array elements, starting with index 0.

**Table 13: Channel labels contained in the bed_channel_assignment list indicated by the nonstd_bed_channel_assignment[] array elements**

| `non_std_bed_channel_assignment[]` array index | Channel label(s) |
|---|---|
| 16 | RC_L |
| 15 | RC_R |
| 14 | RC_C |
| 13 | RC_LFE |
| 12 | RC_LS |
| 11 | RC_RS |
| 10 | RC_LB |
| 9 | RC_RB |
| 8 | RC_TFL |
| 7 | RC_TFR |
| 6 | RC_TSL |
| 5 | RC_TSR |
| 4 | RC_TBL |
| 3 | RC_TBR |
| 2 | RC_LW |
| 1 | RC_RW |
| 0 | 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 14 specifies how the values of *pos3D_X*, *pos3D_Y* and *pos3D_Z* are signalled in the bitstream.

**Table 14: 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:

$$\mathrm{pos3D\_X} = \min\left(1; \frac{\mathrm{pos3D\_X\_bits}}{62} + \frac{\mathrm{ext\_prec\_pos3D\_X}}{62 \times 5}\right).$$

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

NOTE: The value of *ext_prec_pos3D_X* is specified in clause 5.6.6.4.3.

#### 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:

$$\text{pos3D\_Y} = \min\left(1; \frac{\text{pos3D\_Y\_bits}}{62} + \frac{\text{ext\_prec\_pos3D\_Y}}{62 \times 5}\right),$$

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

NOTE: The value of *ext_prec_pos3D_Y* is specified in clause 5.6.6.4.4.

#### 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:

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

#### 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:

$$\text{pos3D\_Z} = \text{pos3D\_Z\_sign} \times \frac{\text{pos3D\_Z\_bits}}{15} + \frac{\text{ext\_prec\_pos3D\_Z}}{15 \times 5}$$

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.

NOTE: The value of *ext_prec_pos3D_Z* is specified in clause 5.6.6.4.5.

#### 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 standard precision value of *pos3D_X* (differential coding).

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

$$\text{pos3D\_X} = \max\left(0; \min\left(1; \frac{\text{pos3D\_X\_bits}_{\text{prev}}}{62} + \frac{\text{diff\_pos3D\_X\_bits}}{62} + \frac{\text{ext\_prec\_pos3D\_X}}{62 \times 5}\right)\right)$$

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

The *pos3D_X_bits*$_{\text{prev}}$ value is the standard precision position value coded in 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 standard precision value of *pos3D_Y* (differential coding).

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

$$\text{pos3D\_Y} = \max\left(0; \min\left(1; \frac{\text{pos3D\_Y\_bits}_{\text{prev}}}{62} + \frac{\text{diff\_pos3D\_Y\_bits}}{62} + \frac{\text{ext\_prec\_pos3D\_Y}}{62 \times 5}\right)\right)$$

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

The *pos3D_Y_bits*$_{\text{prev}}$ value is the standard precision position value coded in 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:

$$\text{pos3D\_Z} = \max\left(-1; \min\left(1; \frac{\text{pos3D\_Z\_bits}_{\text{prev}}}{15} + \frac{\text{diff\_pos3D\_Z\_bits}}{15} + \frac{\text{ext\_prec\_pos3D\_Z}}{15 \times 5}\right)\right)$$

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

The *pos3D_Z_bits*$_{\text{prev}}$ value is the standard precision position value coded in 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 15.

**Table 15: 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 16.

**Table 16: 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 17 lists the values of *object_width*, *object_depth*, and *object_height*.

**Table 17: 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 18.

**Table 18: 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 19.

**Table 19: 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 20.

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

**Table 20: 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) |
| 67 | Reserved |

### 5.6.1.6.2 b_enable_elevation

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

Semantics for the `b_enable_elevation` element are listed in table 21.

**Table 21: Zone constraints indicated by b_enable_elevation**

| **b_enable_elevation** | **Zone constraints** |
|---|---|
| true | Top-Bottom zone included |
| false | Top-Bottom zone 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 22 lists the value of *sample_offset* indicated by the `sample_offset_code` element.

**Table 22: 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 23 lists the value of *sample_offset* indicated by the `sample_offset_idx` element.

**Table 23: 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 24 lists the value of *ramp_duration* indicated by the `ramp_duration_code` element:

**Table 24: 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 25 lists the value of *ramp_duration* indicated by the corresponding table index.

**Table 25: 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 Void

### 5.6.4 Additional metadata

#### 5.6.4.1 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:

reserved_data_size = reserved_data_size_bits + 1

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

### 5.6.4.2 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 | `trim_element` |
| 3 | reserved |
| 4 | reserved |
| 5 | `extended_object_element` |
| 6-15 | reserved |

### 5.6.4.3 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:

oa_element_size = oa_element_size_bits + 1

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

### 5.6.4.4 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. | |

### 5.6.4.5 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.

### 5.6.4.6 b_object_not_active

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

### 5.6.4.7 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<br>• *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 5.6.4.12) |

### 5.6.4.8 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 or is an ISF object.

Whether an object is contained in a bed or is and ISF object can be determined from the order of objects and from the OAMD content description, specified in clause 5.6.0.

The objects are ordered in the bitstream as follows:

1) Objects contained in a bed

2) ISF objects

3) Dynamic objects (not contained in a bed 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.

### 5.6.4.9 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<br>• *pos3D_Y*=0,5<br>• *pos3D_Z*=0<br>• *width*=0<br>• *depth*=0<br>• *height*=0<br>• `zone_constraints_idx`=0<br>• `b_enable_elevation`=true<br>• `b_use_screen_ref`=false<br>• `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 5.6.4.13: some values are signalled in the `object_render_info` block and some values are equal to the corresponding values of the previous metadata update |

#### 5.6.4.10 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.

#### 5.6.4.11 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.

#### 5.6.4.12 obj_basic_info[]

The `obj_basic_info[]` array indicates whether the `object_gain_idx` and `b_default_object_priority` elements are present in the bitstream.

Each element of the `obj_basic_info[]` array is a flag that indicates that a corresponding element is present in the bitstream. Table 30 lists the correspondence between bitstream element and elements of the `obj_basic_info[]` array.

**Table 30: Bitstream elements indicated by the obj_basic_info[] array**

| `obj_basic_info[]` array index | Corresponding bitstream element |
|---|---|
| 1 | `object_gain_idx` |
| 0 | `b_default_object_priority` |

#### 5.6.4.13 obj_render_info[]

The `obj_render_info[]` array indicates the bitstream elements contained in the `object_render_info` block.

Each element of the `obj_render_info[]` array is a flag that indicates whether corresponding elements are present in the bitstream. Table 31 lists the correspondence between bitstream elements and elements of the `obj_basic_info` array.

**Table 31: Bitstream elements indicated by the obj_render_info[] array elements**

| obj_render_info[] array index | Corresponding bitstream elements |
|---|---|
| 3 | 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 |
| 2 | zone_constraints_idx and b_enable_elevation |
| 1 | object_size_idx, object_size_bits, object_width_bits, object_depth_bits, and object_height_bits |
| 0 | b_object_use_screen_ref,screen_factor_bits, and depth_factor_idx |

#### 5.6.4.14 padding

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

### 5.6.5 Trim

#### 5.6.5.1 warp_mode

The warp_mode element is a codeword that indicates an object position adjustment before rendering as specified in table 32.

**Table 32: Object position adjustment indicated by warp_mode**

| warp_mode | Object position adjustment |
|---|---|
| 0b00 | No object position adjustment. |
| 0b01 | The *object_position_Y* value shall be multiplied by a factor of 2. |
| 0b1X | Reserved. |

#### 5.6.5.2 global_trim_mode

The global_trim_mode element is a codeword that indicates the trim mode as specified in table 33. The global trim mode is applicable to all trim configurations.

**Table 33: Global trim mode indicated by global_trim_mode**

| global_trim_mode | Global trim mode | Balance control |
|---|---|---|
| 0 | *default_trim* | Disabled |
| 1 | *disable_trim* | Disabled |
| 2 | *custom_trim* (see note) | Indicated by: • fb_balance_hb_direction • fb_balance_hb_amount • fb_balance_surr_direction • fb_balance_surr_amount |
| 3 | reserved | reserved |
| NOTE: The global trim mode may be overwritten by the per-trim-configuration elements b_default_trim and/or b_disable_trim. b_default_trim is described in clause 5.6.5.3 and b_disable_trim is described in clause 5.6.5.4 | | |

#### 5.6.5.3 b_default_trim

The b_default_trim flag indicates whether the *trim_mode* is set to *default_trim* for all objects in the corresponding trim configuration.

NOTE: If b_disable_trim_per_obj is true, the transmitted information to disable trim for individual objects may override the setting for individual objects in the trim configuration.

### 5.6.5.4 b_disable_trim

The `b_disable_trim` flag indicates whether the *trim_mode* is set to *disable_trim* for all objects in the corresponding trim configuration, or, if transmitted per object (by `b_disable_trim_per_obj`) for the corresponding object.

### 5.6.5.5 trim_balance_presence[]

If true, the Boolean elements of this array indicates that corresponding bitstream elements for trim and/or balance processing are transmitted.

Table 34 lists the correspondence between elements of `trim_balance_presence[]` and bitstream elements of the `trim_element`.

**Table 34: trim_balance_presence[] elements**

| **`trim_balance_presence`** **element index** | **Transmitted bitstream elements** |
|---|---|
| 4 | `trim_centre` |
| 3 | `trim_surround` |
| 2 | `trim_height` |
| 1 | `fb_balance_ohfl_direction`, `fb_balance_ohfl_amount` |
| 0 | `fb_balance_surr_direction`, `fb_balance_surr_amount` |

### 5.6.5.6 trim_centre

The `trim_centre` codeword indicates the *trim_centre* value as specified in table 35.

**Table 35: trim_centre**

| **`trim_centre`** | ***trim_centre*** **in dB** |
|---|---|
| 0 | +6 |
| 1 | +3 |
| 2 | +1,5 |
| 3 | +0,75 |
| 4 | -0,75 |
| 5 | -1,5 |
| 6 | -3 |
| 7 | -4,5 |
| 8 | -6 |
| 9 | -7,5 |
| 10 | -9 |
| 11 | -10,5 |
| 12 | -12 |
| 13 | -13,5 |
| 14 | -16 |
| 15 | -36 |

### 5.6.5.7 trim_surround

The `trim_surround` codeword indicates the *trim_surround* value as specified in table 36.

**Table 36: trim_surround**

| `trim_surround` | ***trim_surround* in dB** |
|---|---|
| 0 - 3 | reserved |
| 4 | -0,75 |
| 5 | -1,5 |
| 6 | -3 |
| 7 | -4,5 |
| 8 | -6 |
| 9 | -7,5 |
| 10 | -9 |
| 11 | -10,5 |
| 12 | -12 |
| 13 | -13,5 |
| 14 | -16 |
| 15 | -36 |

## 5.6.5.8 trim_height

The `trim_height` codeword indicates the *trim_height* value as specified in table 37.

**Table 37: trim_height**

| `trim_height` | ***trim_height* in dB** |
|---|---|
| 0 - 3 | reserved |
| 4 | -0,75 |
| 5 | -1,5 |
| 6 | -3 |
| 7 | -4,5 |
| 8 | -6 |
| 9 | -7,5 |
| 10 | -9 |
| 11 | -10,5 |
| 12 | -12 |
| 13 | -13,5 |
| 14 | -16 |
| 15 | -36 |

## 5.6.5.9 bal3D_Y_sign_tb_code

The `bal3D_Y_sign_tb_code` codeword indicates the direction of balance along the Y-axis to be applied in both the top and bottom planes as specified in table 38.

**Table 38: bal3D_Y_sign_tb indicated by bal3D_Y_sign_tb_code**

| **bal3D_Y_sign_tb_code** | **bal3D_Y_sign_tb** | **Meaning** |
|---|---|---|
| 0 | -1 | The balance is towards the front of the room. |
| 1 | 1 | The balance is towards the back of the room. |

## 5.6.5.10 bal3D_Y_amount_tb

The `bal3D_Y_amount_tb` codeword indicates the amount of balance along the Y-axis to be applied in both the top and bottom planes, determined as:

$$\text{bal3D\_Y\_tb} = \text{bal3D\_Y\_sign\_tb} \times \frac{\text{bal3D\_Y\_amount\_tb}+1}{16}.$$

NOTE: *bal3D_Y_sign_tb* is specified in clause 5.6.5.9.

#### 5.6.5.11 bal3D_Y_sign_lis_code

The `bal3D_Y_sign_lis_code` codeword indicates the direction of balance along the Y-axis to be applied in the listener plane (z = 0) as specified in table 39.

**Table 39: bal3D_Y_sign_lis indicated by bal3D_Y_sign_lis_code**

| bal3D_Y_sign_lis_code | bal3D_Y_sign_lis | Meaning |
|---|---|---|
| 0 | -1 | The balance is towards the front of the room. |
| 1 | 1 | The balance is towards the back of the room. |

#### 5.6.5.12 bal3D_Y_amount_lis

The `bal3D_Y_amount_lis` codeword indicates the amount of balance along the Y-axis to be applied in the listener plane (z = 0), determined as:

$$\text{bal3D\_Y\_lis} = \text{bal3D\_Y\_sign\_lis} \times \frac{\text{bal3D\_Y\_amount\_lis} + 1}{16}.$$

NOTE: *bal3D_Y_sign_lis* is specified in clause 5.6.5.11.

## 5.6.6 extended_object_element

### 5.6.6.1 b_obj_div_block

The `b_obj_div_block` flag indicates whether an `obj_div_block` element is present in the bitstream per metadata update block.

### 5.6.6.2 b_ext_prec_pos_block

The `b_ext_prec_pos_block` flag indicates whether an `ext_prec_pos_block` element is present in the bitstream per metadata update block.

### 5.6.6.3 obj_div_block

#### 5.6.6.3.1 b_object_divergence

The `b_object_divergence` flag indicates whether object divergence metadata is present in the bitstream.

#### 5.6.6.3.2 object_div_mode

The `object_div_mode` element indicates how the value of *object_divergence* is transmitted, as shown in table 40.

**Table 40: object_div_mode**

| object_div_mode | Description |
|---|---|
| 0 | indicated by `object_div_table`. |
| 1 | reuse *object_divergence* as transmitted in the previous `obj_info_block`. |
| 2 | indicated by `object_div_code`. |
| 3 | Reserved. |

#### 5.6.6.3.3 object_div_table

The `object_div_table` element indicates the value of *object_divergence* as shown in table 41.

**Table 41: object_div_table**

| `object_div_table` | ***object_divergence*** |
|---|---|
| 0 | 0,500755 |
| 1 | 0,608529 |
| 2 | 0,704833 |
| 3 | 1,0 |

### 5.6.6.3.4 object_div_code

The `object_div_code` indicates the value of *object_divergence* as shown in table 42.

**Table 42: object_div_code**

| `object_div_code` | ***object_divergence*** |
|---|---|
| 0 | reserved |
| 1 | 0 |
| 2 | 0,004026 |
| 3 | 0,00716 |
| 4 | 0,012731 |
| 5 | 0,020173 |
| 6 | 0,028485 |
| 7 | 0,04021 |
| 8 | 0,050582 |
| 9 | 0,063601 |
| 10 | 0,079914 |
| 11 | 0,100299 |
| 12 | 0,125666 |
| 13 | 0,140532 |
| 14 | 0,157027 |
| 15 | 0,175282 |
| 16 | 0,195417 |
| 17 | 0,217536 |
| 18 | 0,241718 |
| 19 | 0,268002 |
| 20 | 0,296377 |
| 21 | 0,326766 |
| 22 | 0,359017 |
| 23 | 0,392895 |
| 24 | 0,428081 |
| 25 | 0,464184 |
| 26 | 0,500755 |
| 27 | 0,537316 |
| 28 | 0,573389 |
| 29 | 0,608529 |
| 30 | 0,642346 |
| 31 | 0,674524 |
| 32 | 0,704833 |
| 33 | 0,733123 |
| 34 | 0,75932 |
| 35 | 0,783416 |
| 36 | 0,805451 |
| 37 | 0,825506 |
| 38 | 0,843686 |
| 39 | 0,860112 |
| 40 | 0,874914 |
| 41 | 0,888222 |
| 42 | 0,900168 |
| 43 | 0,910875 |
| 44 | 0,920461 |
| 45 | 0,929035 |
| 46 | 0,936698 |
| 47 | 0,943544 |
| 48 | 0,949656 |
| 49 | 0,955112 |
| 50 | 0,95998 |
| 51 | 0,964322 |
| 52 | 0,968195 |
| 53 | 0,974729 |
| 54 | 0,979923 |
| 55 | 0,98405 |
| 56 | 0,98733 |
| 57 | 0,989935 |
| 58 | 0,992874 |
| 59 | 0,994955 |
| 60 | 0,996817 |
| 61 | 0,99821 |
| 62 | 0,998993 |
| 63 | 1 |

## 5.6.6.4 ext_prec_pos_block

### 5.6.6.4.1 b_ext_prec_pos

The `b_ext_prec_pos` flag indicates whether extended precision position metadata is present in the bitstream.

### 5.6.6.4.2 ext_prec_pos_presence[]

If true, the Boolean elements of this array indicates that corresponding extended precision metadata elements are transmitted. If any extended precision metadata element is not transmitted its value shall be 0.

Table 43 lists the correspondence between elements of `ext_prec_pos_presence[]` and bitstream elements of the `ext_prec_pos_block` element.

**Table 43: extp_pos_presence elements**

| **`ext_prec_pos_presence`** **index** | **Transmitted bitstream elements** |
|---|---|
| 2 | `ext_prec_pos3D_X` |
| 1 | `ext_prec_pos3D_Y` |
| 0 | `ext_prec_pos3D_Z` |

### 5.6.6.4.3 ext_prec_pos3D_X

The `ext_prec_pos3D_X` indicates the extended precision position value to be used for calculation of *pos3D_X* as specified in clause 5.6.1.1.8 and clause 5.6.1.1.12. The extended precision value is shown in table 44.

**Table 44: ext_prec_pos3D_X**

| **`ext_prec_pos3D_X`** | **Semantics** |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | -1 |
| 3 | -2 |

### 5.6.6.4.4 ext_prec_pos3D_Y

The `ext_prec_pos3D_Y` indicates the extended precision value to be used for calculation of *pos3D_Y* as specified in clause 5.6.1.1.9 and clause 5.6.1.1.13. The extended precision value is shown in table 45.

**Table 45: ext_prec_pos3D_Y**

| **`ext_prec_pos3D_Y`** | **Semantics** |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | -1 |
| 3 | -2 |

### 5.6.6.4.5 ext_prec_pos3D_Z

The `ext_prec_pos3D_Z` indicates the extended precision value to be used for calculation of *pos3D_Z* as specified in clause 5.6.1.1.11 and clause 5.6.1.1.14. The extended precision value is shown in table 46.

**Table 46: ext_prec_pos3D_Z**

| **`ext_prec_pos3D_Z`** | **Semantics** |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | -1 |
| 3 | -2 |

# 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 OBA essences from a channel-based E-AC-3 bitstream.

The JOC decoder performs a quadrature mirror filter bank 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 quadrature mirror filter bank 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++) {
              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 47 specifies the JOC downmix configuration indicated by `joc_dmx_config_idx`.

**Table 47: 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, Lb, Rb |
| 2 | 5.X + 2 | L, R, C, (LFE), Ls, Rs, Tfl, Tfr |
| 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, Tfl, Tfr |
| 5...7 | reserved | |
| NOTE: If the LFE channel is present, this audio 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 48.

**Table 48: 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:

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

The maximum value for `joc_num_objects_bits` shall be 15.

#### 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 49 specifies the extensional configuration data indicated by `joc_ext_config_idx`.

**Table 49: 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 50 specifies the value of *joc_num_bands* indicated by `joc_num_bands_idx`.

**Table 50: 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 51 specifies the value of *joc_num_quant* indicated by `joc_num_quant_idx`.

**Table 51: 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 52.

**Table 52: 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 quadrature mirror filter bank 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 53 lists the input channel indicated by `joc_channel_idx` for the two different cases `joc_num_channels` = 5 and `joc_num_channels` = 7.

**Table 53: 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 Surround |
| 4 | Right Surround | Right Surround |
| 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* quadrature mirror filter bank matrices are input to the JOC decoder.

$\mathbf{Qin}_{\mathrm{JOC},(1,\,2,\,\ldots,\,\mathrm{joc\_num\_channels})}$
*joc_num_channels* complex valued quadrature mirror filter bank 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* quadrature mirror filter bank matrices are output of the JOC decoder.

$\mathbf{Qout}_{\mathrm{JOC},(1,\,2,\,\ldots,\,\mathrm{joc\_num\_objects})}$
*joc_num_objects* complex valued quadrature mirror filter bank matrices corresponding to the number of output objects to be reconstructed by the JOC decoder.

NOTE: The $\mathbf{Qout}_{\mathrm{JOC}}$ 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 quadrature mirror filter bank 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 quadrature mirror filter bank subbands to JOC parameter bands is listed in table 54. The columns specify different mappings depending on the value of `joc_num_bands` (1 to 23).

**Table 54: Mapping of quadrature mirror filter bank 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(s
b)] +
                                                    (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 quadrature mirror filter bank 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 quadrature mirror filter bank analysis/synthesis transform pair can be employed.

The quadrature mirror filter bank 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 quadrature mirror filter bank analysis.

The quadrature mirror filter bank 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 $Q = M \cdot u$, where the following equation applies: $M_{k,j} = \exp\left(\frac{i \times \pi \times (k + {}^1/_2) \times (j - {}^1/_2)}{n}\right), \begin{cases} 0 \le k < n \\ 0 \le j < 2 \times n \end{cases}.$

**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: $\mathrm{N}_{\mathrm{j,k}} = \frac{1}{\mathrm{n}} \times \exp\left(\frac{\mathrm{i}\times\pi\times\left(\mathrm{k}+{}^{1}/_{2}\right)\times\left(\mathrm{j}\text{-}2\times\mathrm{n}+{}^{1}/_{2}\right)}{\mathrm{n}}\right), \begin{cases} 0 \le k < n \\ 0 \le j < 2 \times n \end{cases}$. 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 quadrature mirror filter bank is configured by the number *n* of QMF subband and the filter length *n_qmf_length*.

The quadrature mirror filter bank 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_103420v010201p0.zip contains the coefficients.

# 8 Enhanced AC-3 decoding

## 8.1 Requirements on Enhanced AC-3 decoding

To decode OBA 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 the independent substream and, if present, all the dependent substreams, as specified in ETSI TS 102 366 [1], clause E.2.8.2, and pass the decoded audio 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.

If the enhanced AC-3 bitstreams contains one or more dependent substreams, the EMDF container encompassing the payloads for OAMD and JOC side information shall be carried in the last dependent substream of the bitstream.

Table 55 lists normative requirements for the EMDF payloads.

**Table 55: Payload restrictions for OAMD and JOC**

| | **OAMD** | **JOC** |
|---|---|---|
| payload_id | 11 | 14 |
| frequency | $^{1}/_{\text{frame}}$ | $^{1}/_{\text{frame}}$ |

Table 56 lists normative requirements for the EMDF payload configuration data, as specified in ETSI TS 102 366 [1], clause H.2.2.3.

**Table 56: 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 |

## 8.3 Extensions contained in the addbsi bitstream field

### 8.3.1 Syntax

For bitstreams conforming to the present document, the `addbsi` field in the same substream as the EMDF container encompassing the payloads for OAMD and JOC side information contains the following elements in given order:

1) reserved - 7 bits

2) `flag_ec3_extension_type_a` - 1 bit

3) `complexity_index_type_a` - 8 bit

NOTE: The `addbsi` field is defined in ETSI TS 102 366 [1], clauses 4.3.2 and 4.4.2.31.

## 8.3.2 Semantics

### 8.3.2.1 flag_ec3_extension_type_a

The element `flag_ec3_extension_type_a` is a flag that, if set to true, indicates the enhanced AC-3 extension as defined in the present document.

### 8.3.2.2 complexity_index_type_a

The element `complexity_index_type_a` is an unsigned integer that indicates the decoding complexity of the enhanced AC-3 extension defined in the present document. The value of this field shall be equal to the total number of bed objects, ISF objects and dynamic objects indicated by the parameters in the `program_assignment` section of the object audio metadata payload. The maximum value of this field shall be 16.

# 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 (note 2) |
| Rs | Surround, Back/Side (note 2) |
| Lb | Back |
| Rb | Back |
| Tfl | Top-Bottom |
| Tfr | Top-Bottom |
| Tbl | Top-Bottom |
| Tbr | Top-Bottom |
| Tsl | Top-Bottom |
| Tsr | Top-Bottom |
| Tfc | Top-Bottom |
| Tbc | Top-Bottom |
| Tc | Top-Bottom |
| Cb | Surround, Back |
| Lw | Side |
| Rw | Side |
| Lscr | Screen |
| Rscr | Screen |
| NOTE 1: The centre-and-back zone is a union of the back zone with the centre speaker.<br>NOTE 2: Depending on the representation speaker configuration: Back for 5.X(.X) or less, Side otherwise. | |

# Annex B (informative): Conversion to ADM format

## B.1 Introduction

The present annex specifies a conversion to ADM Recommendation ITU-R BS.2076 [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 audio object (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 3: 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 4: 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 5: 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 Audio 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" | $2 \times (\min(0; \max(1; x)) - 0{,}5)$ |
| position | coordinate="Y" | $2 \times (0{,}5 - \min(0; \max(1; y)))$ |
| position | coordinate="Z" | $\min(0; \max(1; z))$ |
| 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" | $2 \times$ 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 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 | RoomCentricLeftSurround |
| Right Surround (Rs) | AC_00011006 | RoomCentricRightSurround |
| Left Back (Lb) | AC_00011007 | RoomCentricLeftBack |
| Right Back (Rb) | AC_00011008 | RoomCentricRightBack |
| Top Front Left (Tfl) | AC_00011009 | RoomCentricTopFrontLeft |
| Top Front Right (Tfr) | AC_0001100a | RoomCentricTopFrontRight |
| Top Side Left (Tsl) | AC_0001100b | RoomCentricTopSideLeft |
| Top Side Right (Tsr) | AC_0001100c | RoomCentricTopSideRight |
| Top Back Left (Tbl) | AC_0001100d | RoomCentricTopBackLeft |
| Top Back Right (Tbr) | AC_0001100e | RoomCentricTopBackRight |
| 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.<br>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 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_LS | -1 | 0 | 0 |
| Right Surround (Rs) | RC_RS | 1 | 0 | 0 |
| Left Back (Lb) | RC_LB | -1 | -1 | 0 |
| Right Back (Rb) | RC_RB | 1 | -1 | 0 |
| Top Front Left (Tfl) | RC_TFL | -1 | 1 | 1 |
| Top Front Right (Tfr) | RC_TFR | 1 | 1 | 1 |
| Top Side Left (Tsl) | RC_TSL | -1 | 0 | 1 |
| Top Side Right (Tsr) | RC_TSR | 1 | 0 | 1 |
| Top Back Left (Tbl) | RC_TBL | -1 | -1 | 1 |
| Top Back Right (Tbr) | RC_TBR | 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×priority) |

## B.2.4 Gain

The gain value should be applied to the object audio 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'<br>• 'ZM2_Right' |
| All horizontal zones excluded, except centre-and-back zone | 4 | • 'ZM3_ScreenLeft'<br>• ''ZM3_SideLeft'<br>• ''ZM3_ScreenRight'<br>• ''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'<br>• '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: HH = $\left\lfloor\frac{\text{seconds}}{3\,600}\right\rfloor$, MM = $mod\left(\left\lfloor\frac{\text{seconds}}{60}\right\rfloor, 60\right)$, SS.sssss = $mod(\text{seconds}, 60)$.<br>NOTE 2: $f_S$ is the sampling frequency of the audio essence.<br>NOTE 3: start_time$_{\text{subseq}}$ is the *start_time* value of the subsequent metadata update. | | |

# Annex C (normative): Bit streams in the ISO based media file format

## C.1 Introduction

Bit streams conforming to the present document are stored in enhanced AC-3 tracks and the MIME codecs parameter is *ec-3* as specified in ETSI TS 102 366 [1].

## C.2 General

Media tracks shall conform to the enhanced AC-3 track definition in ETSI TS 102 366 [1], clause F.1 and to the enhanced AC-3 sample definition in ETSI TS 102 366 [1], clause F.2.

The syntax and semantics of the *EC3SpecificBox* that shall be used for bit streams conform to the present document is defined in clause C.3.

NOTE: Decoders conforming to ETSI TS 102 366 [1] can parse the *EC3SpecificBox* as defined in clause C.3. Syntax and semantics are identical, except added bits in the end, which decoders conform to ETSI TS 102 366 [1] will skip as reserved bits.

## C.3 EC3SpecificBox

### C.3.1 EC3SpecificBox

**Syntax** **No of bits**

```
EC3SpecificBox()
{
  BoxHeader.Size; ..................................................... 32
  BoxHeader.Type; ..................................................... 32
  data_rate; .......................................................... 13
  num_ind_sub; ......................................................... 3
  for (i = 0; i < num_ind_sub + 1; i++) {
    fscod; ............................................................. 2
    bsid; .............................................................. 5
    reserved; .......................................................... 1
    asvc; .............................................................. 1
    bsmod; ............................................................. 3
    acmod; ............................................................. 3
    lfeon; ............................................................. 1
    reserved; .......................................................... 3
    num_dep_sub; ....................................................... 4
    if (num_dep_sub > 0) {
      chan_loc; ........................................................ 9
    }
    else {
      reserved; ........................................................ 1
    }
  }
  reserved; ............................................................ 7
  flag_ec3_extension_type_a; ........................................... 1
  complexity_index_type_a; ............................................. 8
  reserved; ........................................................ n * 8
}
```

NOTE: *n* is the number of bytes to complete the size of the box given by `BoxHeader.Size`.

## C.3.2 Semantics

### C.3.2.1 General

Semantics of the contained elements are described in ETSI TS 102 366 [1], clause F.6.2.

### C.3.2.2 flag_ec3_extension_type_a

The element `flag_ec3_extension_type_a` is a flag that, if set to true, indicates the enhanced AC-3 extension as defined in the present document.

### C.3.2.3 complexity_index_type_a

The element `complexity_index_type_a` is an unsigned integer that indicates the decoding complexity of the enhanced AC-3 extension defined in the present document. The value of this field shall be equal to the total number of bed objects, ISF objects and dynamic objects indicated by the parameters in the `program_assignment` section of the object audio metadata payload. The maximum value of this field shall be 16.

# Annex D (normative): Bit streams in MPEG DASH

## D.1 Introduction

This annex describes the requirements and recommendations for delivering streams using the MPEG Dynamic Adaptive Streaming over HTTP (DASH) standard (see ISO/IEC 23009-1 [3]) in conjunction with the ISO base media file format, specifically referencing audio streams within the MPEG DASH media presentation description file.

## D.2 Media Presentation Description (MPD)

### D.2.1 General

Media Presentation Description files compliant to the present document shall conform to the requirements on MPD for enhanced AC-3 bitstreams defined in ETSI TS 102 366 [1], clause I.1. The value of the *@codecs* attribute is *ec-3*.

Additionally, the following applies:

- for signalling presence of the enhanced AC-3 extension as defined in the present document a DASH supplemental property descriptor as specified in clause D.2.2.1 should be used, and
- for signalling complexity of the enhanced AC-3 extension as defined in the present document a DASH supplemental property descriptor as specified in clause D.2.2.2 should be used.

### D.2.2 Descriptors

#### D.2.2.1 Extension type

Presence of the enhanced AC-3 extension as defined in the present document is signalled with a DASH supplemental property descriptor using the *schemeIdUri* `tag:dolby.com,2018:dash:EC3_ExtensionType:2018`.

The value of this DASH descriptor shall be the three character string *JOC*.

#### D.2.2.2 Extension complexity index

Complexity of the enhanced AC-3 extension as defined in the present document is signalled with a DASH supplemental property descriptor using the *schemeIdUri* `tag:dolby.com,2018:dash:EC3_ExtensionComplexityIndex:2018`.

The value of this DASH descriptor shall be decimal representation of the eight-bit element `complexity_index_type_a` in the `EC3SpecificBox` of an enhanced AC-3 audio track.

### D.2.3 Example Media Presentation Description

```
    <?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:dolby="http://www.dolby.com/ns/online/DASH" xmlns="urn:mpeg:dash:schema:mpd:201
1"
    xsi:schemaLocation="urn:mpeg:dash:schema:mpd:2011 DASH-MPD.xsd"
    type="dynamic"
    minimumUpdatePeriod="PT2S"
    timeShiftBufferDepth="PT30M"
    availabilityStartTime="2011-12-25T12:30:00"
    publishTime="2011-12-25T12:30:00"
    minBufferTime="PT4S"
    profiles="urn:mpeg:dash:profile:isoff-live:2011">
    <BaseURL>http://cdn1.example.com/</BaseURL>
    <BaseURL>http://cdn2.example.com/</BaseURL>
    <Period id="2">
        <!-- Video -->
        <AdaptationSet mimeType="video/mp4" codecs="avc1.4D401F" frameRate="30000/1001"
            segmentAlignment="true" startWithSAP="1">
            <BaseURL>video/</BaseURL>
            <SegmentTemplate timescale="90000" media="$Bandwidth$/$Number$.m4s" initializ
ation="$Bandwidth$/0.mp4">
                <SegmentTimeline>
                    <S t="0" d="180180" r="12"/>
                </SegmentTimeline>
            </SegmentTemplate>
            <Representation id="v0" width="320" height="240" bandwidth="250000" />
            <Representation id="v1" width="640" height="480" bandwidth="500000" />
            <Representation id="v2" width="960" height="720" bandwidth="1000000" />
        </AdaptationSet>

        <!-- Object-based immersive audio AdaptationSet -->
        <AdaptationSet mimeType="audio/mp4" codecs="ec-
3" lang="en" segmentAlignment="true" startWithSAP="1">
            <Role schemeIdUri="urn:mpeg:dash:role:2011" value="main" />
            <SegmentTemplate timescale="48000" media="audio/en_main/$Bandwidth$/$Number$.
m4s"
                initialization="audio/en_main/$Bandwidth$/0.mp4">
                <SegmentTimeline>
                    <S t="0" d="96000" r="11"/>
                </SegmentTimeline>
            </SegmentTemplate>
            <Representation id="a1" bandwidth="256000">
                <AudioChannelConfiguration schemeIdUri="urn:mpeg:mpegB:cicp:ChannelConfig
uration" value="6"/>
                <!-- alternative
                <AudioChannelConfiguration schemeIdUri="tag:dolby.com,2014:dash:audio_cha
nnel_configuration:2011" value="F801"/>
                -->
                <SupplementalProperty schemeIdUri="tag:dolby.com,2018:dash:EC3_ExtensionT
ype:2018" value="JOC"/>
                <SupplementalProperty schemeIdUri="tag:dolby.com,2018:dash:EC3_ExtensionC
omplexityIndex:2018" value="12"/>
            </Representation>
        </AdaptationSet>
    </Period>
</MPD>
```

# Annex E (normative): Object-based audio Common Media Application Format (CMAF) profiles

## E.1 Introduction

This clause defines a CMAF profile with normative provisions if files written according to annex C of the present document are to be compliant with ISO/IEC 23000-19 [2].

## E.2 Media tracks

Media tracks shall conform to ISO/IEC 23000-19 [2], sections 6, 7, 8, 10.1 and 10.2.

Additionally, Media tracks shall conform to ETSI TS 102 366 [1], clauses J.1.3, J.1.4, and J.1.5.

## E.3 Elementary streams

The elementary stream shall contain exactly one independent substream (i.e. I0) and may contain one dependent substream (D0), but shall not contain more than one dependent substream.

The `acmod` and possibly present `chanmap` parameter shall indicate one of the channel configurations listed as downmix configurations in table 47.

The `numblkscod` parameter shall always be 3, indicating 6 audio blocks per frame.

## E.4 Switching sets

common media application format switching sets shall conform to ETSI TS 102 366 [1], clause J.2.

## E.5 Core object-based audio media profile

The *FileTypeBox* compatibility brand shall be *ceao* and should be used to indicate media tracks that conform to this media profile.

# History

| Document history | | |
|---|---|---|
| V1.1.1 | July 2016 | Publication |
| V1.2.1 | October 2018 | Publication |
| | | |
| | | |
| | | |