ETSI TS 103 190-2 V1.3.1 (2025-07)

TECHNICAL SPECIFICATION

Digital Audio Compression (AC-4) Standard;
Part 2: Immersive and personalized audio

Reference

RTS/JTC-110-2

Keywords

audio, broadcasting, codec, content, digital, distribution

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 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI IPR online database.

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

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

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 present document is part 2 of a multi-part deliverable covering the Digital Audio Compression (AC-4), as identified below:

Part 1: "Channel based coding";

Part 2: "Immersive and personalized audio".

The symbolic source code for tables referenced throughout the present document is contained in archive ts_10319002v010203p0.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

AC-4 ETSI TS 103 190-1 [1] provides a bit-rate-efficient coding scheme for common broadcast and broadband delivery environment use cases. ETSI TS 103 190-1 [1] also introduced system integration features in order to address particular challenges of modern media distribution, all with the flexibility to support future audio experiences:

- **Inclusive accessibility:**
  - Provides the same high quality of experience for dialogue intelligibility, video description, and multiple languages as is provided by the main service.
- **Advanced loudness and dynamic range control:**
  - Eliminating the need for expensive, stand-alone and single-ended real-time processing. AC-4 provides a fully-integrated and automated solution that ensures any program source meets regulatory needs while intelligently protecting sources that have been previously produced with regulatory needs in mind.
- **Frame alignment between coded audio and video:**
  - Greatly simplifies audio and video time base correction (A/V sync management) in the compressed domain for contribution and distribution applications.
- **Built-in robustness:**
  - Enables adaptive streaming and advertisement insertions, switching bit rate and channel configuration without audible artefacts.

The present document extends the AC-4 codec to a number of new use cases relevant for next-generation audio services:

- **Audio personalization:**
  - A foundation for new business opportunities, allowing listeners to tailor the content to their own preference with additional audio elements and compositional control.
- **Increased immersiveness with surround sound in all three dimensions:**
  - Advanced techniques maintain immersion across a variety of common speaker layouts and environments (including headphones), and future-proof content for later generations.
- **Enhanced adaptability:**
  - Ensures that playback can provide the best appropriate experience across a wide range of devices and applications for today and tomorrow.

Structure of the present document

The present document is structured as follows:

- Clause 4 provides an introduction to immersive audio and personalized audio, and specifies how to create an AC-4 decoder.
- Clause 5 specifies the algorithmic details for the various tools used in the AC-4 decoder.
- Clause 6.2 specifies the details of the AC-4 bitstream syntax.
- Clause 6.3 interprets bits into values used elsewhere in the present document.

1 Scope

The present document specifies a coded representation (a bitstream) of audio information, and specifies the decoding process. The coded representation specified herein is suitable for use in digital audio transmission and storage applications.

Building on the technology specified in ETSI TS 103 190-1 [1], the present document specifies additional functionality that can be used for immersive, personalized, or other advanced playback experiences. Additional representations can be included, targeting individual listener groups or applications (providing the possibility of different audio experience settings in addition to those specified in ETSI TS 103 190-1 [1]).

The present document does not specify an object audio renderer, which would be needed to decode object-based audio to a channel-based representation. A reference object audio renderer that may be used with the technology specified in the present document can be found in [i.2].

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 which are not found to be publicly available in the expected location might be found in the ETSI docbox.

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 necessary for the application of the present document.

[1] ETSI TS 103 190-1: "Digital Audio Compression (AC-4) Standard; Part 1: Channel based coding".

[2] ISO/IEC 10646: "Information technology — Universal Coded Character Set (UCS)".

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

[4] IETF RFC 6381: "The 'Codecs' and 'Profiles' Parameters for "Bucket" Media Types".

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

[6] ISO/IEC 23000-19: "Common Media Application Format for Segmented Media".

[7] IETF BCP 47: "Tags for Identifying Languages".

2.2 Informative references

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

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

The following referenced documents may be useful in implementing an ETSI deliverable or add to the reader's understanding, but are not required for conformance to the present document.

[i.1] Recommendation ITU-R BS.2051-2: "Advanced sound system for programme production".

[i.2] ETSI TS 103 448: "AC-4 Object Audio Renderer for Consumer Use".

[i.3] Recommendation EBU R 128: "Loudness normalisation and permitted maximum level of audio signals".

[i.4] Supplementary information EBU Tech 3344: "Guidelines for distribution and reproduction in accordance with EBU R 128".

[i.5] ISO/IEC 23091-1:2018: "Information technology -- Coding-independent code points — Part 1: Systems".

[i.6] ISO/IEC 23091-3:2018: "Information technology — Coding-independent code points — Part 3: Audio".

[i.7] ISO 639-2:1998: "Codes for the representation of names of languages — Part 2: Alpha-3 code".

[i.8] CTA-861-I: "A DTV Profile for Uncompressed High Speed Digital Interfaces".

[i.9] ISO/IEC 14496-14:2020: "Information technology — Coding of audio-visual objects — Part 14: MP4 file format".

[i.10] ETSI TS 126 244: "Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); LTE; 5G; Transparent end-to-end packet switched streaming service (PSS); 3GPP file format (3GP) (3GPP TS 26.244)".

[i.11] ETSI TS 102 366: "Digital Audio Compression (AC-3, Enhanced AC-3) Standard".

[i.12] IETF RFC 2045: "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies".

[i.13] ISO/IEC 13818-1: "Information technology — Generic coding of moving pictures and associated audio information — Part 1: Systems".

3 Definition of terms, symbols, abbreviations and conventions

3.1 Terms

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

3rd Generation Partnership Project: collaboration of telecommunications standard development organizations that produces reports and specifications for defining cellular telecommunications network technologies

AC-4 sync frame: optional bitstream container structure encapsulating AC-4 codec frames, used to provide synchronization and robustness against transmission errors

NOTE: This container structure may be used when the transmission system does not include information about the framing of AC-4.

accessibility: features that make a program available to people with disabilities

advanced coupling: parametric coding tool for AC-4 in the Quadrature Mirror Filter (QMF) domain, used to improve coding efficiency for multichannel and stereo content

advanced joint channel coding: parametric coding tool for AC-4 in the Quadrature Mirror Filter (QMF) domain, used to efficiently code channel-based immersive audio content

advanced joint object coding: parametric coding tool for AC-4 in the Quadrature Mirror Filter (QMF) domain, used to efficiently code object-based audio content

advanced spectral extension: parametric coding tool for AC-4 in the Quadrature Mirror Filter (QMF) domain, used to efficiently code high frequency content

alternative presentation: presentation that supports the personalized audio use-case by providing alternative object properties to be used in combination with common object essences

associated audio: audio program component designed to be presented together with at least one main audio program component to provide accessibility features or additional information, for example a director's commentary or an audio description for listeners with a visual impairment

associated audio substream: substream that carries associated audio

audio element: smallest addressable unit of an audio program, consisting of one or more audio signals and associated audio metadata

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

audio preselection: set of references to audio program components that represents a selectable version of the audio program for simultaneous decoding

audio program: audio part of a program

audio program component: set of audio elements characterized by an audio program component type and, optionally, a language

NOTE: music and effects cannot be differentiated by language

audio program component type: classifier for an audio program component with regard to its content, such as music and effects, dialogue, Visually Impaired (VI) or Hearing Impaired (HI)

audio spectral front end: waveform coding tool for AC-4 in the Modified Discrete Cosine Transform (MDCT) domain that quantizes a spectrum based on a perceptual model and is tailored for general audio signals

average bit rate: mode of bit rate control that approximates a given bit rate when measured over longer periods of time

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

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

Bfc: bottom front centre channel

Bfl: bottom front left channel

Bfr: bottom front right channel

bit rate: rate or frequency at which bits appear in a bitstream, measured in bps

bitstream: sequence of bits

bitstream element: variable, array or matrix described by a series of one or more bits in an AC-4 bitstream

block: portion of a frame

block length: temporal extent of a block (for example measured in samples or QMF time slots)

C: centre channel

Cb: back-centre channel

channel-audio substream: substream that carries channel-based audio content

channel-based audio: audio content that is presented as one or more audio signals with a one-to-one mapping to nominal speaker positions

channel configuration: definition of the channel assignments of all channel-based audio signals within a program

channel element: bitstream element containing one or more jointly encoded channels

channel mode: coded representation of a channel configuration

codec: system consisting of an encoder and decoder

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.

codec frame length: temporal extent of a codec frame when decoded

commentary: audio program component containing supplementary dialogue, such as a director's commentary to a movie scene

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

companding: parametric coding tool for AC-4 in the Quadrature Mirror Filter (QMF) domain, used to control the temporal distribution of the quantization noise introduced in the Modified Discrete Cosine Transform (MDCT) domain

complete main: audio program component in which music and effects and dialogue are mixed together prior to encoding

constant bit rate: mode of bit rate control that yields the exact same number of bits per each frame or time period

core channel configuration: channel configuration present at the input to the downmix/upmix processing in the channel based renderer in core decoding mode

core decoding: one of two decoding modes for AC-4 decoder implementations which enables a low-complexity decode of a complete audio presentation for devices with limited output capabilities (e.g. mobile devices, tablets, televisions, etc.)

DASH accessibility descriptor: DASH descriptor for accessibility indication as specified by the accessibility scheme

DASH descriptor: DASH element used for property signalling

NOTE: DASH descriptors share a common structure.

DASH element: XML element contained in the DASH media presentation description

DASH essential property descriptor: DASH descriptor that specifies information about the containing DASH element that is considered essential by the media presentation author for processing the containing DASH element

DASH preselection element: DASH element used for audio preselection signalling

DASH representation: collection of one or more audio program components, encapsulated in a delivery file format and associated with descriptive metadata (such as language or role)

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-specific information: record in ISO base media file format files, used for decoder configuration

dialogue: audio program component containing speech

dialogue enhancement: coding tool for AC-4, used to enable the user to adjust the relative level of the dialogue to their preference

dialogue substream: substream that carries dialogue

dynamic adaptive streaming over HTTP: adaptive bit-rate streaming protocol that enables high-quality streaming of media content over the Internet delivered from HTTP

dynamic object: audio object whose spatial position is defined by three-dimensional coordinates that can change over time

dynamic range control: tool that limits the dynamic range of the output

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

elementary stream multiplexer: software component that combines several input elements into one or more output AC-4 elementary streams

NOTE: Such input elements could be AC-4 presentations, AC-4 substreams, complete AC-4 elementary streams, or specific metadata.

Extensible Metadata Delivery Format (EMDF): set of rules and data structures that enables robust signalling of metadata in an end-to-end process, involving a container, metadata payloads, and authentication protocols

NOTE: Specified in ETSI TS 102 366 [i.11] and ETSI TS 103 190-1 [1].

first-in-first-out: mode of buffer, or queue, processing where elements are output in the order they are inserted into the buffer

fps: number of unique consecutive audio or video frames an audio or imaging device produces in one second

frame rate: number of transmission frames decoded per second in realtime operation

framing: method that determines the QMF time slot group borders of signal or noise envelopes in A-SPX

full decoding: one of two decoding modes for AC-4 decoder implementations which enables decoding of a complete audio presentation for devices with expanded output capabilities (e.g. audio/video receiver)

gain: multiplicative factor applied to a signal

helper element: variable, array or matrix derived from a bitstream element

Huffman code: a class of variable-length prefix codes developed by David A. Huffman

Huffman code book: table of Huffman codes

immersive audio: extension to traditional surround sound with better spatial resolution and 3D audio rendering techniques used to enable a more realistic and natural sound experience

NOTE: Reproduction may use speakers located at ear level, overhead, and below the listener.

immersive channel audio: immersive audio that is channel-based

immersive channel configuration: channel configuration used for carriage of immersive audio content

immersive object audio: immersive audio that is object-based

independently decodable frame: AC-4 frame that contains at least one independently decodable SSF granule

input channel configuration: channel configuration present at the input to the downmix/upmix processing in the channel based renderer in full decoding mode

input channel mode: coded representation of the input channel configuration

input track: audio track present after the IMDCT, before A-JOC, S-CPL, A-CPL or A-JCC processing

intermediate decoding signal: output signal of the Stereo and Multichannel Processing tool when decoding the immersive_channel_element

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

intermediate spatial format object: static object whose spatial position is specified by ISF

ISO base media file format: media file format

NOTE: As specified in ISO/IEC 14496-12 [3].

L: left channel

Lb: left back channel

LFE: optional single channel of limited bandwidth (typically less than 120 Hz)

Lo/Ro: downmix from a multichannel to a two-channel output that is compatible for stereo or mono reproduction

Ls: left surround channel

Lt/Rt (Dolby Pro Logic II compatible output): Dolby Surround compatible downmix that contains content from all of the channels matrix encoded and mixed into the Left and Right channels

Lw: left wide channel

MIME type: identifier for file formats and contents

NOTE: As specified in IETF RFC 2045 [i.12].

music and effects: audio program component containing all audio except intelligible dialogue

NOTE: Can be combined with languages other than the primary language to create a dubbed version of the program.

next-generation audio: audio technology that enables broadcasters, operators, and content providers to create and deliver immersive and personalized audio content to be played back on consumer devices, adapting to different device capabilities

OAMD substream: substream that contains object audio metadata

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

object audio metadata: metadata used for rendering an audio object

object audio renderer: renders object-based audio to a specific speaker layout. The input is comprised of objects, and the outputs are speaker feeds

object audio substream: substream that carries object-based audio content

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

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

output channel configuration: channel configuration present at the output of the decoder

packet identifier: unique code that identifies a Packetized Elementary Stream (PES) within a transport stream

NOTE: The PID is contained in the transport stream packet header and is listed in the Service Information (SI) for a transport stream.

packetized elementary stream: elementary stream that is split into small chunks (packets) for transmitting and combining multiple streams within a transport stream

NOTE: Each PES is identified by a unique Packet Identifier (PID).

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

personalization: audio production techniques that enable sound designers and mixers to deliver different versions of the audio program that are tailored to viewers' needs and preferences and can be selected by the user on the UI

personalized audio: audio content provided by content creators and broadcasters that can be modified to the viewers' needs and preferences by offering a consumer choice between different versions of the audio program

premix: method for providing personalization options by mixing or merging of multiple audio program components into a single substream or substream group in accordance with associated metadata, prior to encoding to a Next-Generation Audio (NGA) elementary stream

presentation: set of references to AC-4 substreams to be presented to the listener simultaneously

NOTE: An audio preselection in AC-4.

presentation configuration: composition of an audio presentation, characterized by the types of referenced substreams

primary language: first language in a preference-ordered list of languages

program: self-contained audio-visual piece of content to be presented in TV or other electronic media, such as a television show or a sports game, consisting of an audio program, a video program, and other data

NOTE: A program may be interrupted by interstitial programs, such as advertisements.

Program Map Table: table within an MPEG-2 transport stream that defines the set of elementary streams associated with a specific program

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

QMF matrix: representation of the QMF domain as a matrix with QMF timeslot columns and QMF subband rows

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

QMF subband group: grouping of adjacent QMF subbands

QMF subsample: single element of a QMF matrix

QMF time slot: time range represented by one column in a QMF matrix

Quadrature Mirror Filter: filter that converts a PCM sample stream into two sample streams of half the input sampling rate, so that the output data rate equals the input data rate

R: right channel

Rb: right back channel

real-time loudness leveller: ITU-R based loudness correction system that enables real-time adjustment of program loudness, monitoring of program loudness, and creation of intelligent loudness metadata

Rs: right side/surround channel

Rw: right wide channel

set: collection of zero or more items

simple coupling: time-domain tool for processing immersive audio signals

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

spectral line: frequency coefficient

speech spectral front end: waveform coding tool for AC-4 in the Modified Discrete Cosine Transform (MDCT) domain that quantizes a spectrum based on a source model of speech and is tailored for speech signals

stream access point: data position in a DASH representation from which media playback can proceed without referring to any previously transmitted data other than the initialization segment

substream: decodable unit that represents a specific channel configuration (mono, stereo, or 5.1), and contains audio data and corresponding metadata

substream type: function of a substream in an audio presentation

NOTE: For substreams containing an audio program component, the substream type corresponds to the audio program component type.

table of contents: data record containing version, sequence number, frame rate and other data, and at least one presentation info element

Tbc: top back centre channel

Tbl: top back left channel

Tbr: top back right channel

Tc: top-centre channel

Tfc: top front centre channel

Tfl: top front left channel

Tfr: top front right channel

transmission frame size: data size of a transmission frame in the bitstream domain

transport system target decoder: hypothetical decoder

NOTE: As defined and used in ISO/IEC 13818-1 [i.13].

Tsl: top side left channel

Tsr: top side right channel

Uniform Resource Identifier: group of characters identifying a resource on a network (typically, the Internet)

universally unique identifier: 128-bit value chosen in such a way as to be universally unique

NOTE: The large name space for UUIDs renders the probability of duplicate IDs very close to zero.

window: weighting function associated with the inverse Modified Discrete Cosine Transform (IMDCT) of a block

3.2 Symbols

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

X*	complex conjugate of value *X*, if *X* is a scalar; and conjugate transpose if *X* is a vector
$\lceil \mathbf{x} \rceil$	round x towards plus infinity
$\lfloor \mathbf{x} \rfloor$	round x towards minus infinity
(A|B)	concatenation of matrix A with matrix B
$\mathbf{(A)^T}$	transpose operation of matrix A

3.3 Abbreviations

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

3GPP	3rd Generation Partnership Project
ABR	Average Bit Rate
A-CPL	Advanced Coupling
A-JCC	Advanced Joint Channel Coding
A-JOC	Advanced Joint Object Coding
ASF	Audio Spectral Front end
A-SPX	Advanced Spectral extension
Bfc	Bottom Front Centre
Bfl	Bottom front left
Bfr	Bottom front right
C	Centre
CAS	Channel-Audio Substream
Cb	Back centre
CBR	Constant Bit Rate
CMAF	Common Media Application Format
CRC	Cyclic Redundancy Check
DASH	Dynamic Adaptive Streaming over HTTP
DE	Dialogue Enhancement
DRC	Dynamic Range Control
DSI	Decoder-Specific Information
EMDF	Extensible Metadata Delivery Format
ESM	Elementary Stream Multiplexer
FIFO	First-In-First-Out
fps	frames per second
HCB	Huffman Code Book
IMDCT	inverse Modified Discrete Cosine Transform
ISF	Intermediate Spatial Format
ISOBMFF	ISO Base Media File Format
L	Left
Lb	Left back
LFE	Low-Frequency Effects
Lo/Ro	Left only/Right only

Ls	Left side/surround
Lscr	Left screen
Lw	Left wide
MIME	Multipurpose Internet Mail Extensions
OAMD	Object Audio MetaData
OAR	Object Audio Renderer
OAS	Object Audio Substream
PCM	Pulse Code Modulation
PES	Packetized Elementary Stream
PID	Packet Identifier
PMT	Program Map Table
QMF	Quadrature Mirror Filter
R	Right
Rb	Right back
Rs	Right side/surround
Rscr	Right screen
RTLL	Real-Time Loudness Leveller
Rw	Right wide
S-CPL	Simple Coupling
SMP	Stereo and Multichannel Processing
SSF	Speech Spectral Front end
Tbc	Top back centre
Tbl	Top back left
Tbr	Top back right
Tc	Top centre
Tfc	Top front centre
Tfl	Top front left
Tfr	Top front right
TOC	Table Of Contents
Tsl	Top side left
Tsr	Top side right
T-STD	Transport System Target Decoder
UI	User Interface
uimsbf	unsigned integer, most-significant bit first
UTF	Unicode Transformation Format
UUID	Universally Unique IDentifier

3.4 Conventions

Unless otherwise stated, the following convention regarding the notation is used:

- Constants are indicated by uppercase italic, e.g. *NOISE_FLOOR_OFFSET*.
- Tables are indicated as *TABLE[idx]*.
- Functions are indicated as *func(x)*.
- Variables are indicated by italic, e.g. *variable*.
- Vectors and vector components are indicated by bold lowercase names, e.g. **vector** or $\mathbf{vector}_{\mathbf{idx}}$.
- Matrices (and vectors of vectors) are indicated by bold uppercase single letter names, e.g. **M** or $\mathbf{M}_{\mathbf{row,column}}$.
- Indices to tables, vectors and matrices are zero based. The top left element of matrix **M** is $\mathbf{M}_{\mathbf{0,0}}$.
- Bitstream syntactic elements are indicated by the use of a different font, e.g. `presentation_id`. `ac4_presentation_v1_dsi/presentation_id` indicates the `presentation_id` element part of `ac4_presentation_v1_dsi()`. All bitstream elements are described in clause 6.3.
- Normal pseudocode interpretation of flowcharts is assumed, with no rounding or truncation unless explicitly stated.

- Units of [dB_2] refer to the approximation 1dB ≅ factor of $\sqrt[6]{2,0}$, i.e. 6dB ≅ factor of 2,0.
- Fractional frame rates are written in "shorthand notation" as defined in table 1.
- Hexadecimal constants are denoted `0x…` .
- Binary constants are denoted `0b…` .
- Where several alternatives exist for a digit, `x` is used as placeholder.
- Table 2 specifies standard functions used throughout pseudocode sections. Functions with a single argument also apply to vectors and matrices by mapping the function element wise.
- Speaker and channel configurations are either specified as f/s/h.x or hp.x.h. The notation can be abbreviated to f/s/h or hp.x respectively, if the number of the corresponding speakers is zero. Table 3 defines symbols f, s, h, x, and hp.
- If a Boolean bitstream element is defined to signal a *whether statement* then the *statement* is true if the Boolean is true; i.e. the corresponding bit is set to 1.
- Arrays in the bitstream are sent element by element, where the element with the *lowest* index is sent *first*.

Table 1: Shorthand notation for frame rates

Fractional framerate	Shorthand
$24 \times {}^{1\,000}/_{1\,001}$	23,976
$30 \times {}^{1\,000}/_{1\,001}$	29,97
$48 \times {}^{1\,000}/_{1\,001}$	47,952
$60 \times {}^{1\,000}/_{1\,001}$	59,94
$120 \times {}^{1\,000}/_{1\,001}$	119,88

Table 2: Standard functions used in pseudocode

Function	Meaning
`abs(`*arg*`)`	|*arg*|
`sqrt(`*arg*`)`	$arg^{0,5}$
`pow(`*arg1*`,` *arg2*`)`	$arg1^{arg2}$
r `=` *a* `?` *b* `:` *c*	`if (`*a*`)` *r* `=` *b*`; else` *r* `=` *c*`;`
`clip3(`*min*`,` *max*`,` *val*`)`	`(`*val* `<` *min*`) ?` *min* `: (`*val* `>` *max*`) ?` *max* `:` *val*

Table 3: Symbols for speaker/channel configurations

Symbol	Speakers
f	front
s	surround
h	height
x	low-frequency effects
hp	horizontal plane

4 Decoding the AC-4 bitstream

4.1 Introduction

The following clauses provide introduction, description, and specification for several topics.

- Clause 4.2 introduces object-based audio coding and describes differences to traditional channel-based audio coding.
- Clause 4.3 introduces the immersive audio experience.
- Clause 4.4 introduces the functionality of personalized audio in the context of AC-4.
- Clause 4.5 describes the logical and physical structure of an AC-4 bitstream.
- Clause 4.6 specifies compatibilities for the decoder to bitstream (elements) specified in ETSI TS 103 190-1 [1].
- Clause 4.7 specifies the decoding modes an AC-4 decoder supports.
- Clause 4.8 specifies how to build an AC-4 decoder by using the decoding tools specified in clause 5.

4.2 Channels and objects

Objects give content creators more control over how content is rendered to loudspeakers in consumer homes.

In channel-based audio, 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.

The following channel designations and associated abbreviations are used in this specification:

- Left (L)
- Right (R)
- Centre (C)
- Low-Frequency Effects (LFE)
- Left Side/Surround (Ls)
- Right Side/Surround (Rs)
- Left Back (Lb)
- Right Back (Rb)
- Top Front Left (Tfl)
- Top Front Right (Tfr)
- Top Side Left (Tsl)
- Top Side Right (Tsr)
- Top Back Left (Tbl)
- Top Back Right (Tbr)
- Top Front Centre (Tfc)
- Top centre (Tc)

- Top back centre (Tbc)
- Bottom Front Left (Bfl)
- Bottom Front Right (Bfr)
- Bottom Front Centre (Bfc)
- Back Centre (Cb)
- Left Wide (Lw)
- Right Wide (Rw)
- Left Screen (Lscr)
- Right Screen (Rscr)

In object-based audio, rendering is applied to audio objects (the object essence in conjunction with individually assigned properties). The properties more explicitly specify how the content creator intends the audio content to be rendered, i.e. they place constraints on how to render essence into speakers.

Constraints include:

- **Location and extent:**
    - Controlling how much of the object energy gets rendered on individual speakers.
- **Importance:**
    - Controlling which objects get prioritized if rendering to few speakers overloads the experience.
- **Spatial exclusions:**
    - Controlling which regions in the output an object should not be rendered into.
- **Divergence:**
    - Controlling width and presence of (predominantly dialogue) objects.

AC-4 specifies the following object types:

- **Dynamic object:**
    - An object whose spatial position is defined by 3-dimensional coordinates, which may change dynamically over time.
- **Bed object:**
    - An object whose spatial position is defined by an assignment to a speaker of the output channel configuration.
- **Intermediate spatial format object:**
    - An object whose spatial position is defined by an assignment to a speaker in a stacked ring representation.

The tool for rendering dynamic objects or bed objects to speakers is called the object audio renderer. Rendering Intermediate Spatial Format (ISF) objects to speakers is referred to as intermediate spatial format rendering. The present document does not mandate any specific rendering algorithm.

4.3 Immersive audio

Immersive audio refers to the extension of traditional surround sound reproduction to include higher (spatial) resolution and full 3D audio rendering techniques (including ear-level, overhead, and floor speakers located below the listener) in order to reproduce more realistic and natural auditory cues. It also is often associated with audio creation and playback systems that leverage object-based and/or hybrid-channel/object-audio representations.

The present document specifies transmission capabilities for channel-based, intermediate-spatial, and object-based formats. Each format comes with pros and cons; it is up to content creators to pick the right trade-off for them:

- **Channel-based audio:**
  - In traditional channel-based audio mixing, sound elements are mixed and mapped to individual, fixed speaker channels, e.g. Left, Right, Centre, Left Side/Surround, and Right Side/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. The present document extends channel-based coding to higher number of speakers, including overhead speakers.
  - However, with the popularity of new speaker setups, no assumption can be made about the speaker setup used for playback, and channel-based audio does not offer good means of adapting a presentation where the source speaker layout does not match the speaker layout at the decoding end. This presents a challenge when trying to author content that plays back well no matter what speaker configuration is present.
- **Intermediate spatial format audio:**
  - The intermediate spatial format addresses this problem by providing audio in a form that can be rendered more easily to different speaker layouts.
- **Object-based audio:**
  - In object-based audio, individual sound elements are delivered to the playback device where they are rendered based on the speaker layout in use. Because individual sound elements can be associated with a much richer set of metadata, giving meaning to the elements, the adaptation to the reproducing speaker configuration can make much better choices of how to render to fewer speakers.

NOTE: When no single scheme fits well, combinations are possible; diffuse, textural sounds such as scene ambience, crowd noise, and music can be delivered using a traditional channel-based mix referred to as an audio bed and combined with object-based audio elements.

4.4 Personalized Audio

Personalization allows the content creator to deliver multiple stories for their content, providing audio experiences that are tailored to the consumer's preference and to the capabilities of the playback device. The present document specifies syntax and tools to support personalized audio experiences:

- **Presentations:**
  - The ability to deliver multiple audio program components and combine these into presentations is the key building block for delivering personalized experiences. Presentations allow flexibility in creating a wider range of audio experiences that are relevant to their content. Simple personalization can be achieved by creating a selection of presentations relevant to the content.

EXAMPLE 1: For sports events, a "team-biased" presentation containing a "home" and an "away" team can be created, having different commentators and supporter crowd effects. AC-4 carries descriptive text to allow a decoding device to present the consumer with a way of selecting the presentation that aligns with the consumer preference.

- **Target settings for playback devices:**
  - Some choices are not driven by consumer preference but rather by playback device capabilities. AC-4 provides the means to define multiple sets of target device metadata. Target-device metadata enables the content creator to define how the same audio components are combined on different devices.

EXAMPLE 2: An audio engineer may define the balance between dialogue and substream elements differently for a 5.1 playback system and a mobile device. On the mobile device, the audio engineer may decide that louder dialogue and a lower substream level are appropriate. Target-device rendering works in conjunction with dynamic range control to enable the audio engineer to create audio targeted towards each category of consumer devices.

The inclusion of target-device metadata is optional but, where used, this rendering may support the downmix process and give the content creator greater artistic flexibility as to how their mix plays back on different devices.

- **Extended User Interface (UI) controls:**
  - For devices that support a UI or a second screen device, the content creator can define which controls are to be presented to the consumer to provide greater flexibility in creating a personalized experience.

- **Classification of objects:**
  - AC-4 metadata fields allow objects to be classified as a specific type (for example dialogue). This expands on the functionality offered by dialogue enhancement by enabling a wider range of control to adjust the balance between dialogue and other audio components of the selected presentation.

NOTE: While the present document describes the metadata and controls available for creating personalization, it does not specify requirements. Specifying decoder behaviour is left to application specifications.

4.5 AC-4 bitstream

4.5.1 Bitstream structure

This clause describes the top level structure of the bitstream, from Table Of Contents (TOC) level down.

An AC-4 bitstream is composed of a series of raw AC-4 frames, each of which can be encapsulated in the AC-4 sync frame transport format (see annex C), or in another transport format. Figure 1 shows how the key structures are composed:

- **TOC:**
  - The TOC lists the presentations that are carried in the stream.
  - The TOC contains a list of one or more presentations.
- **Presentation:**
  - Presentations are described by the `ac4_presentation_v1_info()` structures if `bitstream_version` ≥ 1, and by the `ac4_presentation_info()` structures if `bitstream version` = 0. Presentations with `bitstream_version` = 0 are described in ETSI TS 103 190-1 [1].
  - A presentation informs the decoder which parts of an AC-4 stream are intended to be played back simultaneously at a given point in time. As such, presentations describe available user experiences. Features such as loudness or dialogue enhancement are therefore managed on presentation level.
  - Presentations consist of substream groups.
- **Substream group:**
  - Substream groups are referenced through the `ac4_substream_group_info()` element in the TOC.
  - Each substream group has a specific role in the user experience: music and effects, dialogue, or associated audio. The substream group carries properties common to all substreams contained in the substream group. Depending on the role, specific metadata is associated with the substream group (e.g. maximum amount of dialogue boost).
  - Substream groups can be shared between presentations, so that parts common to several experiences do not need to be transmitted twice. Relative gains applicable to substream groups can be transmitted per presentation (i.e. a substream group can be rendered with different gains in different presentations). Further, substream groups can indicate that their referenced substreams are not contained in the AC-4 stream, but rather in a separate elementary stream. This provides support for dual-PID and second-screen scenarios.
  - Substream groups consist of one or more individual substreams. Substreams in one substream groups are either all channel coded or all object coded.
- **Substream:**
  - Substreams are referenced through the `ac4_substream_info_chan()`, `ac4_substream_info_ajoc()`, and `ac4_substream_info_obj()` elements in the TOC.
  - Substreams contain the coded audio signal. Coded audio can either represent channel-based audio or object-based audio. One substream can be part of several different substream groups, for efficiency reasons, and thus be part of different presentations.
  - Substreams can be shared between substream groups, and therefore between different presentations.

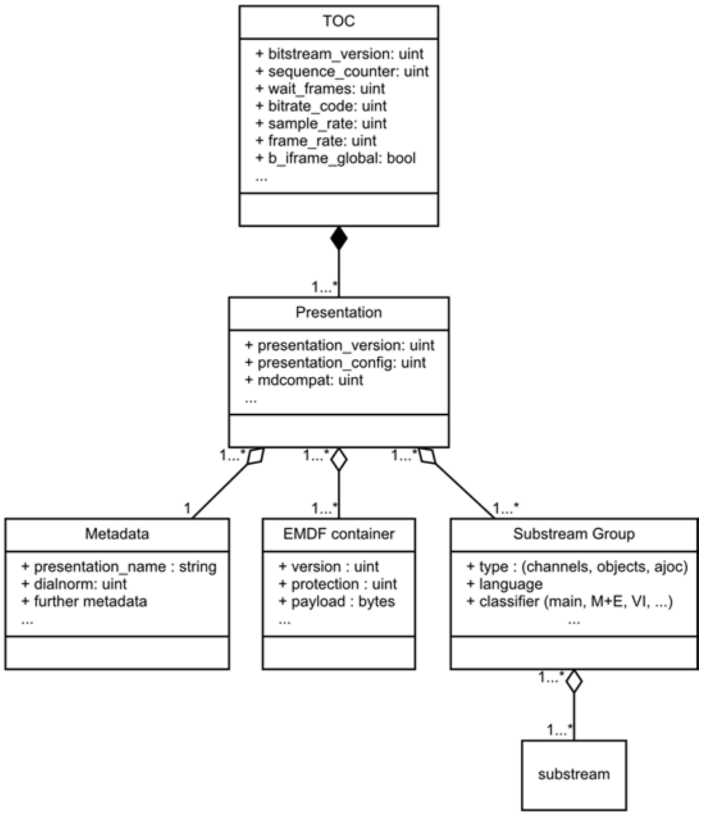


Figure 1: High-level bitstream structure

EXAMPLE: Figure 2 shows a TOC with several presentations for music and effects together with different language tracks. The selected presentation contains the 5.1 music and effects substream, as well as English dialogue. In the example, the defined substream groups just contain single substreams. Other presentations could include different languages or additional commentary.

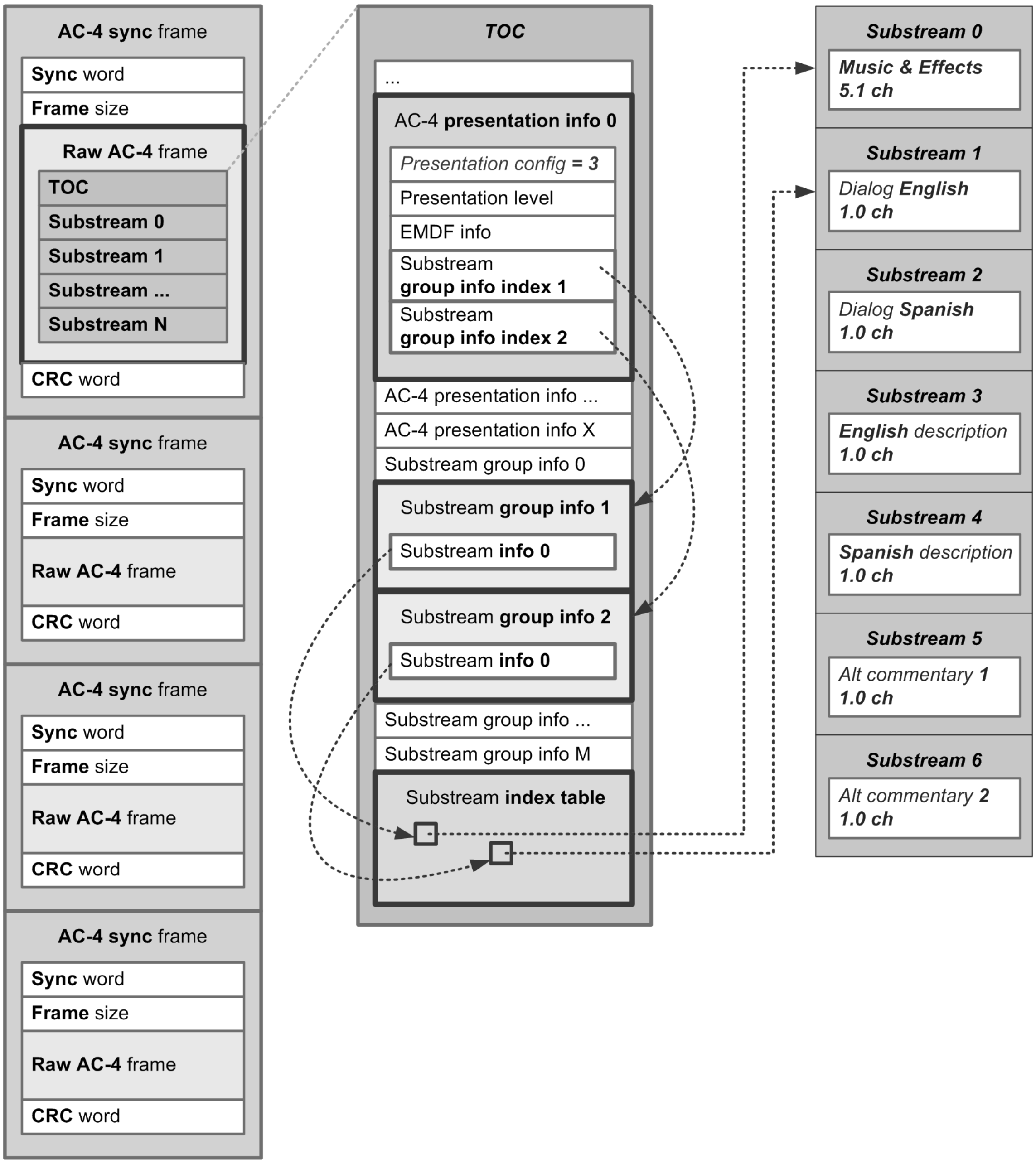


Figure 2: Example of complex frame with several presentations and substream groups

4.5.2 Data dependencies

The AC-4 bitstream syntax supports data to be encoded incrementally over multiple consecutive codec frames (dependency over time). The bitstream syntax conforming to the present document supports individual dependency over time for substreams. If the transmitted data of one codec frame has no dependency over time for any of the substreams, the corresponding codec frame is an I-frame, which is also indicated by the helper variable *b_iframe_global*.

NOTE 1: In a bitstream conforming to ETSI TS 103 190-1 [1], an I-frame is present if `b_iframe` is true.

The following substream specific flags indicate whether the current data of the corresponding substream has no dependency over time and hence the data can be decoded independently from preceding frames:

b_audio_ndot Valid for an `ac4_substream`

b_pres_ndot Valid for the `ac4_presentation_substream`

b_oamd_ndot Valid for the `oamd_substream`

NOTE 2: The `oamd_dyndata_single` and `oamd_dyndata_multi` elements present in one codec frame can differ in the dependency over time, because both elements can be contained in different substream types.

4.5.3 Frame rates

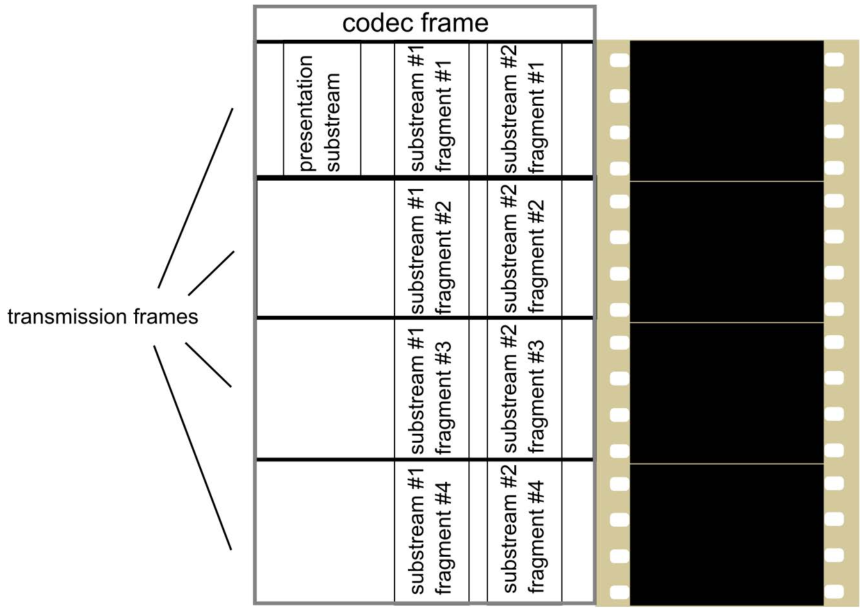
The transmission frame rate of AC-4 can be controlled to match it to another frame rate, such as video.

In a number of scenarios, it is desirable to match the duration of transmitted audio frames to the duration of video frames. Amongst other things, this assists with achieving clean splices and seamless switching between Dynamic Adaptive Streaming over HTTP (DASH) representations.

There are two important variables here: `frame_rate_index` and `frame_rate_fraction`.

`frame_rate_index` refers to the rate of transmitted audio frames. It is also possible to control the efficiency of transmitting AC-4 at high frame rates by distributing a codec frame over several transmission frames, as "fractions" of frames, using the efficient high frame rate mode. To achieve this, a `frame_rate_fraction` can be specified, so that the codec frame rate can be lower than the transmission frame rate by a factor of 2 or 4.

Figure 3 shows the relationship of video frames, audio codec frames, and audio transmission frames.



NOTE: frame_rate_fraction = 4.

Figure 3: Efficiently transmitting AC-4 with a frame rate matched to video

See also clause 5.1.3 and clause 6.3.2.4.

4.6 Decoder compatibilities

Decoders conforming to the present document shall be capable of decoding bitstreams where 0 ≤ `bitstream_version` ≤ 2. Moreover, decoders conforming to the present document shall be capable of decoding all presentations that conform to the decoder compatibility level as per clause 6.3.2.2.3 while also conforming to either ETSI TS 103 190-1 [1] (`presentation_version` = 0) or to the present document (`presentation_version` = 1).

Table 4 shows the `presentation_config` values that the decoder shall be able to decode, dependent on the bitstream version and presentation version. Table 5 shows the substream version values the decoder shall be able to decode, dependent on the bitstream version and presentation version.

If the bitstream version is 1 and the presentation version is 1, the decoder shall decode the `ac4_presentation_info()` element and evaluate the contained `presentation_config` value. If that `presentation_config` is 7, the decoder shall decode the `ac4_presentation_v1_info()` element contained in `presentation_config_ext_info()`.

NOTE: If `bitstream_version` = 1, for compatibility with ETSI TS 103 190-1 [1], the `ac4_presentation_v1_info()` can be contained in `ac4_presentation_info()`.

Table 4: Valid combinations of bitstream version, presentation version, and presentation_config

Bitstream version	Presentation version	`presentation_config` contained in `ac4_presentation_info()`	`presentation_config` contained in `ac4_presentation_v1_info()`
0	0	{0 … 6} or `presentation_config` not present	N/A
1	0	{0 … 6} or `presentation_config` not present	N/A
1	1	7	{0 .. 6} or `presentation_config` not present
2	{1 ; 2}	N/A	{0 .. 6} or `presentation_config` not present
NOTE: The `presentation_config` can be contained in `ac4_presentation_info()` and in `ac4_presentation_v1_info()`, but does not need to be present in the corresponding bitstream element. The `presentation_config` elements marked with N/A are not available because the bitstream element in which they are contained is not present in the bitstream.			

Table 5: Valid combinations of bitstream version, presentation version, and substream version

Bitstream version	Presentation version	Substream version
0	0	0
1	0	0
1	1	{0; 1}
2	{1; 2}	1

4.7 Decoding modes

4.7.1 Introduction

The present document specifies two decoding modes: full decoding and core decoding. The decoder implementation shall support at least one of these two modes.

4.7.2 Full decoding mode

The full decoding mode supports fully immersive audio experience and the highest resolution of spatial information. In this decoding mode, the decoding tools, advanced coupling and advanced joint channel coding, decode all coded channels and enable the immersive channel configurations to be played back with the maximum number of channels present. The advanced joint object coding tool decodes the maximum number of audio objects present individually, resulting in the highest spatial fidelity.

4.7.3 Core decoding mode

The core decoding mode enables the immersive experience with a reduced number of channels and the object-audio experience with reduced spatial information to support decoding on low-complexity platforms. In this decoding mode, decoding tools such as advanced coupling, advanced joint channel coding or advanced joint object coding operate in the core decoding mode or are turned off. The decoding of a subset of the channels present in an immersive channel configuration supports the immersive audio experience for decoders on low-complexity platforms. For object audio, the core decoding mode supports decoding of a reduced number of audio objects individually. This does not mean that any of the objects present are discarded, but the spatial resolution of the object-audio experience can be reduced.

4.8 Decoding process

4.8.1 Overview

This clause describes how the decoder, specified in the present document, shall decode the AC-4 bitstream by utilizing the AC-4 decoding tools specified in clause 5. This process implies the usage of the AC-4 bitstream elements that are specified and described in clause 6. The general process that shall be performed to decode an AC-4 bitstream is described by the consecutive steps:

1) Select presentation.

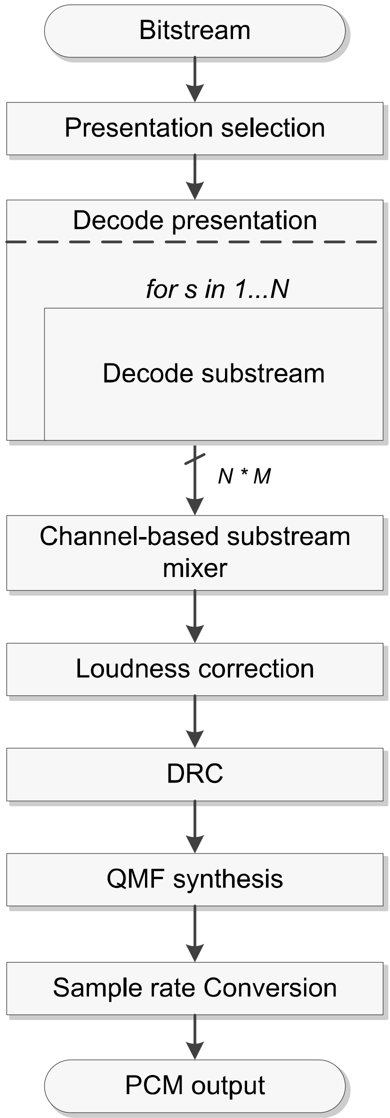
2) Decode presentation information.

3) Decode all substreams of the selected presentation.

4) Mix substreams.

5) Perform dynamic range control and loudness processing.

Figure 4 shows these steps as a flowchart.



NOTE: N = Number of substreams in presentation; M = number of objects in OAS (M = 1 for CAS).

Figure 4: Flow diagram of the decoding process (AC-4)

4.8.2 Selecting a presentation

The selection of presentations is application dependent. To successfully select an appropriate presentation and the related substreams, the decoder may execute the following steps:

1) Create a list of presentations available in the bitstream:

   - Initially derive the number of presentations, *n_presentations*, from the bitstream elements `b_single_presentation` and `b_more_presentations` in `ac4_toc()`, as indicated in clause 6.2.1.1.

   - For each of these *n_presentations*, parse the presentation information. The presentation information is carried in the `ac4_presentation_info()` element if `bitstream_version` ≤ 1 and as `ac4_presentation_v1_info()` otherwise.

2) Select the presentation that is appropriate for the decoder and the output scenario:

- Details available to support presentation selection are mainly language type, availability of associated audio, and type of audio (multichannel for speaker rendering, or a pre-virtualized rendition for headphones). For an alternative presentation (`b_alternative` is true), the `presentation_name` element provides a label that may be displayed in an appropriate UI.

NOTE: A decoding system usually employs a user agent to make the choice without the need to directly interact with the end user.

- Only presentations with an `md_compat` value that matches the decoder compatibility level shall be selected as indicated in clause 6.3.2.2.3.
- A presentation that is signalled as disabled (`b_enable_presentation` is false) shall not be selected.
- The decoder should not rely on the order and number of presentations, as both can change over time.

3) Select the substreams to decode:

- Depending on the origin of the presentation information for the selected presentation, the audio substreams to decode are determined differently:
  - If the presentation information originates from an `ac4_presentation_info()` element, the substreams associated with the presentation are given by `substream_index` fields within the substream info elements which are part of the `ac4_presentation_info()` element.
  - If the presentation information originates from an `ac4_presentation_v1_info()` element, each substream group with `b_substreams_present` is true, referenced by `ac4_sgi_specifier()` elements, references substreams by means of `substream_index` fields within substream info elements. All substreams from all substream groups within the presentation shall be selected for subsequent decoding. If `b_multi_pid` = 1, additional substream groups from other elementary streams shall be selected as described in clause 5.1.2.
- The `substream_index` values are used as an index into the `substream_index_table()` and the substream offset is calculated as described in ETSI TS 103 190-1 [1], clause 4.3.3.12.4.
- Alternative presentations (`b_alternative` is true) allow the application of alternative metadata to the selected substreams. Each substream used in the alternative presentation keeps OAMD in an `oamd_dyndata_single()` field. The `alt_data_set_index` field is used for the identification of the alternative OAMD as described in clause 6.3.3.1.15.

4.8.3 Decoding of substreams

4.8.3.1 Introduction

In a presentation where `presentation_version` = 1, two types of substreams containing audio content can be present:

- Channel-Audio Substream (CAS) (`b_channel_coded` is true)
- Object Audio Substream (OAS) (`b_channel_coded` is false)

Presentations with `presentation_version` = 0 support CAS coding only.

The decoding of each substream is specified by the following steps:

1) Decode the object properties for all objects present in object audio substreams.
2) Apply audio spectral front end or speech spectral front end processing.
3) Apply stereo and multichannel processing.
4) Depending on the content, apply one of the following decoding tools:

- Simple Coupling (S-CPL):

        - Applicable to core decoding mode and full decoding mode.
    - Advanced Coupling (A-CPL):
        - Applicable to full decoding mode (for core decoding mode, gain factors shall be applied instead).
    - Advanced Joint Channel Coding (A-JCC):
        - Applicable to core decoding mode and full decoding mode.
    - Advanced Joint Object Coding (A-JOC):
        - Applicable to full decoding mode.
5) Apply dialogue enhancement.
6) For CAS only:
    - Apply dynamic range control gains present in the bitstream.
7) Render to the output channel configuration.

Because these steps utilize AC-4 decoding tools specified for different processing domains, additional steps for time-to-frequency-domain transformation and vice versa, which are not listed in the general steps, shall be performed. A more detailed specification of the decoding process is shown in figure 5. The following clauses specify individual steps in more detail.

The output of the decoding process of one substream depends on the type of the substream as follows:

- CAS:
    - *N* audio sample blocks associated to channels, where *N* is the number of channels in the output channel configuration.
- OAS:
    - *M* times *N* audio sample blocks associated to channels, where *N* is the number of channels in the output channel configuration and *M* is the number of audio objects present in the corresponding substream. Each object is processed individually by the object audio renderer, which produces *N* audio sample blocks associated to channels.

4.8.3.2 Identification of substream types

For the purpose of decoding, substreams can be categorized into the following substream types:

- main;
- music and effects;
- associated audio; or
- dialogue.

NOTE 1: The main substream and the music and effects substream require the same processing; they are considered as one single substream type in the present document.

NOTE 2: For `presentation_version` = 1, substream types can be identified as specified in clause 6.3.2.2.1 (for single substream group presentations) and clause 6.3.2.2.2, table 53 depending on the `presentation_config`. For `presentation_version` = 0, substream types can be identified as specified in ETSI TS 103 190-1 [1], clause 4.3.3.3.4.

4.8.3.3 Substream decoding overview

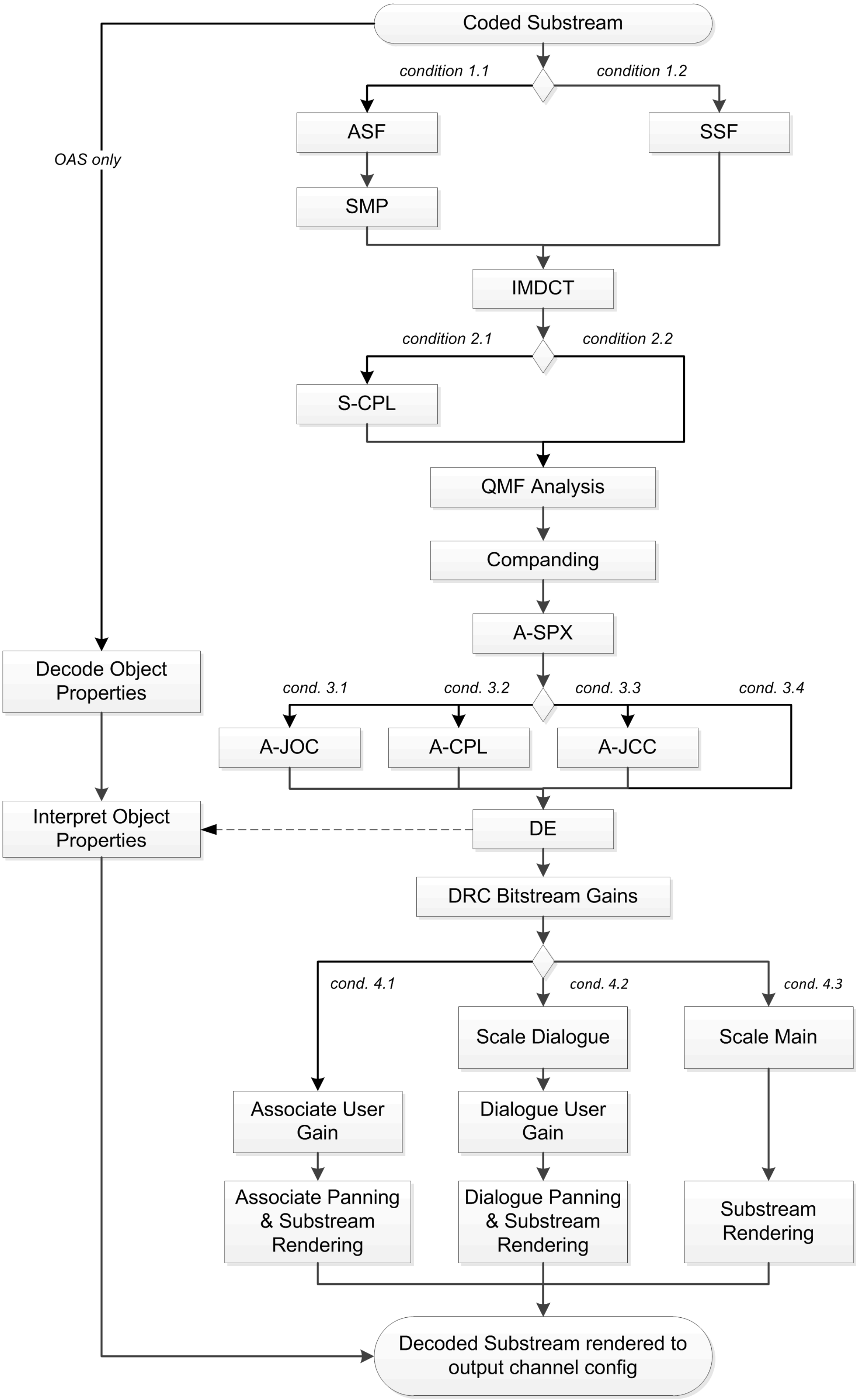


Figure 5: Substream decoding

The conditions shown in figure 5 are specified in table 6.

Table 6: Conditions for substream decoding

No.	Condition	Description
1.1	if (spec_frontend == 0)	Tracks with an associated spectral frontend type of ASF (spec_frontend = 0) shall be processed by the ASF tool.
1.2	if (spec_frontend == 1)	Tracks associated with the type speech spectral front end (SSF) (spec_frontend=1) shall be processed by the SSF tool.
2.1	if ((channel element == immersive_channel_element) and (immersive_codec_mode in [SCPL, ASPX_SCPL])) /* see note 1 */	If the decoder processes an `immersive_channel_element` where the `immersive_codec_mode` is either SCPL or ASPX_SCPL, the decoder shall utilize the S-CPL tool.
2.2	All other cases not covered by the condition 2.1	For all channel elements plus codec mode processing cases not covered by the condition 2.1, or for object audio, the S-CPL tool shall be bypassed.
3.1	if ((decoding mode == full decoding) and (b_channel_coded == 0) and (b_ajoc == 1))	In full decoding mode, the decoder shall use the A-JOC tool for A-JOC coded content.
3.2	if (codec mode in [ASPX_ACPL_1, ASPX_ACPL_2, ASPX_ACPL_3]) /* see note 2 */	For full decoding of one channel element where the corresponding codec mode is one of {`ASPX_ACPL_1`, `ASPX_ACPL_2`, `ASPX_ACPL_3`}, the decoder shall utilize the A-CPL tool.
3.3	if ((channel element == immersive_channel_element) and (immersive_codec_mode == ASPX_AJCC))	For decoding the `immersive_channel_element` where `immersive_codec_mode` is ASPX_AJCC, the decoder shall utilize the A-JCC tool.
3.4	All other cases	All other cases not covered by any of the conditions defined as 3.1, 3.2, or 3.3 in this table.
4.1	if the substream type is associated audio	Refer to clause 4.8.3.2.
4.2	if the substream type is dialogue	Refer to clause 4.8.3.2.
4.3	if the substream type is either main or music and effects	Refer to clause 4.8.3.2.
NOTE 1: `channel element` refers to the channel element to be processed by the decoder, and it can take one of the following values: [`single_channel_element`, `channel_pair_element`, `3_0_channel_element`, `5_X_channel_element`, `7_X_channel_element`, `immersive_channel_element`, `22_2_channel_element`]. NOTE 2: `codec mode` corresponding to the channel element, which can be one of {`mono_codec_mode`, `stereo_codec_mode`, `3_0_codec_mode`, `5_X_codec_mode`, `7_X_codec_mode`, `immersive_codec_mode`, `22_2_codec_mode`}.		

4.8.3.4 Decoding of object properties

4.8.3.4.1 Introduction

Decoding of object properties present in the Object Audio Metadata (OAMD) portions of the bitstream shall comprise the following steps:

1) Retrieve and decode OAMD from the different locations in the bitstream:

   - Determine the number and type of objects present in the currently decoded substream group by decoding the OAMD configuration data.

   - Decode the OAMD common data and OAMD timing data applicable to all objects of the substream group.

   - For each object decode the OAMD dynamic data. For an alternative presentation, use the alternative metadata as indicated in the decoded presentation.

2) Process the OAMD timing data to synchronize the object properties with the pulse code modulation (PCM) audio samples of the object essence(s) as specified in clause 5.9.

4.8.3.4.2 Object audio metadata location

The location of object audio metadata in the substreams depends on whether object properties, coded within the OAMD portion, apply to all objects, a group of objects, or an individual object.

Object properties are coded in multiple OAMD portions inside the bitstream, which are present on different bitstream layers. The principle factor that determines the location of specific OAMD information in the bitstream is whether the object audio substream is coded with A-JOC or not (as indicated by `b_ajoc`). The location of OAMD dynamic data for substreams that are not coded with A-JOC also depends on whether alternative presentations are present (as indicated by `b_alternative`). Table 7 shows the location of the OAMD portions for possible values of `b_ajoc`.

Table 7: OAMD locations in the bitstream

OAMD portion	b_ajoc is false	b_ajoc is true
OAMD configuration data	`ac4_substream_info_obj` (clause 6.2.1.11)	`ac4_substream_info_ajoc` (clause 6.2.1.9)
OAMD common data	`oamd_common_data` (clause 6.3.9.2), contained in `oamd_substream` (clause 6.3.3.2)	`oamd_common_data` (clause 6.3.9.2), contained in `oamd_substream` (clause 6.3.3.2) or `ac4_substream_info_ajoc` (clause 6.2.1.9)
OAMD timing data	`oamd_timing_data` (clause 6.3.9.3), contained in `oamd_substream` (clause 6.3.3.2)	`oamd_timing_data` (clause 6.3.9.3), contained in `oamd_substream` (clause 6.3.3.2) or `audio_data_ajoc` (clause 6.2.3.4)
OAMD dynamic data	The location depends on the value of `b_alternative`: • `b_alternative` is false. `OAMD_dyndata_multi` (clause 6.3.9.5), contained in `oamd_substream` (clause 6.3.3.2) • `b_alternative` is true. `OAMD_dyndata_single` (clause 6.3.9.4), contained in `metadata` (clause 6.2.7.1), which is contained in `ac4_substream` (clause 6.2.2.2) and `OAMD_dyndata_multi` (clause 6.3.9.5), contained in `oamd_substream` (clause 6.3.3.2)	`OAMD_dyndata_single` (clause 6.3.9.4), contained in `audio_data_ajoc` (clause 6.2.3.4)

If `b_alternative` is true, alternative object properties can be present in the `oamd_dyndata_single` element.

If the object audio substream is coded with A-JOC and `b_static_dmx` is false, two individual OAMD portions exist inside the corresponding substream. Each portion comprises one `oamd_dyndata_single` element and up to two optional `oamd_timing_data` elements. Presence of `oamd_timing_data` elements in the bitstream is indicated by flags `b_dmx_timing` or `b_umx_timing`. The decoder shall use the first portion present in the bitstream for core decoding mode and the second portion present in the bitstream for full decoding mode.

4.8.3.5 Spectral frontends

The spectral frontend is the first tool that shall be utilized to decode a substream. The spectral frontend decodes the data present in `sf_data` and provides a block of spectral lines for an audio track and associated information about the subsequent windowing process for the frequency-to-time transformation.

The `sf_data` elements are present in the channel elements specified in ETSI TS 103 190-1 [1], clause 4.2.6 and clause 6.2.4. These channel elements are used to encode both channel-audio substreams and object-audio substreams. Channel-audio substreams contain one or more channel elements to represent the input channel configuration. Object-audio substreams contain either a single object coded in `mono_data`, or multiple objects coded in channel element(s).

The AC-4 decoder specification provides two spectral frontend tools:

- **Audio spectral front end:**
  - The Audio Spectral Front end (ASF) is specified in ETSI TS 103 190-1 [1], clause 5.1. If `spec_frontend` is 0, the ASF shall be utilized.

Speech spectral front end:

  - The SSF is specified in ETSI TS 103 190-1 [1], clause 5.2. If `spec_frontend` is 1, the SSF shall be utilized.

The decoder shall utilize the spectral frontend to decode all `sf_data` elements while taking into taking account the associated `sf_info` and `sf_info_lfe` elements as specified in ETSI TS 103 190-1 [1], clauses 6.2.6.3 and 6.2.6.4, respectively.

4.8.3.6 Stereo and Multichannel Processing (SMP)

The Stereo and Multichannel Processing (SMP) tool shall be utilized to process the output tracks of ASF. If the input channel configuration is one of the immersive channel configurations, the decoder shall utilize the SMP tool for immersive input as specified in clause 5.2. For all other input channel configurations or the processing of an object audio substream, the decoder shall utilize the SMP tool specified in ETSI TS 103 190-1 [1], clause 5.3.

If the input channel configuration is 7.X.4, 9.X.4, 5.X, or 7.X, the audio track acquired from the first `sf_data` element shall be mapped to the LFE channel, and shall be directly routed to the Inverse Modified Discrete Cosine Transform (IMDCT) stage, i.e. it is not passed into the SMP tool. If the input channel configuration is 22.2, the audio tracks acquired from the first two `sf_data` elements shall be mapped to the LFE channels, and shall be directly routed to the IMDCT stage, i.e. they are not passed into the SMP tool.

4.8.3.7 Inverse Modified Discrete Cosine Transformation (IMDCT)

After SSF processing or ASF processing and SMP, the decoder shall perform a frequency-to-time transformation utilizing the IMDCT tool specified in ETSI TS 103 190-1 [1], clause 5.5.

4.8.3.8 Simple Coupling (S-CPL)

If the decoder processes an `immersive_channel_element` where the *immersive_codec_mode* is either SCPL or ASPX_SCPL, the decoder shall utilize the S-CPL tool specified in clause 5.3. For the processing of all other channel elements or object audio, the S-CPL tool shall be bypassed.

4.8.3.9 QMF analysis

The output of the IMDCT tool (subsequently S-CPL processed if applicable) shall be transformed into the Quadrature Mirror Filter (QMF) spectral domain by the QMF analysis tool described in ETSI TS 103 190-1 [1], clause 5.7.3. The output of the QMF analysis for each track is one matrix $Q(ts, sb)$, where $0 \leq ts < num\_qmf\_timeslots$ and $0 \leq sb < num\_qmf\_subbands$.

4.8.3.10 Companding

4.8.3.10.1 Introduction

Use of the companding tool depends on the type of audio substream and the channel elements present in the substream. The companding tool is specified in ETSI TS 103 190-1 [1], clause 5.7.5. Clause 6.2.9 in the same document defines how companding is applied in channel audio.

4.8.3.10.2 Channel audio substream

When decoding a channel audio substream containing a `single_channel_element`, `channel_pair_element`, `3_0_channel_element`, `5_X_channel_element` or `7_X_channel_element`, the decoder shall utilize the companding tool as specified in ETSI TS 103 190-1 [1], clause 6.2.9.

4.8.3.10.3 Channel audio substream with an immersive channel element

When decoding a channel-audio substream containing an `immersive_channel_element`, where *immersive_codec_mode* is ASPX_AJCC, the decoder shall utilize the companding tool as described in ETSI TS 103 190-1 [1], clause 6.2.9 for the input channels L, R, C, Ls, and Rs, where this order is also the order of the channels in `companding_control`.

4.8.3.10.4 Object audio substream

In an object-audio substream, the objects are coded in channel elements as specified in clause 6.2.3.3. Objects can be coded in `single_channel_element`, `channel_pair_element`, `3_0_channel_element`, `5_X_channel_element`. For objects coded into these channel elements, the decoder shall utilize the companding tool as specified in ETSI TS 103 190-1 [1], clause 6.2.9 for the corresponding channel elements.

4.8.3.11 A-SPX

4.8.3.11.1 Introduction

For coding configurations specified in this clause, an AC-4 decoder shall process the QMF domain signals except for the LFE channel signal with the advanced spectral extension (A-SPX) tool specified in ETSI TS 103 190-1 [1], clause 5.7.6. ETSI TS 103 190-1 [1], clause 6.2.10 specifies how the A-SPX tool shall be utilized to process `single_channel_element`, `channel_pair_element`, `3_0_channel_element`, `5_X_channel_element`, and `7_X_channel_element`. The processing of these channel elements is also applicable for the processing of an object audio substream because objects are coded in some of the listed channel elements.

Table 8 summarizes how the decoder shall utilize the A-SPX tool to process an `immersive_channel_element` and a `22_2_channel_element`. The processing of these channel elements depends on the decoding mode, the codec mode and *b_5fronts*. If `immersive_codec_mode` is SCPL, or the `22_2_codec_mode` is SIMPLE, no A-SPX processing shall be performed. Because A-SPX processing is preceded by S-CPL processing, but before A-CPL or A-JCC is applied, the input signals to A-SPX are either named as channels or still as intermediate decoding signals (*A''*-*M''*), respectively.

Table 8: Channels processed by A-SPX tool

Channel element (see Note 1)	Decoding mode	Codec mode (see Note 2)	*b_5fronts*	Channels in A-SPX (see Note 3)
ice	Full	ASPX_SCPL	False	(Ls, Lb), (Rs, Rb), C, (L, R), (Tfl, Tbl), (Tfr, Tbr)
ice	Full	ASPX_SCPL	True	(Ls, Lb), (Rs, Rb), C, (L, Lscr), (R, Rscr), (Tfl, Tbl), (Tfr, Tbr)
ice	Core	ASPX_SCPL	X	[Ls], [Rs], C, (L, R), [Tfl], [Tfr]
ice	Full/core	ASPX_ACPL_1/ ASPX_ACPL_2	X	(A", B"), (D", E"), (F", G"), C"
ice	Full/core	ASPX_AJCC	X	(A", B"), (D", E"), C"
22	Only full decoding supported	ASPX	Not present	(L, R), (C, Tc), (Ls, Rs), (Lb, Rb), (Tfl, Tfr), (Tbl, Tbr), (Tsl, Tsr), (Tfc, Tbc), (Bfl, Bfr), (Bfc, Cb), (Lw, Rw)

NOTE 1: ice = `immersive_channel_element`, 22 = `22_2_channel_element`.
NOTE 2: Codec mode is one of {`immersive_codec_mode`, `22_2_codec_mode`}.
NOTE 3: Order matches `aspx_data_1ch()` and `aspx_data_2ch()` in the bitstream.
NOTE 4: Channels without parenthesis or bracket are processed with one `aspx_data_1ch()` element.
NOTE 5: Channels grouped by parentheses are processed together with one `aspx_data_2ch()` element.
NOTE 6: Channels in square brackets are processed with the first of two channels of one `aspx_data_2ch()` element.

4.8.3.11.2 Core decoding mode with ASPX_SCPL codec mode

When operating in core decoding mode with *immersive_codec_mode* = ASPX_SCPL:

- The decoder shall utilize the A-SPX postprocessing tool specified in clause 5.4 for the channels listed in table 9.

Table 9: Channels to be processed by the A-SPX post-processing tool

b_5fronts	channels
0	Ls, Rs, Tfl, Tfr
1	L, R, Ls, Rs, Tfl, Tfr

- The decoder shall apply a gain factor $g = 2$ to each output QMF subsample $Qout_{\mathrm{ASPX},a}(ts, sb)$ of output channel *a*, for $0 \leq ts < num\_qmf\_timeslots$ and $0 \leq sb < num\_qmf\_subbands$.

- If two input channels are processed in this operation, the decoder shall apply the same gain factor *g* to each output QMF subsample $Qout_{\mathrm{ASPX},b}(ts, sb)$ of output channel *b*, respectively.

4.8.3.11.3 Full decoding mode with ASPX_SCPL codec mode

When operating in full decoding mode with *immersive_codec_mode* = ASPX_SCPL, the decoding requirements depend on the value of *b_5fronts*.

When *b_5fronts* is false:

- The decoder shall apply a channel-dependent gain factor *g* to each of the output QMF subsamples $Qout_{\mathrm{ASPX},a}(ts, sb)$ of output channel *a* for $0 \leq ts < num\_qmf\_timeslots$ and $0 \leq sb < num\_qmf\_subbands$.

- If two input channels are processed in this operation, the decoder shall apply the same gain factor *g* to each output QMF subsample $Qout_{\mathrm{ASPX},b}(ts, sb)$ of output channel *b* respectively.

Table 10 shows the gain factor *g* for the processing of different input channels for *b_5fronts* is false.

Table 10: Channel-dependent gain for full decoding in immersive_codec_mode = ASPX_SCPL and b_5fronts is false

Input channels to A-SPX	Gain factor *g*
L, C, R	2
Ls, Lb, Rs, Rb, Tfl, Tbl, Tfr, Tbr	$\sqrt{2}$

When *b_5fronts* is true:

- The decoder shall apply a channel dependent gain factor *g* to each of the output QMF subsamples $Qout_{\mathrm{ASPX},a}(ts, sb)$ of output channel *a*, for $0 \leq ts < num\_qmf\_timeslots$ and $0 \leq sb < num\_qmf\_subbands$.

- If two input channels are processed in this operation, the decoder shall apply the same gain factor *g* to each output QMF subsample $Qout_{\mathrm{ASPX},b}(ts, sb)$ of output channel *b* respectively.

Table 11 shows the gain factor *g* for the processing of different input channels for *b_5fronts*is true.

Table 11: Channel-dependent gain for full decoding in immersive_codec_mode = ASPX_SCPL and b_5fronts = 1

Input channels to A-SPX	Gain factor *g*
C	2
L, Lscr, R, Rscr	1
Ls, Lb, Rs, Rb, Tfl, Tbl, Tfr, Tbr	$\sqrt{2}$

4.8.3.12 Advanced joint channel coding (A-JCC)

For the full decoding mode of a channel audio substream containing `immersive_channel_element` where `immersive_codec_mode` is ASPX_AJCC, the decoder shall utilize the A-JCC tool for full decoding mode as specified in clause 5.6.3.5.2.

For the core decoding mode of a channel audio substream containing `immersive_channel_element` where `immersive_codec_mode` is ASPX_AJCC, the decoder shall utilize the A-JCC tool for core decoding mode as specified in clause 5.6.3.5.3.

The input to the A-JCC tool for full decoding mode, or the A-JCC tool for core decoding mode, is the output of the preceding A-SPX tool.

4.8.3.13 Advanced Joint Object Coding (A-JOC)

For the full decoding mode of an object audio substream containing A-JOC coded content (`b_ajoc` is true), the decoder shall utilize the A-JOC tool as specified in clause 5.7.

The input to the A-JOC tool is the output of the preceding A-SPX tool.

4.8.3.14 Advanced coupling (A-CPL)

If the AC-4 decoder operates in full decoding mode, the decoder shall utilize the advanced coupling tool. For decoding of `immersive_channel_element`, the A-CPL tool is specified in clause 5.5.2. Table 12 shows which channels the decoder shall process. For decoding of `22_2_channel_element`, no A-CPL processing is required. For decoding of all other channel elements, the decoder shall utilize the A-CPL tool specified in ETSI TS 103 190-1 [1], clause 5.7.7. ETSI TS 103 190-1 [1], clause 6.2.11, specifies which channels the decoder shall process for these channel elements.

Table 12: Channels processed by A-CPL tool for immersive_channel_element

immersive_codec_mode	***b_5fronts***	**Channels to be processed by A-CPL**
ASPX_ACPL_1, ASPX_ACPL_2	0	C, L, R, (Ls, Lb), (Rs, Rb), (Tfl, Tbl), (Tfr, Tbr)
ASPX_ACPL_1, ASPX_ACPL_2	1	C, (L, Lscr), (R, Rscr), (Ls, Lb), (Right Side/Surround (Rs), Rb), (Tfl, Tbl), (Tfr, Tbr)
NOTE: Channels grouped by parentheses are processed together.		

If the AC-4 decoder operates in core decoding mode for decoding of `immersive_channel_element`, the decoder shall not utilize the A-CPL tool, but apply a gain value $g = 2$ to all QMF subsamples of each present channel instead (except for the LFE channel).

4.8.3.15 Dialogue enhancement

The application of dialogue enhancement is performed by different processes depending on the decoding mode and the substream type: Channel Audio Substream (CAS) and Object Audio Substream (OAS).

For CAS processing, the decoder shall utilize the dialogue enhancement processing tool depending on the input channel configuration and the coding mode as specified in table 13.

For OAS processing, the decoder shall utilize the dialogue enhancement processing tool depending on `b_ajoc` as specified in table 14.

Table 13: Dialogue enhancement tools for CAS processing

Input channel configuration	**Codec mode**	**Dialogue enhancement tool for core decoding**	**Dialogue enhancement tool for full decoding**
stereo, 5.X, 7.X	Any	ETSI TS 103 190-1 [1], clause 5.7.8	ETSI TS 103 190-1 [1], clause 5.7.8
7.X.4, 9.X.4	SCPL, ASPX_SCPL, ASPX_ACPL_1	ETSI TS 103 190-1 [1], clause 5.7.8	ETSI TS 103 190-1 [1], clause 5.7.8
7.X.4	ASPX_ACPL_2, ASPX_AJCC	ETSI TS 103 190-1 [1], clause 5.7.8	ETSI TS 103 190-1 [1], clause 5.7.8
9.X.4	ASPX_ACPL_2	Clause 5.8.2.2	ETSI TS 103 190-1 [1], clause 5.7.8
9.X.4	ASPX_AJCC	Clause 5.8.2.1	ETSI TS 103 190-1 [1], clause 5.7.8
22.2	Any	ETSI TS 103 190-1 [1], clause 5.7.8	ETSI TS 103 190-1 [1], clause 5.7.8

Table 14: Dialogue enhancement tools for OAS processing

`b_ajoc`	**Dialogue enhancement tool for core decoding**	**Dialogue enhancement tool for full decoding**
True	Clause 5.8.2.4	Clause 5.8.2.3
False	Clause 5.8.2.5	Clause 5.8.2.5

If b_de_simulcast is true, the decoder shall use the second de_data in dialog_enhancement for the core decoding mode.

Table 15 specifies which channels shall be processed by the dialogue enhancement tool specified in ETSI TS 103 190-1 [1], clause 5.7.8.

Table 15: Channels processed by the dialogue enhancement tool

Input channel configuration	Dialogue enhancement channels
Mono	C
Stereo	L, R
5.X, 7.X, 7.X.4	L, R, C
9.X.4, 22.2	Lscr, Rscr, C

4.8.3.16 Direct dynamic range control bitstream gain application

When decoding a channel audio substream with coded dynamic range control gain values present in the bitstream (drc_compression_curve_flag = 0), the decoder shall decode the gain values for the channel configurations listed in ETSI TS 103 190-1 [1], clauses 4.3.13.7.1, and 5.7.9.3.2. The decoder shall utilize the channel groups specified in table 69 to perform the gain-value decoding analogously to ETSI TS 103 190-1 [1], clause 5.7.9.3.2. The decoder shall apply the decoded gain values to the present channels as specified in ETSI TS 103 190-1 [1], clause 5.7.9.3.3. For core decoding mode the decoder should discard gain values which are assigned to channels that are not present in the core channel configuration.

4.8.3.17 Substream gain application for operation with associated audio

This clause describes the application of gains related to associated audio for presentations that include an associated audio substream.

Location of the associated audio gains

If the selected presentation has a presentation_version = 1 and b_associated is true in the ac4_presentation_substream associated with this presentation, the gains related to associated audio should be extracted from there.

If the selected presentation has a presentation_version = 0 and b_associated is true in the extended_metadata of the associated audio substream, the gains related to associated audio decoding should be extracted from there.

Associated audio user gain

Decoder systems should provide an option for consumers to control the level of the associated audio signal by applying a gain *g_assoc* $\in$ [-∞, 0] dB. If no gain is set, it shall default to 0 dB.

This control may be disabled if the assignment in the language_tag_bytes of the associated audio substream is set to one option with Mix Type premix - see ETSI TS 103 190-1 [1], clause 4.3.3.8.8.

Gain application

The resulting audio signal $Y\_associated_{ch}$ for each channel *ch* of the associated audio substream can be derived from the input signal $X\_associated_{ch}$ according to:

$$\mathrm{Y\_associated}_{\mathrm{ch}} = \mathrm{X\_associated}_{\mathrm{ch}} \times \mathrm{g\_assoc}$$

Scale main or music and effects substream

The decoding allows for a gain adjustment of the channels or audio objects in the main or music and effects substream if an adjustment needs to be done. If no gain values are given, a default of 0 dB shall be used.

For channel-audio substreams, the gain for the Centre channel of the main or music and effects substream, if available, should be adjusted first using scale_main_centre. Next, the gain for the two front channels L and R of the main or music and effects substream should be adjusted using scale_main_front. And finally, the gain for all channels of the main or music and effects substream should be adjusted using scale_main.

For object audio substreams, the gain of each audio object should be adjusted using a gain factor *g_main* which shall be interpolated depending on the position of the audio object. It shall be determined in the decibel (dB) domain according to:

$$\text{g\_main}(x, y, z) = c_c \times \text{scale\_main\_centre} + c_f \times \text{scale\_main\_front} + \text{scale\_main}$$

with the following position-dependent factors:

$$c_c = \begin{cases} (1 - |2x - 1|) \times (1 - 2y) \times (1 - 2z) & , \quad y < ½ \cap z < ½ \\ 0 & , \quad y \geq ½ \cup z \geq ½ \end{cases}$$

$$c_f = \begin{cases} |2x - 1| \times (1 - 2y) \times (1 - 2z) & , \quad y < ½ \cap z < ½ \\ 0 & , \quad y \geq ½ \cup z \geq ½ \end{cases}$$

where *x* = *object_position_X*, *y* = *object_position_Y* and *z* = *object_position_Z* are the object's OAMD coordinates as specified in clause 6.3.9.

Scale dialogue substream

If the selected presentation includes one or more dialogue substreams alongside an associated audio substream, the gain for all channels and objects of the dialogue substreams should be adjusted using `scale_main_centre`, `scale_main_front`, and `scale_main`, analogously to the main or music and effects substream. If no gain values are given, a default value of 0 dB shall be used.

4.8.3.18 Substream gain application for operation with dialogue substreams

This clause describes the application of gains related to dialogue for presentations that include a dialogue substream.

Location of the dialogue gains

If the selected presentation has a `presentation_version` = 1 and `b_dialog` is true in the `extended_metadata` of a substream, the gains related to decoding of this substream should be extracted from there.

If the selected presentation has a `presentation_version` = 0 and `b_dialog` is true in the `extended_metadata` of the dialogue substream, the gains related to decoding of dialogue substreams should be extracted from there.

Dialogue user gain

A single user-agent-provided gain *g_dialog* $\in$ [-∞, *g_dialog_max*] dB should be applied to all channels of the dialogue substream. This allows for a user-controlled adjustment of the relative dialogue level. The user-agent default value for *g_dialog* shall be 0 dB. If `b_dialog_max_gain` is true, *g_dialog_max* shall be retrieved from `dialog_max_gain` as defined in ETSI TS 103 190-1 [1], clause 4.3.12.4.11, and set to 0 otherwise.

The resulting audio signal Y_dialog_{ch} for each channel *ch* of the dialogue substream can be derived from the input signal X_dialog_{ch} according to:

$$\text{Y\_dialog}_{ch} = \text{X\_dialog}_{ch} \times \text{g\_dialog}$$

4.8.3.19 Substream rendering

Substream rendering describes the process of rendering the decoded channels or object essences to the output channel configuration. This process is significantly different for the two types of available substreams: channel audio substreams and object audio substreams:

- **Channel Audio Substream (CAS):**
  - The decoder shall utilize the channel renderer tool specified in clause 5.10.2 to render the decoded channels to the output channel configuration, including the consideration of downmix coefficients as described in the tool specification. Hence, the output of the substream rendering process for one CAS is one instance of the output channel configuration to be mixed by the following substream mixer.

- **Object Audio Substream (OAS):**
  - If b_isf is false, the decoder shall utilize an Object Audio Renderer (OAR) tool, not specified in the present document, to render each present object essence to one instance of the output channel configuration. A reference AC-4 object audio renderer that may be used as specified in ETSI TS 103 448 [i.2]. If b_isf is true, the decoder shall utilize the ISF renderer tool specified in clause 5.10.3 to render each present object to one instance of the output channel configuration. Hence, the output of the substream rendering process for one OAS comprises multiple instances of the output channel configuration to be mixed by the subsequent substream mixer. The input to the OAS for the rendering process of one object is the decoded object essence of the corresponding object plus its decoded associated object properties, provided by the decoder interface for object audio specified in annex F.

4.8.4 Mixing of decoded substreams

This clause describes the process of applying substream group gains to the channel configuration instances of the selected presentation as well as mixing these substreams into a single output channel configuration instance. The channel configuration instances are the output of the substreams renderers, as described in clause 4.8.3.19.

Application of substream group gains

For presentations with presentation_version = 1, the respective substream group gain g_{sg}, as defined in table 70, shall be applied to the audio signal $X_{s,sg}$ of each substream *s* which is part of a substream group *sg* according to:

$$Y_{s,sg} = g_{sg} \times X_{s,sg}$$

Mixing of substreams

The actual mixing is done for each channel *ch* by summing up that channel $X_{s,ch}$ of each substream *s*, according to:

$$Y_{ch} = \frac{\sum_{s=0}^{n\_sub-1} X_{s,ch}}{n\_sub}$$

where *n_sub* is the total number of substreams that belong to the presentation to be decoded.

4.8.5 Loudness correction

4.8.5.1 Introduction

The loudness of the audio signal is determined by the dialnorm value. The AC-4 bitstream syntax supports presence of additional loudness correction data for the cases where:

- downmixing to a lower channel configuration;
- decoding of an alternative presentation;
- real-time loudness correction data (derived from real-time loudness estimates) is available.

The following clauses refer to the corresponding bitstream data and specify how to apply the loudness correction.

4.8.5.2 Dialnorm location

For presentations with presentation_version = 1, the dialnorm value shall be extracted from the ac4_presentation_substream.

For presentations with presentation_version = 0, the dialnorm value shall be derived from the following sources:

- the basic_metadata of the associated substream for presentations containing associated audio; and
- the substream indicated in table 16 for main audio decoding.

Table 16: Substream containing valid dialnorm information

`presentation_config`	Substream
0	dialogue
1	Main
2	Main
3	dialogue
4	Main
5	Main or dialogue

4.8.5.3 Downmix loudness correction

When downmixing is done in the decoder, the loudness shall be adjusted using the output channel-specific loudness correction factor from the `loud_corr` element that relates to the selected downmix.

EXAMPLE: When transmitting 7.1.4 content and downmixing it to 7.1, the loudness correction factor *loud_corr_gain_7_X* is used:

$$s_{\text{loud\_corr,ch}}(\text{ts,sb}) = 2^{\text{loud\_corr\_gain\_OUT\_CH\_CONF}/6} \times s_{\text{in,ch}}(\text{ts,sb})$$

where $0 \leq ts < num\_qmf\_timeslots$ and $0 \leq sb < num\_qmf\_subbands$.

Here, $s_{\text{in},ch}$ refers to the samples of each input channel *ch* and $s_{\text{loud\_corr},ch}$ refers to the samples of each output channel *ch*.

Once a downmix loudness correction factor has been received, this factor is valid until an update is transmitted. A default value of 0 dB should be used until the first reception of a loudness correction factor.

4.8.5.4 Alternative presentation loudness correction

When decoding an alternative presentation, i.e. an AC-4 presentation with `b_alternative` is true, a target-specific loudness correction shall be applied:

$$s_{\text{target\_corr,ch}}(\text{ts,sb}) = 2^{\text{target\_corr\_gain}/6} \times s_{\text{in,ch}}(\text{ts,sb})$$

where $0 \leq ts < num\_qmf\_timeslots$ and $0 \leq sb < num\_qmf\_subbands$.

Here, $\boldsymbol{s}_{\text{in},ch}$ refers to the samples of each input channel *ch* and $\boldsymbol{s}_{\text{target\_corr},ch}$ refers to the samples of each output channel *ch*. *target_corr_gain* is the target-specific loudness correction factor specified for the target-device category (see table 67) that matches the playback device. If target-specific loudness correction factors are specified for some target-device categories only, these factors are used for the unspecified target-device categories according to table 17.

Table 17: Fallback target loudness correction factors

Output channel configuration	Target device category	First fallback	Second fallback
Stereo	1D	2D	3D
5.X, 7.X	2D	3D	1D
5.X.2, 5.X.4, 7.X.2, 7.X.4, 9.X.4	3D	2D	1D
Stereo	Portable	No fallback	No fallback

4.8.5.5 Real-time loudness correction data

When real-time loudness correction data *rtll_comp_gain* is present in the bitstream, this loudness correction factor shall be applied as:

$$s_{\text{rtll\_comp,ch}}(\text{ts,sb}) = 10^{\text{rtll\_comp\_gain}/20} \times s_{\text{in,ch}}(\text{ts,sb})$$

where $0 \leq ts < num\_qmf\_timeslots$ and $0 \leq sb < num\_qmf\_subbands$.

Here, $\boldsymbol{s}_{\text{in},ch}$ refers to the samples of each input channel *ch* and $\boldsymbol{s}_{\text{rtll\_comp},ch}$ refers to the samples of each output channel *ch*.

4.8.6 Dynamic range control

NOTE: Only gains originating from compression curves are applied in this clause. Directly transmitted gains are applied in the substream decoding process as specified in clause 4.8.3.16.

The decoder shall utilize the dynamic range control (DRC) tool specified in ETSI TS 103 190-1 [1], clause 5.7.9 and DRC metadata `drc_frame` to apply gains to the channels in order to adjust the dynamic range of the output signal. For processing of the `immersive_channel_element` and the `22_2_channel_element`, the decoder shall derive the number of processed channels and the grouping of the corresponding channels from table 69.

If the presentation to be decoded contains an `ac4_presentation_substream`, `drc_frame` shall be extracted from this one.

If the presentation to be decoded does not contain an `ac4_presentation_substream`, `drc_frame` shall be extracted from `metadata`, which is present in the `ac4_substream`. If more than one instance of `ac4_substream` is present, the decoder may be set to select `ac4_substream` according to the substream that provides the `dialnorm` value for this presentation as specified in clause 4.8.5.2.

The DRC tool requires two inputs per channel *ch*: the audio signal that is the output from a preceding tool (usually loudness correction), $\mathbf{Qin1}_{\mathbf{DRC},ch}$, and the signal that is used to measure levels and drive the side chain, $\mathbf{Qin2}_{\mathbf{DRC},ch}$.

The $\mathbf{Qin2}_{\mathbf{DRC}}$ signal driving the side chain should be the signal that is present at the input to dialogue enhancement (i.e. it does not include dialogue enhancement) as specified in clause 4.8.3.15. Alternatively, the side chain signal $\mathbf{Qin2}_{\mathbf{DRC}}$ may be driven with $\mathbf{Qin1}_{\mathbf{DRC}}$.

4.8.7 QMF synthesis

A QMF synthesis filter shall transform each audio channel from the QMF domain back to the time domain as specified in ETSI TS 103 190-1 [1], clause 5.7.4.

4.8.8 Sample rate conversion

For values of `frame_rate_index` not equal to 13, the following requirements apply:

- The decoder shall be operated at external sampling frequencies of 48 kHz, 96 kHz, or 192 kHz.
- The decoder shall utilize the frame-rate control tool specified in clause 5.11 to adjust the sampling frequency to the external sampling frequency.
- The decoder shall use the resampling ratio as specified in ETSI TS 103 190-1 [1], clause 4.3.3.2.6.

For `frame_rate_index` = 13, the decoder may be operated at any external sampling frequency.

5 Algorithmic details

5.1 Bitstream processing

5.1.1 Introduction

Bitstream tools assemble, multiplex, or select the correct parts of the bitstream for processing.

All the substream tools refer to bits that are parsed from the bitstream. Before substream parsing can commence, preparatory steps sometimes need to be taken:

- Clause 5.1.2 specifies where to locate the correct parts when presentations have been divided into separate elementary streams.
- Clause 5.1.3 specifies a pre-collection step before the substream decoder starts.

5.1.2 Elementary stream multiplexing tool

The ESM tool combines substreams from multiple bitstreams according to presentation structure and selection.

AC-4 allows distributing a presentation over multiple elementary streams. The ESM tool takes multiple AC-4 elementary streams as input, and selects the appropriate substreams of each to present to the decoder for further processing in a single instance.

On TOC level, the `ac4_presentation_v1_info()`indicates the presentation configuration as described in clause 6.3.2.2.2:

- If the `b_multi_pid` bit in the `ac4_presentation_v1_info()` element is false, the presentation is fully contained within a single AC-4 elementary stream, and each substream belonging to the presentation can be referenced from the `substream_index_table` as contained in the Table Of Contents (TOC).
- If the `b_multi_pid` bit in the `ac4_presentation_v1_info()` element is true, the presentation is split over more than one AC-4 elementary stream, and not all substreams required by the presentation are contained within a single AC-4 elementary stream. In this case the `substream_info_*` elements of the TOC do not contain information about the substream location (and a substream is not included). It is then assumed that system level signalling provides information about which elementary streams are necessary to fully decode the presentation.

In the latter case, the ESM tool shall examine the TOC of all available AC-4 streams for matching `ac4_presentation_v1_info()` and `ac4_substream_group_info()` elements. These elements, in combination, provide all necessary references to substreams.

NOTE 1: Details of how to identify matching presentations are outside the scope of the present document.

NOTE 2: The compatibility indication (see clause 6.3.2.2.3) signals the compatibility level necessary for decoding, rendering and mixing all substreams, regardless of whether they are contained in one elementary stream or distributed. Thus, all matching presentations share the same value of `md_compat`.

The ESM tool shall then provide the collated presentation information to subsequent processing steps. Details are implementation-dependent; in a straightforward implementation, it may merge all the `ac4_presentation_v1_info()` and `ac4_substream_group_info()` elements, as well as the substream payloads, into a new multiplex, building a self-contained elementary stream.

EXAMPLE: Figure 6 shows a simple example with two input AC-4 bitstreams, ES1 and ES2, both containing the same presentation information. Two presentations exist that offer the possible combinations of music and effects with English dialogue or music and effects with French dialogue. The ES1 AC-4 bitstream contains both the music and effects and English dialogue substreams, and the `substream_index_table` indicates the location of these in the bitstream. However, it does not indicate the location for the French dialogue substream, as it is not contained in ES1. ES2 contains the French dialogue and a `substream_index_table` entry for it. The ESM tool combines both ES1 and ES2 to produce an output that contains all the substreams required by each presentation, be that music and effects with English or French dialogue. This tool also updates `ac4_substream_group_info()` and `substream_index_table` to reflect the new bitstream structure.

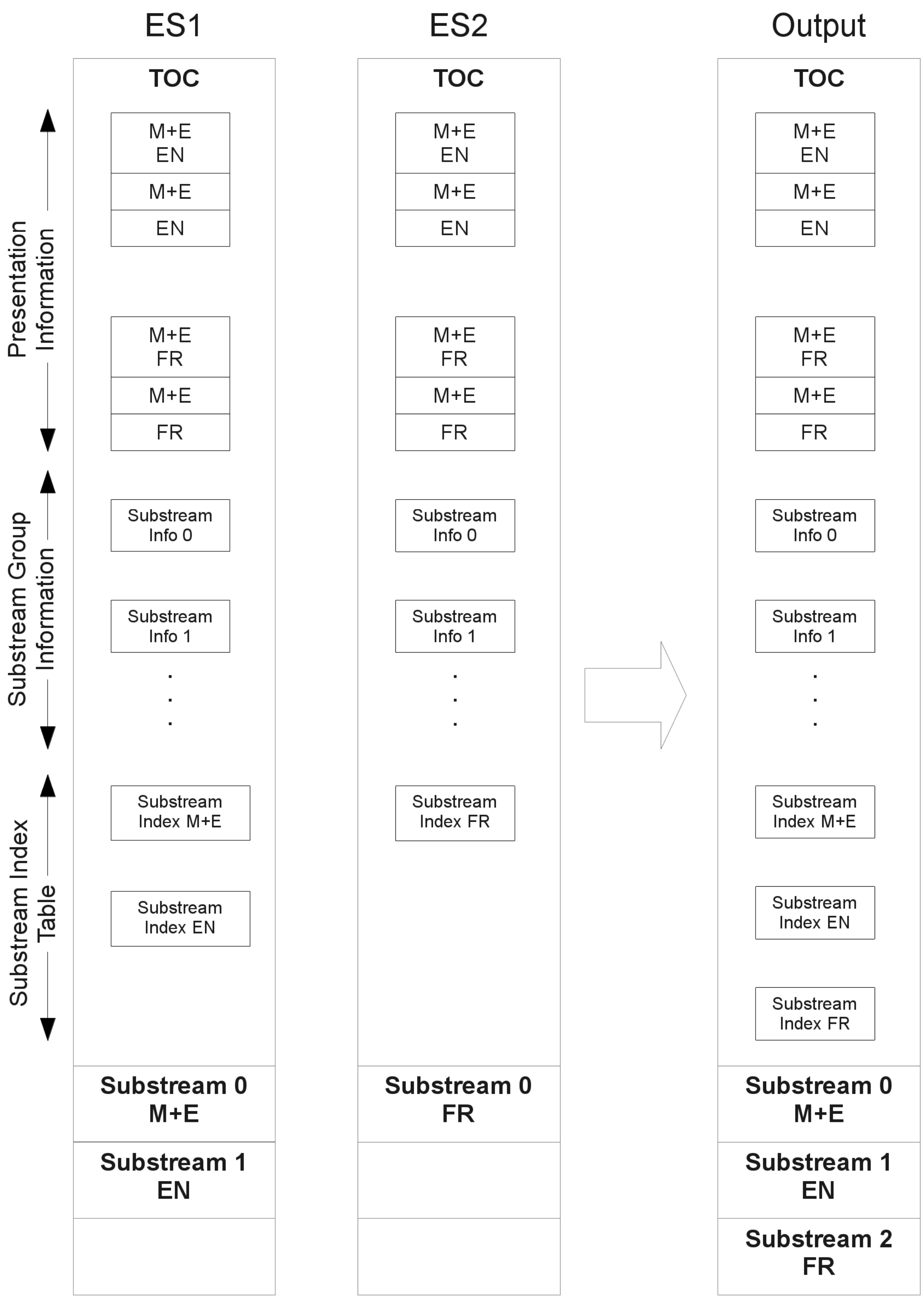


Figure 6: music and effects with English and French dialogue, distributed over two elementary streams

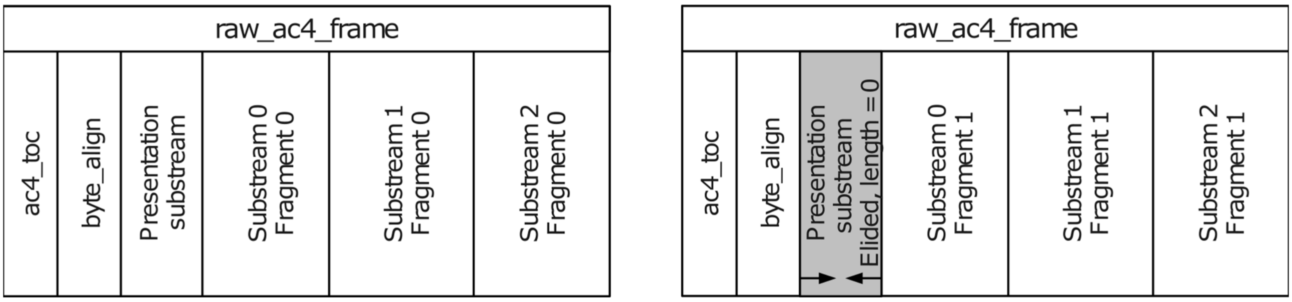
5.1.3 Efficient high frame rate mode

AC-4 is capable of aligning the frame rates with video signals of up to 120 frames per second. To improve encoding efficiency at high frame rates, the present document introduces an efficient high frame rate mode.

The efficient high frame rate mode is enabled on a per-presentation basis. In this mode, the codec receives AC-4 frames at the transmission frame rate. The decoder assembles a decodable audio payload from a group of *frame_rate_fraction* = 2 or *frame_rate_fraction* = 4 frames of the elementary stream at the nominal frame rate. Assembling a decodable audio payload starts at a frame where $\text{sequence\_counter} \mathit{mod} \text{frame\_rate\_fraction} \equiv 0$ (called "the first frame"). Assembling ends at a frame where $(\text{sequence\_counter} + 1) \mathit{mod} \text{frame\_rate\_fraction} \equiv 0$ (called "the last frame"). When assembling is complete, the decoder can decode the entire audio payload.

The first frame contains an unfragmented `ac4_presentation_substream`. Each successive `raw_ac4_frame()` contains the same number *n_substreams* of substream fragments. See figure 7 for an example.

NOTE 1: Substream fragment sizes of zero length are possible.



NOTE: *n_substreams* = 4.

Figure 7: Example of fragmented payload

The efficient high frame rate mode is active in a bitstream when all of the following conditions are met:

- the `bitstream_version` is 1 or greater;
- the frame rate of the stream as indicated by `frame_rate_index` is larger than 30 fps (frames per second); and
- the *frame_rate_fraction* as transmitted in `frame_rate_fractions_info` is 2 or 4.

The resulting audio frame rates are shown in table 18.

Table 18: Determining the codec internal audio frame rate

Stream frame_rate_index	Stream frame rate [fps]	frame_rate_fraction	audio_frame_rate_index	Audio frame rate [fps]
5	47,95	2	0	23,976
6	48	2	1	24
7	50	2	2	25
8	59,94	2	3	29,97
9	60	2	4	30
10	100	2	7	50
		4	2	25
11	119,88	2	8	59,94
		4	3	29,97
12	120	2	9	60
		4	4	30

To implement the feature, a decoder shall provide a first-in-first-out (FIFO) input buffer capable of storing partial frames.

NOTE 2: This increases the decoder latency by *frame_rate_fraction* - 1 frames.

Frames are pushed into the FIFO buffer as they arrive from the system interface. The decoder shall process frames in units, where each unit consists of a sequence of *frame_rate_fraction* consecutive frames, starting with a frame where $\text{sequence\_counter}\,mod\,\text{frame\_rate\_fraction} \equiv 0$.

To process a unit, the decoder shall reassemble the frame by concatenating all the `ac4_substream_data` fragments that are referenced in the selected presentation.

NOTE 3: The control data delay as specified in ETSI TS 103 190-1 [1], clause 5.6.2 provides smooth operation across source changes.

EXAMPLE: A possible implementation of the FIFO is shown in figure 8.

NOTE 4: f=*frame_rate_fraction*; c=*sequence_counter*.

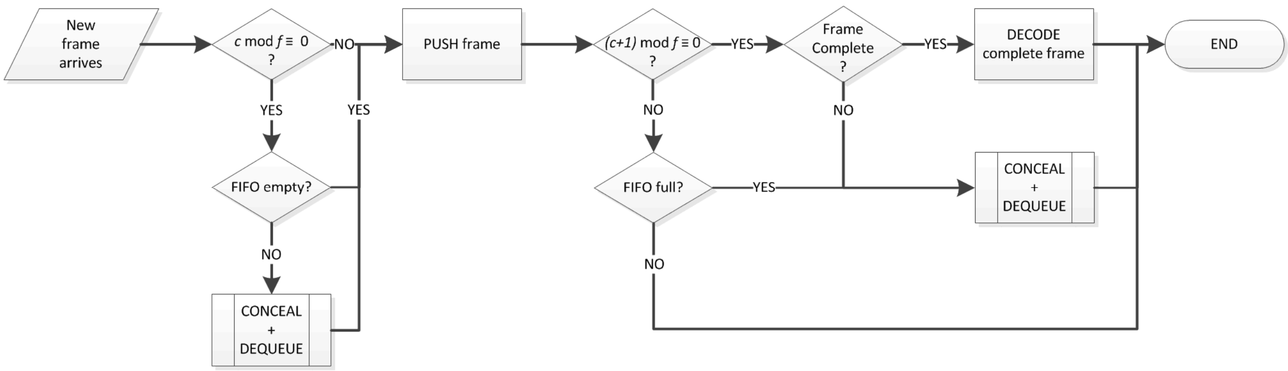


Figure 8: Framing algorithm

See also clause 4.5.3, and clause 6.3.2.4.

5.2 Stereo and Multichannel Processing (SMP) for immersive audio

5.2.1 Introduction

This tool extends the SMP tool as specified in ETSI TS 103 190-1 [1], clause 5.3, by introducing additional processing modes for the `channel_data_elements` specified in clause 6.2.4.

For all the `channel_data_elements` specified in ETSI TS 103 190-1 [1], clause 4.2.6, SMP shall be applied according to ETSI TS 103 190-1 [1], clause 5.3.3. The present document additionally specifies the requirements for processing `immersive_channel_element` as well as `22_2_channel_element`.

The multichannel tools take their input from the audio spectral frontend, receiving n-tuples of spectra with matching time/frequency layout. The multichannel processing tool applies a number of time- and frequency-varying matrix operations on these tuples. Between two and twenty-two spectra are transformed at a time.

Parameters for the transformations are transmitted by `chparam_info()` elements; the various transform matrices **M** (up to 22 x 22) are built up from these parameters as described in the following paragraphs.

When the tuples have been transformed, they are passed on to the IMDCT transform, defined in ETSI TS 103 190-1 [1], clause 5.5. Generally, the input to the multichannel processing tool does not have a "channel" meaning. For the processing of `immersive_channel_element`, the output of the SMP tool represents intermediate decoding signals. For the processing of all other channel elements, the output from the tool carries channels ordered as L, R, C, Ls, Rs, etc.

The general processing flow of the tool is illustrated in figure 9.

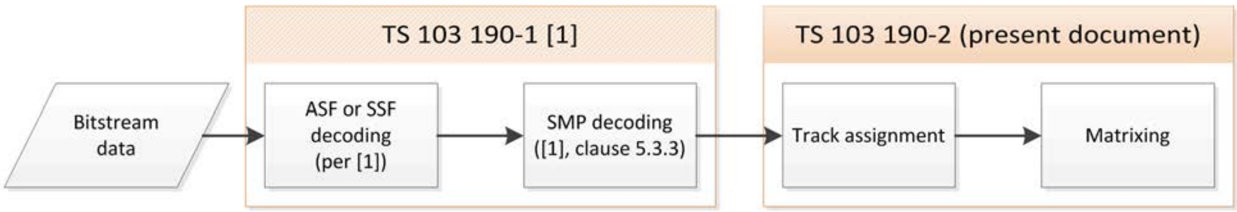


Figure 9: General processing flow in the extended SMP tool

5.2.2 Interface

5.2.2.1 Inputs

The input to the SMP tool are scaled spectral lines of tracks derived from decoding the `sf_data` element stored in:

- a `channel_pair_element` ($n_{SAP} = 2$);
- a `3_0_channel_element` ($n_{SAP} = 3$);
- a `5_X_channel_element` ($n_{SAP} = 5$);
- a `7_X_channel_element` ($n_{SAP} = 7$);
- an `immersive_channel_element` ($n_{SAP} = 11/13$); or
- a `22_2_channel_element` ($n_{SAP} = 22$);

using the ASF tool:

$\mathbf{s}_{SMP,[0|1|\ldots]}$ Up to n_{SAP} vectors of spectral lines, each vector representing a track decoded from an `sf_data` element.

NOTE: The tracks are numbered according to their occurrence in the bitstream, starting from track $\mathbf{s}_{SMP,0}$.

5.2.2.2 Outputs

The outputs from the extended SMP tool are n_{SAP} blocks of scaled spectral lines:

$S_{\mathrm{SMP,[A''|B''|C''|...]}}$ n_{SAP} vectors of *blk_len* spectral lines assigned to intermediate decoding signals. In most cases these signals are implicitly assigned to channels (L, R, C, …) with discrete speaker locations.

NOTE: See clause A.3 for a listing of channel abbreviations.

5.2.2.3 Controls

The bitstream and additional information used by the stereo audio processing tool is:

blk_len Block length. Equal to the number of input and output spectral lines in one channel.

sap_used[g][sfb] Array indicating the operating mode of the stereo audio processing tool for group *g* and scale factor band *sfb*.

sap_gain[g][sfb] Array of real-valued gains for group *g* and scale factor band *sfb*.

5.2.3 Processing the immersive_channel_element

5.2.3.1 Introduction

The `immersive_channel_element` enables the immersive channel configurations listed in clause A.3, in table A.31, columns "5.X.0" through "9.X.4". The `immersive_channel_element` provides a number of audio tracks derived from different combinations of channel data elements (see ETSI TS 103 190-1 [1], clause 5.3.3), similarly to the `5_X_channel_element` and the `7_X_channel_element`.

The following clauses specify the processing for different settings of the *immersive_codec_mode*.

5.2.3.2 immersive_codec_mode ∈ {SCPL, ASPX_SCPL, ASPX_ACPL_1}

This clause defines stereo/multichannel processing when *immersive_codec_mode* ∈ {SCPL, ASPX_SCPL, ASPX_ACPL_1}.

1) Tracks O_0, O_1, …, O_{12} shall be produced as specified in ETSI TS 103 190-1 [1], clause 5.3.3.

NOTE 1: If *b_5fronts* is false, the tool operates only on 11 input tracks. In this case, all subsequent operations on tracks O_{11}, O_{12} (and [L,M] further down) can be disregarded.

2) Tracks O_0, O_1, …, O_{12} shall be assigned to internal output tracks *A*, *B*, *C*, … as specified in table 19.

Table 19: Track assignment

core_5ch_grouping	0 (see note 2)			1		2		3	
2ch_mode		**0**	**1**	**N/A**					
Element	**Input**	**Output**		**Input**	**Output**	**Input**	**Output**	**Input**	**Output**
`mono_data`	[6]	[C]	[C]	-	-	[6]	[C]	-	-
1st `two_channel_data`	[0,1]	[A,B]	[A,D]	[3,4]	[D,E]	[4,5]	[F,G]	[5,6]	[F,G]
2nd `two_channel_data`	[2,3]	[D,E]	[B,E]	[5,6]	[F,G]	[7,8]	[H,I]	[7,8]	[H,I]
3rd `two_channel_data`	[4,5]	[F,G]	[F,G]	[7,8]	[H,I]	[9,10]	[J,K]	[9,10]	[J,K]
4th `two_channel_data`	[7,8]	[H,I]	[H,I]	[9,10]	[J,K]	[11,12]	[L,M]	[11,12]	[L,M]
5th `two_channel_data`	[9,10]	[J,K]	[J,K]	[11,12]	[L,M]	-	-	-	-
6th `two_channel_data`	[11,12]	[L,M]	[L,M]	-	-	-	-	-	-
`three_channel_data`	-	-		[0,1,2]	[A,B,C]	-	-	-	-
`four_channel_data`	-	-		-	-	[0,1,2,3]	[A,B,D,E]	-	-
`five_channel_data`	-	-		-	-	-	-	[0,1,2,3,4]	[A,B,C,D,E]
NOTE 1: Tracks O_i are labelled [i] in this table for convenience of notation. NOTE 2: When `core_5ch_grouping` = 0, the assignment of input tracks from the first two `two_channel_data` elements to the output signals depends on the element `2ch_mode`.									

EXAMPLE: Let `core_5ch_grouping` = 1. Processing the first `two_channel_data` element (specified in ETSI TS 103 190-1 [1], clause 5.3.3) produces outputs O_0,O_1. The outputs are assigned to tracks E, D. These are input to the next step.

3) Determine parameters a_i, b_i, c_i, d_i ($i \in \{0,1\}$).

- If `b_use_sap_add_ch` is true, the parameters a_i, b_i, c_i, d_i ($i \in \{0,1\}$) shall be read from the contained `chparam_info` elements as specified in ETSI TS 103 190-1 [1], clause 5.3.2.

- Otherwise, the parameters shall be determined as follows: $a_i = d_i = 1$, $b_i = c_i = 0$.

4) Process tracks D, E, F, G as follows:

$$\begin{bmatrix} D' \\ F' \\ E' \\ G' \end{bmatrix} = \begin{bmatrix} a_0 & b_0 & 0 & 0 \\ c_0 & d_0 & 0 & 0 \\ 0 & 0 & a_1 & b_1 \\ 0 & 0 & c_1 & d_1 \end{bmatrix} \times \begin{bmatrix} D \\ F \\ E \\ G \end{bmatrix}$$

NOTE 2: Only the signals *D*, *E*, *F*, and *G* are modified in this step; all others are passed through into *A'* through *C'* and *F'* through *M'*.

5) Determine parameters a'_j.

- If the `sap_mode` = `full SAP`, the parameters a'_j shall be extracted from *n_elem* `chparam_info` elements, where $\text{n\_elem} = \begin{cases} 6 & if\,\text{b\_5fronts} \neq 0 \\ 4 & else \end{cases}$ and $0 \leq j < n\_elem$.

- Otherwise, the parameters a'_j shall be set to 0.

6) Produce the outputs of the stereo and multichannel tool as specified in table 20.

NOTE 3: These outputs are not assigned to dedicated channels until they have passed either one of the coupling tools (S-CPL/A-CPL) or the A-JCC tool.

Table 20: Matrixing

b_5fronts	Mapping
0	$\begin{bmatrix} A'' \\ B'' \\ C'' \\ D'' \\ E'' \\ F'' \\ G'' \\ H'' \\ I'' \\ J'' \\ K'' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a'_0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a'_1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & a'_2 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a'_3 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} A' \\ B' \\ C' \\ D' \\ E' \\ F' \\ G' \\ H' \\ I' \\ J' \\ K' \end{bmatrix}$
1	$\begin{bmatrix} A'' \\ B'' \\ C'' \\ D'' \\ E'' \\ F'' \\ G'' \\ H'' \\ I'' \\ J'' \\ K'' \\ L'' \\ M'' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a'_0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a'_1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & a'_2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a'_3 & 0 & 0 & 0 & 1 & 0 & 0 \\ a'_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & a'_5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} A' \\ B' \\ C' \\ D' \\ E' \\ F' \\ G' \\ H' \\ I' \\ J' \\ K' \\ L' \\ M' \end{bmatrix}$

5.2.3.3 immersive_codec_mode = ASPX_ACPL_2

This clause defines processing when *immersive_codec_mode* = ASPX_ACPL_2.

1) Tracks O_0, O_1, …, O_6 shall be produced as specified in ETSI TS 103 190-1 [1], clause 5.3.3.

2) Tracks *A*, *B*, *C*, *D*, *E*, *F*, *G* shall be assigned to tracks *A'* through *G'* as specified in step 2 in clause 5.2.3.2.

3) The rest of the tracks shall be filled with silence.

   - If `b_5fronts` is false, silence tracks *H'* through *K'*.
   - If `b_5fronts` is true, silence tracks *H'* through *M'*.

4) The outputs of the stereo and multichannel tool are tracks *A'* through *K'*/*M'*.

5.2.3.4 immersive_codec_mode = ASPX_AJCC

This clause defines processing when *immersive_codec_mode* = ASPX_AJCC.

1) Tracks O_0,O_1,…,O_4 shall be produced as specified in ETSI TS 103 190-1 [1], clause 5.3.3.

2) Tracks *A*, *B*, *C*, *D*, *E* shall be assigned to tracks *A'* through *E'* as specified in step 2 in clause 5.2.3.2, where *b_5fronts* is false.

3) The rest of the tracks shall be filled with silence.

   - If *b_5fronts* false, silence tracks *F'* through *K'*.
   - If *b_5fronts* is true, silence tracks *F'* through *M'*.

4) The outputs of the stereo and multichannel tool are tracks *A'* through *K'*/*M'*.

5.2.4 Processing the 22_2_channel_element

The 22_2_channel_element enables the channel configuration for 24 channels (including two LFE channels) as described in table A.27, column "22.2". The 22_2_channel_element comprises two mono_data elements and 11 two_channel_data elements.

These channel data elements shall be processed according to ETSI TS 103 190-1 [1], clause 5.3.3. The processed elements shall be assigned to output channels as shown in table 21.

Table 21: Input and output mapping for 22_2_codec_mode {SIMPLE, ASPX}

element	Input	Output
mono_data[0]	[0]	[LFE]
mono_data[1]	[1]	[LFE2]
two_channel_data[0]	[2,3]	[L, R]
two_channel_data[1]	[4,5]	[C, Tc]
two_channel_data[2]	[6,7]	[Ls,Rs]
two_channel_data[3]	[8,9]	[Lb, Rb]
two_channel_data[4]	[10,11]	[Tfl, Tfr]
two_channel_data[5]	[12,13]	[Tbl,Tbr]
two_channel_data[6]	[14,15]	[Tsl,Tsr]
two_channel_data[7]	[16,17]	[Tfc,Tbc]
two_channel_data[8]	[18,19]	[Bfl,Bfr]
two_channel_data[9]	[20,21]	[Bfc,Cb]
two_channel_data[10]	[22,23]	[Lw,Rw]

5.3 Simple Coupling (S-CPL)

5.3.1 Introduction

The simple coupling tool operates in the time domain, processing the output of the IMDCT. The simple coupling tool is used on signals that are coded in an `immersive_channel_element` when *immersive_codec_mode* $\in$ {SCPL, ASPX_SCPL}.

5.3.2 Interface

5.3.2.1 Inputs

Xin $_{\mathbf{SCPL,[A|B|\ldots]}}$ $n_{\mathrm{SCPL,in}}$ time domain signals, each being an IMDCT processed output signal of the SMP tool.

The **Xin** $_{\mathrm{SCPL}}$ signals each consist of *frame_length* PCM audio samples. The number of S-CPL input signals, $n_{\mathrm{SCPL,in}}$, depends on *b_5fronts* as indicated in table 22.

Table 22: Number of S-CPL input signals

b_5fronts	$n_{\mathrm{SCPL,in}}$
False	11
True	13

5.3.2.2 Outputs

Xout $_{\mathbf{SCPL,[a|b|\ldots]}}$ $n_{\mathrm{SCPL,out}}$ decoupled time signals.

The **Xout** $_{\mathrm{SCPL}}$ signals each consist of *frame_length* PCM audio samples. The number of S-CPL output signals, $n_{\mathrm{SCPL,out}}$, equals $n_{\mathrm{SCPL,in}}$. For core decoding, $n_{\mathrm{SCPL,out}}$ is limited to a maximum of seven channels.

5.3.3 Reconstruction of the output channels

5.3.3.1 Full decoding

In full decoding mode, the output channels (L, R, C, …) of the simple coupling tool are created using a multiplication of a matrix $\mathbf{M}_{\mathrm{SCPL}}$ with the 11 or 13 IMDCT processed output signals (A", B", C", …) of the stereo and multichannel processing tool (see table 20). The descriptors (A", B", C", …) are re-used to clarify the connection between these two tools.

The output channels shall be created as specified in table 23, where the values of *c_gain* and *m_gain* are assigned as follows:

$$\mathrm{c\_gain} = \begin{cases} 2 & if\,\mathrm{immersive\_codec\_mode} = \mathrm{SCPL} \\ 1 & if\,\mathrm{immersive\_codec\_mode} = \mathrm{ASPX\_SCPL} \end{cases}$$

$$\mathrm{m\_gain} = \begin{cases} \sqrt{2} & if\,\mathrm{immersive\_codec\_mode} = \mathrm{SCPL} \\ 1 & if\,\mathrm{immersive\_codec\_mode} = \mathrm{ASPX\_SCPL} \end{cases}$$

Table 23: S-CPL channel mapping for immersive_codec_mode {SCPL, ASPX_SCPL} for full decoding

b_5fronts	Output channel mapping
0	$\begin{bmatrix} L \\ R \\ C \end{bmatrix} = \text{c\_gain} \times \begin{bmatrix} A'' \\ B'' \\ C'' \end{bmatrix}$ $\begin{bmatrix} Ls \\ Lb \\ Rs \\ Rb \\ Tfl \\ Tbl \\ Tfr \\ Tbr \end{bmatrix} = \text{m\_gain} \times 2 \times \begin{bmatrix} ½ & ½ & 0 & 0 & 0 & 0 & 0 & 0 \\ ½ & -½ & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & ½ & ½ & 0 & 0 & 0 & 0 \\ 0 & 0 & ½ & -½ & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & ½ & ½ & 0 & 0 \\ 0 & 0 & 0 & 0 & ½ & -½ & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & ½ & ½ \\ 0 & 0 & 0 & 0 & 0 & 0 & ½ & -½ \end{bmatrix} \times \begin{bmatrix} D'' \\ H'' \\ E'' \\ I'' \\ F'' \\ J'' \\ G'' \\ K'' \end{bmatrix}$
1	$[C] = \text{c\_gain} \times [C'']$ $\begin{bmatrix} Lw \\ Lscr \\ Rw \\ Rscr \end{bmatrix} = 2 \times \begin{bmatrix} ½ & ½ & 0 & 0 \\ ½ & -½ & 0 & 0 \\ 0 & 0 & ½ & ½ \\ 0 & 0 & ½ & -½ \end{bmatrix} \times \begin{bmatrix} A'' \\ L'' \\ B'' \\ M'' \end{bmatrix}$ $\begin{bmatrix} Ls \\ Lb \\ Rs \\ Rb \\ Tfl \\ Tbl \\ Tfr \\ Tbr \end{bmatrix} = \text{m\_gain} \times 2 \times \begin{bmatrix} ½ & ½ & 0 & 0 & 0 & 0 & 0 & 0 \\ ½ & -½ & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & ½ & ½ & 0 & 0 & 0 & 0 \\ 0 & 0 & ½ & -½ & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & ½ & ½ & 0 & 0 \\ 0 & 0 & 0 & 0 & ½ & -½ & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & ½ & ½ \\ 0 & 0 & 0 & 0 & 0 & 0 & ½ & -½ \end{bmatrix} \times \begin{bmatrix} D'' \\ H'' \\ E'' \\ I'' \\ F'' \\ J'' \\ G'' \\ K'' \end{bmatrix}$

5.3.3.2 Core decoding

In core decoding mode, the output channels (L, R, C, …) of the simple coupling tool are created by processing of the first $\mathbf{n}_{\text{SCPL,out}}$ processed IMDCT output signals (A", B", C", …) of the stereo and multichannel tool (see table 20) and assigning them to the output channels.

The output channels shall be created as specified in table 24, where the value of *c_gain* is assigned as follows:

$$\text{c\_gain} = \begin{cases} 2 & if \, \text{immersive\_codec\_mode} = \text{SCPL} \\ 1 & if \, \text{immersive\_codec\_mode} = \text{ASPX\_SCPL} \end{cases}$$

Table 24: S-CPL channel mapping for immersive_codec_mode {SCPL, ASPX_SCPL} for core decoding

Output channel mapping
$\begin{bmatrix} L \\ R \\ C \\ Ls \\ Rs \\ Tfl \\ Tfr \end{bmatrix} = \text{c\_gain} \times \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} A'' \\ B'' \\ C'' \\ D'' \\ E'' \\ F'' \\ G'' \end{bmatrix}$

5.4 Advanced Spectral extension (A-SPX) postprocessing tool

5.4.1 Introduction

The A-SPX post processing tool applies a gain factor of -1,5 dB to the input signal(s) and passes it to the output.

5.4.2 Interface

5.4.2.1 Inputs

$\mathbf{Qin}_{\mathbf{ASPX\_PP,[0|1|2|...]}}$ *num_sig* complex-valued QMF matrices.

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

If `b_5fronts` is true, *num_sig* is 6; otherwise, *num_sig* is 4.

5.4.2.2 Outputs

$\mathbf{Qout}_{\mathbf{ASPX\_PP,[0|1|2|...]}}$ *num_sig* complex-valued QMF matrices.

The $\mathbf{Qout}_{\mathrm{ASPX}}$ matrices each consist of *num_qmf_timeslots* columns and *num_qmf_subbands* rows.

5.4.3 Processing

The decoder shall process the input signals:

$$\mathbf{Qin}_{\mathrm{ASPX\_PP,[0|1|2|...]}} = Q\_in\_ASPX\_PP[i]\text{, for i=0,1,2,...}$$

to calculate the output signals:

$$\mathbf{Qout}_{\mathrm{ASPX\_PP,[0|1|2|...]}} = Q\_out\_ASPX\_PP[i]\text{, for i=0,1,2,...}$$

as specified in pseudocode 1.

Pseudocode 1: A-SPX post processing

```
for (i=0; i<num_sig; i++)
{
  for(ts=0; ts<num_qmf_timeslots; ts++)
  {
    for(sb=sbx; sb<num_qmf_subbands; sb++)
    {
      Q_out_ASPX_PP[i][ts][sb] = 0.841395 * Q_in_ASPX_PP[i][ts][sb]; // -1.5 dB
    }
  }
}
```

NOTE: *sbx* is specified in ETSI TS 103 190-1 [1], clause 5.7.6.

5.5 Advanced coupling (A-CPL) for immersive audio

5.5.1 Introduction

The specification of the A-CPL tool in this section extends the specification of the A-CPL tool in ETSI TS 103 190-1 [1], clause 5.7.7, to support A-CPL for channel-based immersive audio.

5.5.2 Processing the immersive_channel_element

When decoding an `immersive_channel_element` in full decoding mode and *immersive_codec_mode* ∈ {ASPX_ACPL_1, ASPX_ACPL_2}, the decoder shall utilize the A-CPL tool specified in this clause. In this case, the *core_channel_config* is 7CH_STATIC and either four or six parallel A-CPL modules are utilized. If *b_5fronts* is true, thirteen input channels are present and are processed by six A-CPL modules. If *b_5fronts* is false, eleven input channels are present and are processed by four A-CPL modules. The mapping of the channels to A-CPL input (x0 … x12) and output variables (z0 … z12) is specified in table 25.

Table 25: Mapping of channels with A-CPL input/output variables for immersive_channel_element and b_5fronts

A-CPL Input / Output variable	Channel
x0 / z0	L
x1 / z2	R
x2 / z4	C
x3 / z1	Lscr (see note)
x4 / z3	Rscr (see note)
x5 / z5	Ls
x6 / z7	Rs
x7 / z6	Lb
x8 / z8	Rb
x9 / z9	Tfl
x10 / z11	Tfr
x11 / z10	Tbl
x12 / z12	Tbr
NOTE: Only applicable if *b_5fronts* is True	

Pseudocode 2 describes how the decoder shall calculate the output signals.

Pseudocode 2: Calculation of A-CPL output signals for immersive audio

```
x5in  = 2*x5;
x6in  = 2*x6;
x9in  = 2*x9;
x10in = 2*x10;
u0 = inputSignalModification(x5in); // use decorrelator D0
u1 = inputSignalModification(x6in); // use decorrelator D0
u2 = inputSignalModification(x9in); // use decorrelator D1
u3 = inputSignalModification(x10in); // use decorrelator D1
y0 = applyTransientDucker(u0);
y1 = applyTransientDucker(u1);
y2 = applyTransientDucker(u2);
y3 = applyTransientDucker(u3);
if (codec_mode == ASPX_ACPL_1) {
  x7in  = 2*x7;
  x8in  = 2*x8;
  x11in  = 2*x11;
  x12in  = 2*x12;
  (z5,  z6)  = ACplModule(acpl_alpha_1_dq, acpl_beta_1_dq, num_pset_1, x5in, x7in, y0);
  (z7,  z8)  = ACplModule(acpl_alpha_2_dq, acpl_beta_2_dq, num_pset_2, x6in, x8in, y1);
  (z9,  z10) = ACplModule(acpl_alpha_3_dq, acpl_beta_3_dq, num_pset_3, x9in, x11in, y2);
  (z11, z12) = ACplModule(acpl_alpha_4_dq, acpl_beta_4_dq, num_pset_4, x10in, x12in, y3);
  if (b_5fronts) {
    u4 = inputSignalModification(x0in); // use decorrelator D2
    u5 = inputSignalModification(x1in); // use decorrelator D2
    y4 = applyTransientDucker(u4);
    y5 = applyTransientDucker(u5);
    x0in = 2*x0;
    x1in = 2*x1;
    x3in = 2*x3;
    x4in = 2*x4;
    (z0, z1) = ACplModule(acpl_alpha_5_dq, acpl_beta_5_dq, num_pset_5, x0in, x3in, y4);
    (z2, z3) = ACplModule(acpl_alpha_6_dq, acpl_beta_6_dq, num_pset_6, x1in, x4in, y5);
  }
  else {
    z0 = 2*x0;
    z2 = 2*x1;
  }
}
else if (codec_mode == ASPX_ACPL_2) {
  (z5,  z6)  = ACplModule(acpl_alpha_1_dq, acpl_beta_1_dq, num_pset_1, x5in, 0, y0);
  (z7,  z8)  = ACplModule(acpl_alpha_2_dq, acpl_beta_2_dq, num_pset_2, x6in, 0, y1);
  (z9,  z10) = ACplModule(acpl_alpha_3_dq, acpl_beta_3_dq, num_pset_3, x9in, 0, y2);
  (z11, z12) = ACplModule(acpl_alpha_4_dq, acpl_beta_4_dq, num_pset_4, x10in, 0, y3);
  if (b_5fronts) {
    u4 = inputSignalModification(x0in); // use decorrelator D2
    u5 = inputSignalModification(x1in); // use decorrelator D2
    y4 = applyTransientDucker(u4);
    y5 = applyTransientDucker(u5);
    x0in = 2*x0;
    x1in = 2*x1;
    (z0, z1) = ACplModule(acpl_alpha_5_dq, acpl_beta_5_dq, num_pset_5, x0in, 0, y4);
    (z2, z3) = ACplModule(acpl_alpha_6_dq, acpl_beta_6_dq, num_pset_6, x1in, 0, y5);
  }
  else {
    z0 = 2*x0;
    z2 = 2*x1;
  }
}
z4 = 2*x2;
z5  *= sqrt(2);
z6  *= sqrt(2);
z7  *= sqrt(2);
z8  *= sqrt(2);
z9  *= sqrt(2);
z10 *= sqrt(2);
z11 *= sqrt(2);
z12 *= sqrt(2);
```

The functions *inputSignalModification()* and *applyTransientDucker()* are defined in ETSI TS 103 190-1 [1], clauses 5.7.7.4.2 and 5.7.7.4.3, respectively, and *ACplModule()* is defined in ETSI TS 103 190-1 [1], clause 5.7.7.5.

The variables *num_pset_1* to *num_pset_6* indicate the value *acpl_num_param_sets* of the corresponding `acpl_data_1ch()` element.

The arrays *acpl_alpha_1_dq* and *acpl_beta_1_dq* are the dequantized values of `acpl_alpha1` and `acpl_beta1` of the first `acpl_data_1ch()` element and all analogue variables with higher numbering should be calculated the same way using the corresponding `acpl_data_1ch()` element.

The dequantization is performed as described in ETSI TS 103 190-1 [1], clause 5.7.7.7.

5.6 Advanced Joint Channel Coding (A-JCC)

5.6.1 Introduction

The Advanced Joint Channel Coding (A-JCC) tool improves coding of multiple audio channels. The coding efficiency is achieved by representing the multichannel audio using a five-channel audio signal and parametric side information. The A-JCC tool supports the full decoding mode as specified in clause 5.6.3.5.2 and the core decoding mode as specified in clause 5.6.3.5.3.

5.6.2 Interface

5.6.2.1 Inputs

Qin $_{\mathbf{AJCC,[A|B|C|D|E]}}$ Five complex valued QMF matrices of five input audio channels to be processed by the A-JCC tool.

The **Qin** $_{\mathrm{AJCC}}$ matrices each consist of *num_qmf_timeslots* columns and *num_qmf_subbands* rows.

5.6.2.2 Outputs

Qout $_{\mathbf{AJCC,[L|R|C|...]}}$ *ajcc_num_out* complex-valued QMF matrices corresponding to the number of reconstructed audio channels.

The **Qout** $_{\mathrm{AJCC}}$ matrices each consist of *num_qmf_timeslots* columns and *num_qmf_subbands* rows. *ajcc_num_out* denotes the number of reconstructed output channels and depends on the decoding mode and *b_5fronts* as specified in table 26.

Table 26: ajcc_num_out

Decoding mode	*b_5fronts*	*ajcc_num_out*
Full decoding	False	11
	True	13
Core decoding	X	7

5.6.2.3 Controls

The control information for the A-JCC tool consists of decoded and dequantized A-JCC side information. The side information contains parameters to control the dequantization process described in clause 5.6.3.2, the interpolation process described in clause 5.6.3.3, the decorrelation process described in clause 5.6.3.4, and parameters which are used in the reconstruction process described in clause 5.6.3.5. The parameter band to QMF subband mapping is explained in clause 5.6.3.1.

5.6.3 Processing

5.6.3.1 Parameter band to QMF subband mapping

The A-JCC parameters are transmitted per parameter band. Like in the A-CPL tool, the parameter bands are groupings of QMF subbands and they have lower frequency resolution than the QMF subbands. The mapping of parameter bands to QMF subbands for the A-JCC tool is the same as the mapping for the A-CPL tool as specified in ETSI TS 103 190-1 [1], table 197. The number of parameter bands: 7, 9, 12, or 15 is indicated via the bitstream element *ajcc_num_param_bands_id*.

5.6.3.2 Differential decoding and dequantization

Differential decoding as described in the pseudocode shown in pseudocode 3 shall be performed to get the quantized values *ajcc_**SET**_q* from the Huffman decoded values *ajcc_**SET***. **SET** is an identifier for the A-JCC parameter set type, and is one of: dry1f, dry2f, dry3f, dry4f, dry1b, dry2b, dry3b, dry4b, wet1f, wet2f, wet3f, wet4f, wet5f, wet6f, wet1b, wet2b, wet3b, wet4b, wet5b, wet6b, alpha1, alpha2, beta1, beta2, dry1, dry2, dry3, dry4, wet1, wet2, wet3, wet4, wet5, wet6.

Pseudocode 3: Differential decoding for A-JCC

```
// differential decoding for A-JCC
// input: array ajcc_SET (SET in {dry1f, dry2f, ..., wet6})
// vector ajcc_SET_q_prev
// output: array ajcc_SET_q
num_ps = num_ps[SET];  // number of ajcc parameter sets for SET
                       // = ajcc_num_param_sets_code + 1 for SET
for (ps = 0; ps < num_ps; ps++) {
  if (diff_type[ps] == 0) {    // DIFF_FREQ
    ajcc_SET_q[ps][0] = ajcc_SET[ps][0];
    for (i = 1; i < num_bands; i++) {
      ajcc_SET_q[ps][i] = ajcc_SET_q[ps][i-1] + ajcc_SET[ps][i];
    }
  }
  else {                       // DIFF_TIME
    for (i = 0; i < num_bands; i++) {
      ajcc_SET_q[ps][i] = ajcc_SET_q_prev[i] + ajcc_SET[ps][i];
    }
  }
  ajcc_SET_q_prev = ajcc_SET_q[ps];
}
```

The quantized values from the last corresponding parameter set of the previous AC-4 frame, *ajcc_**SET**_q_prev*, are needed when delta coding in the time direction over AC-4 frame boundaries.

The dequantized values *ajcc_alpha1_dq*, *ajcc_alpha2_dq*, *ajcc_beta1_dq*, and *ajcc_beta2_dq* are obtained from *ajcc_alpha1_q*, *ajcc_alpha2_q*, *ajcc_beta1_q*, and *ajcc_beta2_q* using ETSI TS 103 190-1 [1], table 203 and ETSI TS 103 190-1 [1], table 204 if the quantization mode is set to fine(`ajcc_qm_ab` = 0), and using ETSI TS 103 190-1 [1], table 205 and ETSI TS 103 190-1 [1], table 206 if the quantization mode is set to coarse (`ajcc_qm_ab` = 1).

For each parameter, an index *ibeta* is obtained from ETSI TS 103 190-1 [1], table 203 or ETSI TS 103 190-1 [1], table 205 during the dequantization of the alpha values. This value is used in ETSI TS 103 190-1 [1], table 204 or ETSI TS 103 190-1 [1], table 206, for fine and coarse quantization modes respectively, to calculate the corresponding dequantized beta value.

The dequantized values *ajcc_dry**X**_dq* are obtained from the *ajcc_dry**X**_q* values by multiplying the entries of *ajcc_dry**X**_q* by the delta factor corresponding to the signalled quantization mode and by subtracting a value of 0,6. This operation is shown in pseudocode 4.

Pseudocode 4: Dequantization of A-JCC dry values

```
// dequantization of A-JCC dry values

if (quant_mode == 0)    // fine quantization
  delta_dry = 0.1;
else
  delta_dry = 0.2;
```

```
for (i = 0; i < num_bands; i++)
{
  ajcc_dryX_dq[i] = ajcc_dryX_q[i] * delta_dry - 0.6;
}
```

The dequantized values *ajcc_wet**X**_dq* are obtained from the *ajcc_wet**X**_q* values by multiplying the entries of *ajcc_wet**X**_q* by the delta factor corresponding to the signalled quantization mode and by subtracting a value of 2,0. This operation is shown in pseudocode 5.

Pseudocode 5: Dequantization of A-JCC wet values

```
// dequantization of A-JCC wet values

if (quant_mode == 0)    // fine quantization
  delta_wet = 0.1;
else
  delta_wet = 0.2;

for (i = 0; i < num_bands; i++)
{
  ajcc_wetX_dq[i] = ajcc_wetX_q[i] * delta_wet - 2.0;
}
```

5.6.3.3 Interpolation

Parameter sets are transmitted either once or twice per frame, determined by the variable *ajcc_num_param_sets*.

Decoded and dequantized A-JCC parameters carried in the bitstream are time-interpolated to calculate values that are applied to the input of the decorrelator and to the ducked output of the decorrelator. Two forms of interpolation, smooth and steep, are utilized to interpolate values for each QMF subsample:

- When smooth interpolation is used, the values for each QMF subsample between consecutive parameter sets are linearly interpolated.
- When steep interpolation is used, the values for each QMF subsamples are switched over instantaneously at the QMF time slot indicated by `ajcc_param_timeslot`.

The *interpolate_ajcc()* function, used in clause 5.6.3.5, is described by the pseudocode shown in pseudocode 6.

Pseudocode 6: interpolate_ajcc()

```
interpolate_ajcc(ajcc_param, num_pset, sb, ts)
{
  num_ts = num_qmf_timeslots;

  if (ajcc_interpolation_type == 0) {  // smooth interpolation
    if (num_pset == 1) {  // 1 parameter set
      delta  = ajcc_param[0][sb_to_pb(sb)] - ajcc_param_prev[sb];
      interp = ajcc_param_prev[sb] + (ts+1)*delta/num_ts;
    }
    else {                // 2 parameter sets
      ts_2 = floor(num_ts/2);
      if (ts < ts_2) {
        delta  = ajcc_param[0][sb_to_pb(sb)] - ajcc_param_prev[sb];
        interp = ajcc_param_prev[sb] + (ts+1)*delta/ts_2;
      }
      else {
        delta  = ajcc_param[1][sb_to_pb(sb)] - ajcc_param[0][sb_to_pb(sb)];
        interp = ajcc_param[0][sb_to_pb(sb)] + (ts-ts_2+1)*delta/(num_ts-ts_2);
      }
    }
  }
  else {  // steep interpolation
    if (num_pset == 1) {  // 1 parameter set
      if (ts < ajcc_param_timeslot[0]) {
        interp = ajcc_param_prev[sb];
      }
      else {
        interp = ajcc_param[0][sb_to_pb(sb)];
      }
    }
```

```
    else {                // 2 parameter sets
      if (ts < ajcc_param_timeslot[0]) {
        interp = ajcc_param_prev[sb];
      }
      else if (ts < ajcc_param_timeslot[1]) {
        interp = ajcc_param[0][sb_to_pb(sb)];
      }
      else {
        interp = ajcc_param[1][sb_to_pb(sb)];
      }
    }
  }
  return interp;
}
```

The *sb_to_pb()* function maps from QMF subbands to parameter bands according to ETSI TS 103 190-1 [1], table 197. The array *ajcc_param_prev[sb]*, which holds the dequantized A-JCC parameters from the previous AC-4 frame related to the provided *ajcc_param[pset][pb]* array, is also passed on to the *interpolate_ajcc()* function although *ajcc_param_prev* is not shown as input parameter.

The pseudocode shown in pseudocode 7 describes the initialization of *ajcc_param_prev[sb]* for all relevant dequantized A-JCC parameter arrays for the next AC-4 frame at the end of the A-JCC tool processing.

Pseudocode 7: ajcc_param_prev()

```
for (sb = 0; sb < num_qmf_subbands; sb++) {
  ajcc_param_prev[sb] = ajcc_param[num_pset-1][sb_to_pb(sb)];
}
```

When decoding the first AC-4 frame, all elements of all *ajcc_param_prev[sb]* arrays shall be 0.

5.6.3.4 Decorrelator and transient ducker

The A-JCC processing includes multiple decorrelation processes where *ajcc_num_decorr* parallel decorrelator instances generate the output signals:

Qdecorr_out$_{\mathbf{AJCC,[a|b|...]}}$ *ajcc_num_decorr* complex-valued QMF matrices; each one is the output of one of the parallel decorrelator instances.

The output signals are calculated from the decorrelation input signals:

Qdecorr_in$_{\mathbf{AJCC,[a|b|...]}}$ *ajcc_num_decorr* complex-valued QMF matrices; each one is the input to one of the parallel decorrelator instances.

The **Qdecorr_in**$_{\mathrm{AJCC}}$ and **Qdecorr_out**$_{\mathrm{AJCC}}$ matrices each consist of *num_qmf_timeslots* columns and *num_qmf_subbands* rows. The number *ajcc_num_decorr* of parallel decorrelator instances depends on the decoding mode and for the full decoding mode on *b_5fronts* as shown in table 27.

Table 27: ajcc_num_decorr

Decoding mode	*b_5fronts*	*ajcc_num_decorr*
Full decoding	False	6
	True	8
Core decoding	X	4

The decorrelators used in A-JCC processing are identical to the decorrelators of the advanced coupling tool (D0, D1 and D2). The coefficients of the three used decorrelators are described in ETSI TS 103 190-1 [1], clause 5.7.7.4.2. The frequency subbands are grouped as described in ETSI TS 103 190-1 [1], clause 5.7.7.4.1. As the maximum number of active decorrelator instances is given by *ajcc_num_decorr = 8*, the three available decorrelators D0, D1, and D2 are assigned as described in the comments of each respective *inputSignalModification()* call inside the pseudocode in clause 5.6.3.5.2 and clause 5.6.3.5.3. The output signals of these decorrelator instances are also processed by the transient ducker algorithm as described in ETSI TS 103 190-1 [1], clause 5.7.7.4.3.

5.6.3.5 Reconstruction of the output channels

5.6.3.5.1 Input channels

For the A-JCC reconstruction processing five input channels are present. Their elements are addressed as:

First input channel $x_0(ts,sb) \in \mathbf{Qin}_{\mathrm{AJCC,A}}$

Second input channel $x_1(ts,sb) \in \mathbf{Qin}_{\mathrm{AJCC,B}}$

Third input channel $x_2(ts,sb) \in \mathbf{Qin}_{\mathrm{AJCC,C}}$

Fourth input channel $x_3(ts,sb) \in \mathbf{Qin}_{\mathrm{AJCC,D}}$

Fifth input channel $x_4(ts,sb) \in \mathbf{Qin}_{\mathrm{AJCC,E}}$

5.6.3.5.2 A-JCC full decoding mode

The reconstructed output channels in full decoding mode are addressed as:

Left output channel $z_0(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,}}\mathrm{L}$

Right output channel $z_1(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,}}\mathrm{R}$

Centre output channel $z_2(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,}}\mathrm{C}$

Left Screen output channel $z_3(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,}}\mathrm{Lscr}$

Right Screen output channel $z_4(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,}}\mathrm{Rscr}$

Left Side/Surround output channel $z_5(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,}}\mathrm{Ls}$

Right Side/Surround output channel $z_6(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,}}\mathrm{Rs}$

Left Back output channel $z_7(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,}}\mathrm{Lb}$

Right Back output channel $z_8(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,}}\mathrm{Rb}$

Top Front Left output channel $z_9(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,}}\mathrm{Tfl}$

Top Front Right output channel $z_{10}(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,}}\mathrm{Tfr}$

Top Back Left output channel $z_{11}(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,}}\mathrm{Tbl}$

Top Back Right output channel $z_{12}(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,}}\mathrm{Tbr}$

The outputs z_3 and z_4 are only present if *b_5fronts* is true.

The outputs are derived according to the pseudocode in pseudocode 8.

Pseudocode 8: A-JCC output in full decoding mode

```
x0in = (2+1/sqrt(2))*x0;
x1in = (2+1/sqrt(2))*x1;
x2in = (2+1/sqrt(2))*x2;
x3in = (2+1/sqrt(2))*x3;
x4in = (2+1/sqrt(2))*x4;

if (b_5fronts) {
  u0 = inputSignalModification(x0in); // D0
  u1 = inputSignalModification(x0in); // D2
  u2 = inputSignalModification(x1in); // D0
  u3 = inputSignalModification(x1in); // D2
  u4 = inputSignalModification(x3in); // D1
  u5 = inputSignalModification(x3in); // D2
  u6 = inputSignalModification(x4in); // D1
  u7 = inputSignalModification(x4in); // D2

  num_pset_1 = ajcc_nps_lf;
  num_pset_2 = ajcc_nps_rf;
  num_pset_3 = ajcc_nps_lb;
  num_pset_4 = ajcc_nps_rb;

  y6 = applyTransientDucker(u6);
  y7 = applyTransientDucker(u7);
}
else {
  (w1in, w2in) = input_sig_pre_modification(x0in, x3in, x1in, x4in);

  u0 = inputSignalModification(x0in); // D0
  u1 = inputSignalModification(w1in); // D2
  u2 = inputSignalModification(x3in); // D1
  u3 = inputSignalModification(x1in); // D0
  u4 = inputSignalModification(w2in); // D2
  u5 = inputSignalModification(x4in); // D1

  num_pset_1 = ajcc_nps_l;
  num_pset_2 = ajcc_nps_r;
}

y0 = applyTransientDucker(u0);
y1 = applyTransientDucker(u1);
y2 = applyTransientDucker(u2);
y3 = applyTransientDucker(u3);
y4 = applyTransientDucker(u4);
y5 = applyTransientDucker(u5);

if (b_5fronts) {
  (z0, z9, z3) = ajcc_module_1(ajcc_dry1f_dq, ajcc_dry2f_dq,
                               ajcc_wet1f_dq, ajcc_wet2f_dq, ajcc_wet3f_dq,
                               num_pset_1, x0in, y0, y1);
  (z1, z10, z4) = ajcc_module_1(ajcc_dry3f_dq, ajcc_dry4f_dq,
                                ajcc_wet4f_dq, ajcc_wet5f_dq, ajcc_wet6f_dq,
                                num_pset_2, x1in, y2, y3);
  (z5, z7, z11) = ajcc_module_1(ajcc_dry1b_dq, ajcc_dry2b_dq,
                                ajcc_wet1b_dq, ajcc_wet2b_dq, ajcc_wet3b_dq,
                                num_pset_3, x3in, y4, y5);
  (z6, z8, z12) = ajcc_module_1(ajcc_dry3b_dq, ajcc_dry4b_dq,
                                ajcc_wet4b_dq, ajcc_wet5b_dq, ajcc_wet6b_dq,
                                num_pset_4, x4in, y6, y7);
}
else {
  (z0, z5, z7, z9, z11) = ajcc_module_2(ajcc_alpha1_dq, ajcc_beta1_dq,
                                        ajcc_dry1_dq, ajcc_dry2_dq,
                                        ajcc_wet1_dq, ajcc_wet2_dq, ajcc_wet3_dq,
                                        num_pset_1, x0in, x3in, y0, y1, y2);
  (z1, z6, z8, z10, z12) = ajcc_module_2(ajcc_alpha2_dq, ajcc_beta2_dq,
                                         ajcc_dry3_dq, ajcc_dry4_dq,
                                         ajcc_wet4_dq, ajcc_wet5_dq, ajcc_wet6_dq,
                                         num_pset_2, x1in, x4in, y3, y4, y5);
}

z2 = x2in;
z5 *= sqrt(2);
z6 *= sqrt(2);
```

```
z7 *= sqrt(2);
z8 *= sqrt(2);
z9 *= sqrt(2);
z10 *= sqrt(2);
z11 *= sqrt(2);
z12 *= sqrt(2);
```

Functions *inputSignalModification()* and *applyTransientDucker()* are specified in ETSI TS 103 190-1 [1], clauses 5.7.7.4.2 and 5.7.7.4.3, respectively.

The pseudocode in pseudocode 9 shows the *input_sig_pre_modification()* function.

Pseudocode 9: input_sig_pre_modification()

```
input_sig_pre_modification(in1, in2, in3, in4)
{
  g = 0;
  d = 0;
  if (ajcc_core_mode == ajcc_core_mode_prev) {
    if (ajcc_core_mode == 0) {
      g = 1;
    }
  }
  else {
    if (ajcc_core_mode == 0) {
      d = 1/num_qmf_timeslots;
    }
    else {
      g = 1;
      d = -1/num_qmf_timeslots;
    }
    ajcc_core_mode_prev = ajcc_core_mode;
  }
  for (ts = 0; ts < num_qmf_timeslots; ts++) {
    g += d;
    for (sb = 0; sb < num_qmf_subbands; sb++) {
      out1[ts][sb] = g*in2[ts][sb] + (1-g)*in1[ts][sb];
      out2[ts][sb] = g*in4[ts][sb] + (1-g)*in3[ts][sb];
    }
  }
  return (out1, out2);
}
```

The helper variable *ajcc_core_mode_prev* shall be initialized to `ajcc_core_mode`.

The pseudocode in pseudocode 10 shows the *ajcc_module_1()* function.

Pseudocode 10: ajcc_module_1()

```
ajcc_module_1(ajcc_dry1, ajcc_dry2,
              ajcc_wet1, ajcc_wet2, ajcc_wet3,
              num_pset, x, y0, y1)
{
  for (ps = 0; ps < num_pset; ps++) {
    for (pb = 0; pb < ajcc_num_bands_table[ajcc_num_param_bands_id]; pb++) {
      ajcc_dry3[ps][pb] = 1 - ajcc_dry1[ps][pb] - ajcc_dry2[ps][pb];
      p0[ps][pb] = 1/sqrt(2) * (ajcc_wet1[ps][pb] + ajcc_wet3[ps][pb]);
      p1[ps][pb] = 1/sqrt(2) * (ajcc_wet3[ps][pb] + ajcc_wet2[ps][pb]);
      p2[ps][pb] = -1/sqrt(2) * ajcc_wet3[ps][pb];
      p3[ps][pb] = -1/sqrt(2) * ajcc_wet2[ps][pb];
      p4[ps][pb] = -1/sqrt(2) * ajcc_wet1[ps][pb];
      p5[ps][pb] = -1/sqrt(2) * ajcc_wet3[ps][pb];
    }
  }
  for (sb = 0; sb < num_qmf_subbands; sb++) {
    for (ts = 0; ts < num_qmf_timeslots; ts++) {
      interp_d0 = interpolate_ajcc(ajcc_dry1, num_pset, sb, ts);
      interp_d1 = interpolate_ajcc(ajcc_dry2, num_pset, sb, ts);
      interp_d2 = interpolate_ajcc(ajcc_dry3, num_pset, sb, ts);
      interp_p0 = interpolate_ajcc(p0, num_pset, sb, ts);
      interp_p1 = interpolate_ajcc(p1, num_pset, sb, ts);
      interp_p2 = interpolate_ajcc(p2, num_pset, sb, ts);
      interp_p3 = interpolate_ajcc(p3, num_pset, sb, ts);
      interp_p4 = interpolate_ajcc(p4, num_pset, sb, ts);
      interp_p5 = interpolate_ajcc(p5, num_pset, sb, ts);

      z0[ts][sb] = interp_d0*x[ts][sb] + interp_p0*y0[ts][sb] + interp_p1*y1[ts][sb];
      z1[ts][sb] = interp_d1*x[ts][sb] + interp_p2*y0[ts][sb] + interp_p3*y1[ts][sb];
      z2[ts][sb] = interp_d2*x[ts][sb] + interp_p4*y0[ts][sb] + interp_p5*y1[ts][sb];
    }
  }
  return (z0, z1, z2);
}
```

The pseudocode in pseudocode 11 shows the *ajcc_module_2()* function.

Pseudocode 11: ajcc_module_2()

```
ajcc_module_2(ajcc_alpha, ajcc_beta,
              ajcc_dry1, ajcc_dry2,
              ajcc_wet1, ajcc_wet2, ajcc_wet3,
              num_pset, x0, x1, y0, y1, y2)
{
  if (ajcc_core_mode == 0) {
    for (ps = 0; ps < num_pset; ps++) {
      for (pb = 0; pb < ajcc_num_bands_table[ajcc_num_param_bands_id]; pb++) {
        d0[ps][pb] = (1 + ajcc_alpha[ps][pb])/2;
        d1[ps][pb] = 0;
        d2[ps][pb] = 0;
        d3[ps][pb] = (1 - ajcc_alpha[ps][pb])/2;
        d4[ps][pb] = 0;
        d5[ps][pb] = 0;
        d6[ps][pb] = ajcc_dry1[ps][pb];
        d7[ps][pb] = ajcc_dry2[ps][pb];
        d8[ps][pb] = 0;
        d9[ps][pb] = 1-ajcc_dry1[ps][pb]-ajcc_dry2[ps][pb];
        w0[ps][pb] = ajcc_beta[ps][pb]/2;
        w1[ps][pb] = 0;
        w2[ps][pb] = 0;
        w3[ps][pb] = -1*ajcc_beta[ps][pb]/2;
        w4[ps][pb] = 0;
        w5[ps][pb] = 0;
        w6[ps][pb] = (ajcc_wet1[ps][pb]+ajcc_wet3[ps][pb])/sqrt(2);
        w7[ps][pb] = -1*ajcc_wet3[ps][pb]/sqrt(2);
        w8[ps][pb] = 0;
        w9[ps][pb] = -1*ajcc_wet1[ps][pb]/sqrt(2);
        w10[ps][pb] = 0;
        w11[ps][pb] = (ajcc_wet3[ps][pb]+ajcc_wet2[ps][pb])/sqrt(2);
        w12[ps][pb] = -1*ajcc_wet2[ps][pb]/sqrt(2);
        w13[ps][pb] = 0;
        w14[ps][pb] = -1*ajcc_wet3[ps][pb]/sqrt(2);
```

```
      }
    }
  }
  else {
    for (ps = 0; ps < num_pset; ps++) {
      for (pb = 0; pb < ajcc_num_bands_table[ajcc_num_param_bands_id]; pb++) {
        d0[ps][pb] = ajcc_dry1[ps][pb];
        d1[ps][pb] = ajcc_dry2[ps][pb];
        d2[ps][pb] = 1-ajcc_dry1[ps][pb]-ajcc_dry2[ps][pb];
        d3[ps][pb] = 0;
        d4[ps][pb] = 0;
        d5[ps][pb] = 0;
        d6[ps][pb] = 0;
        d7[ps][pb] = 0;
        d8[ps][pb] = (1 + ajcc_alpha[ps][pb])/2;
        d9[ps][pb] = (1 - ajcc_alpha[ps][pb])/2;
        w0[ps][pb] = (ajcc_wet1[ps][pb]+ajcc_wet3[ps][pb])/sqrt(2);
        w1[ps][pb] = -1*ajcc_wet3[ps][pb]/sqrt(2);
        w2[ps][pb] = -1*ajcc_wet1[ps][pb]/sqrt(2);
        w3[ps][pb] = 0;
        w4[ps][pb] = 0;
        w5[ps][pb] = (ajcc_wet3[ps][pb]+ajcc_wet2[ps][pb])/sqrt(2);
        w6[ps][pb] = -1*ajcc_wet2[ps][pb]/sqrt(2);
        w7[ps][pb] = -1*ajcc_wet3[ps][pb]/sqrt(2);
        w8[ps][pb] = 0;
        w9[ps][pb] = 0;
        w10[ps][pb] = 0;
        w11[ps][pb] = 0;
        w12[ps][pb] = 0;
        w13[ps][pb] = ajcc_beta[ps][pb]/2;
        w14[ps][pb] = -1*ajcc_beta[ps][pb]/2;
      }
    }
  }
  for (sb = 0; sb < num_qmf_subbands; sb++) {
    for (ts = 0; ts < num_qmf_timeslots; ts++) {
      interp_d0 = interpolate_ajcc(d0, num_pset, sb, ts);
      interp_d1 = interpolate_ajcc(d1, num_pset, sb, ts);
      interp_d2 = interpolate_ajcc(d2, num_pset, sb, ts);
      interp_d3 = interpolate_ajcc(d3, num_pset, sb, ts);
      interp_d4 = interpolate_ajcc(d4, num_pset, sb, ts);
      interp_d5 = interpolate_ajcc(d5, num_pset, sb, ts);
      interp_d6 = interpolate_ajcc(d6, num_pset, sb, ts);
      interp_d7 = interpolate_ajcc(d7, num_pset, sb, ts);
      interp_d8 = interpolate_ajcc(d8, num_pset, sb, ts);
      interp_d9 = interpolate_ajcc(d9, num_pset, sb, ts);
      interp_w0 = interpolate_ajcc(w0, num_pset, sb, ts);
      interp_w1 = interpolate_ajcc(w1, num_pset, sb, ts);
      interp_w2 = interpolate_ajcc(w2, num_pset, sb, ts);
      interp_w3 = interpolate_ajcc(w3, num_pset, sb, ts);
      interp_w4 = interpolate_ajcc(w4, num_pset, sb, ts);
      interp_w5 = interpolate_ajcc(w5, num_pset, sb, ts);
      interp_w6 = interpolate_ajcc(w6, num_pset, sb, ts);
      interp_w7 = interpolate_ajcc(w7, num_pset, sb, ts);
      interp_w8 = interpolate_ajcc(w8, num_pset, sb, ts);
      interp_w9 = interpolate_ajcc(w9, num_pset, sb, ts);
      interp_w10 = interpolate_ajcc(w10, num_pset, sb, ts);
      interp_w11 = interpolate_ajcc(w11, num_pset, sb, ts);
      interp_w12 = interpolate_ajcc(w12, num_pset, sb, ts);
      interp_w13 = interpolate_ajcc(w13, num_pset, sb, ts);
      interp_w14 = interpolate_ajcc(w14, num_pset, sb, ts);

      z0[ts][sb] = interp_d0*x0[ts][sb] + interp_d5*x1[ts][sb] + interp_w0*y0[ts][sb] + interp_w5*y1
[ts][sb] + interp_w10*y2[ts][sb];
      z1[ts][sb] = interp_d1*x0[ts][sb] + interp_d6*x1[ts][sb] + interp_w1*y0[ts][sb] + interp_w6*y1
[ts][sb] + interp_w11*y2[ts][sb];
      z2[ts][sb] = interp_d2*x0[ts][sb] + interp_d7*x1[ts][sb] + interp_w2*y0[ts][sb] + interp_w7*y1
[ts][sb] + interp_w12*y2[ts][sb];
      z3[ts][sb] = interp_d3*x0[ts][sb] + interp_d8*x1[ts][sb] + interp_w3*y0[ts][sb] + interp_w8*y1
[ts][sb] + interp_w13*y2[ts][sb];
      z4[ts][sb] = interp_d4*x0[ts][sb] + interp_d9*x1[ts][sb] + interp_w4*y0[ts][sb] + interp_w9*y1
[ts][sb] + interp_w14*y2[ts][sb];
    }
  }
  return (z0, z1, z2, z3, z4);
}
```

5.6.3.5.3 A-JCC core decoding mode

The reconstructed output channels in core decoding mode are addressed as:

Left output channel	$z_0(ts,sb) \in \mathbf{Qout}_{\text{AJCC,}}\text{L}$
Right output channel	$z_1(ts,sb) \in \mathbf{Qout}_{\text{AJCC,}}\text{R}$
Centre output channel	$z_2(ts,sb) \in \mathbf{Qout}_{\text{AJCC,}}\text{C}$
Left Side/Surround output channel	$z_3(ts,sb) \in \mathbf{Qout}_{\text{AJCC,}}\text{Ls}$
Right Side/Surround output channel	$z_4(ts,sb) \in \mathbf{Qout}_{\text{AJCC,}}\text{Rs}$
Top Side Left output channel	$z_5(ts,sb) \in \mathbf{Qout}_{\text{AJCC,}}\text{Tsl}$
Top Side Right output channel	$z_6(ts,sb) \in \mathbf{Qout}_{\text{AJCC,}}\text{Tsr}$

The output signals shall be derived according to the pseudocode in pseudocode 12.

Pseudocode 12: A-JCC output in core decoding mode

```
x0in = (2+1/sqrt(2))*x0;
x1in = (2+1/sqrt(2))*x1;
x2in = (2+1/sqrt(2))*x2;
x3in = (2+1/sqrt(2))*x3;
x4in = (2+1/sqrt(2))*x4;

u0 = inputSignalModification(x0in); // D0
u1 = inputSignalModification(x3in); // D2
u2 = inputSignalModification(x1in); // D0
u3 = inputSignalModification(x4in); // D2

y0 = applyTransientDucker(u0);
y1 = applyTransientDucker(u1);
y2 = applyTransientDucker(u2);
y3 = applyTransientDucker(u3);

if (b_5fronts) {
  num_pset_1 = ajcc_nps_lf;
  num_pset_2 = ajcc_nps_rf;
  num_pset_3 = ajcc_nps_lb;
  num_pset_4 = ajcc_nps_rb;

  (z0, z3, z5) = ajcc_module_3(ajcc_dry1f_dq, ajcc_dry2f_dq,
                 ajcc_wet1f_dq, ajcc_wet2f_dq, ajcc_wet3f_dq,
                 ajcc_dry1b_dq, ajcc_dry2b_dq,
                 ajcc_wet1b_dq, ajcc_wet2b_dq, ajcc_wet3b_dq,
                 num_pset_1, num_pset_3, x0in, x3in, y0, y1);

  (z1, z4, z6) = ajcc_module_3(ajcc_dry3f_dq, ajcc_dry4f_dq,
                 ajcc_wet4f_dq, ajcc_wet5f_dq, ajcc_wet6f_dq,
                 ajcc_dry3b_dq, ajcc_dry4b_dq,
                 ajcc_wet4b_dq, ajcc_wet5b_dq, ajcc_wet6b_dq,
                 num_pset_2, num_pset_4, x1in, x4in, y2, y3);
}
else {
  num_pset_1 = ajcc_nps_l;
  num_pset_2 = ajcc_nps_r;

  (z0, z3, z5) = ajcc_module_4(ajcc_alpha1_dq, ajcc_beta1_dq,
                 ajcc_dry1_dq, ajcc_dry2_dq,
                 ajcc_wet1_dq, ajcc_wet2_dq, ajcc_wet3_dq,
                 num_pset_1, x0in, x3in, y0, y1);

  (z1, z4, z6) = ajcc_module_4(ajcc_alpha2_dq, ajcc_beta2_dq,
                 ajcc_dry3_dq, ajcc_dry4_dq,
                 ajcc_wet4_dq, ajcc_wet5_dq, ajcc_wet6_dq,
                 num_pset_2, x1in, x4in, y2, y3);
}
z2 = x2in;
```

Functions *inputSignalModification()* and *applyTransientDucker()* are specified in ETSI TS 103 190-1 [1], clauses 5.7.7.4.2 and 5.7.7.4.3 respectively.

The pseudocode in pseudocode 13 shows the *ajcc_module_3()* function.

Pseudocode 13: ajcc_module_3()

```
ajcc_module_3(ajcc_dry1f, ajcc_dry2f,
              ajcc_wet1f, ajcc_wet2f, ajcc_wet3f,
              ajcc_dry1b, ajcc_dry2b,
              ajcc_wet1b, ajcc_wet2b, ajcc_wet3b,
              num_pset_1, num_pset_2, x0, x1, y0, y1)
{
  for (pb = 0; pb < ajcc_num_bands_table[ajcc_num_param_bands_id]; pb++) {
    for (ps = 0; ps < num_pset_1; ps++) {
      d0[ps][pb] = 1-ajcc_dry2f[ps][pb];
      d1[ps][pb] = 0;
      d2[ps][pb] = ajcc_dry2f[ps][pb];
      w0[ps][pb] = -
sqrt(0.5*ajcc_wet3f[ps][pb]*ajcc_wet3f[ps][pb] + 0.5*ajcc_wet2f[ps][pb]*ajcc_wet2f[ps][pb]);
      w1[ps][pb] = 0;
      w2[ps][pb] = sqrt(0.5*ajcc_wet3f[ps][pb]*ajcc_wet3f[ps][pb] + 0.5*ajcc_wet2f[ps][pb]*ajcc_wet2
f[ps][pb]);
    }
    for (ps = 0; ps < num_pset_2; ps++) {
      d3[ps][pb] = 0;
      d4[ps][pb] = ajcc_dry1b[ps][pb]+ajcc_dry2b[ps][pb];
      d5[ps][pb] = 1-ajcc_dry1b[ps][pb]-ajcc_dry2b[ps][pb];
      w3[ps][pb] = 0;
      w4[ps][pb] = -
sqrt(0.5*ajcc_wet1b[ps][pb]*ajcc_wet1b[ps][pb] + 0.5*ajcc_wet3b[ps][pb]*ajcc_wet3b[ps][pb]);
      w5[ps][pb] = sqrt(0.5*ajcc_wet1b[ps][pb]*ajcc_wet1b[ps][pb] + 0.5*ajcc_wet3b[ps][pb]*ajcc_wet3
b[ps][pb]);
    }
  }
  for (sb = 0; sb < num_qmf_subbands; sb++) {
    for (ts = 0; ts < num_qmf_timeslots; ts++) {
      interp_d0 = interpolate_ajcc(d0, num_pset_1, sb, ts);
      interp_d1 = interpolate_ajcc(d1, num_pset_1, sb, ts);
      interp_d2 = interpolate_ajcc(d2, num_pset_1, sb, ts);
      interp_d3 = interpolate_ajcc(d3, num_pset_2, sb, ts);
      interp_d4 = interpolate_ajcc(d4, num_pset_2, sb, ts);
      interp_d5 = interpolate_ajcc(d5, num_pset_2, sb, ts);
      interp_w0 = interpolate_ajcc(w0, num_pset_1, sb, ts);
      interp_w1 = interpolate_ajcc(w1, num_pset_1, sb, ts);
      interp_w2 = interpolate_ajcc(w2, num_pset_1, sb, ts);
      interp_w3 = interpolate_ajcc(w3, num_pset_2, sb, ts);
      interp_w4 = interpolate_ajcc(w4, num_pset_2, sb, ts);
      interp_w5 = interpolate_ajcc(w5, num_pset_2, sb, ts);

      z0[ts][sb] = interp_d0*x0[ts][sb] + interp_d3*x1[ts][sb] + interp_w0*y0[ts][sb] + interp_w3*y1
[ts][sb];
      z1[ts][sb] = interp_d1*x0[ts][sb] + interp_d4*x1[ts][sb] + interp_w1*y0[ts][sb] + interp_w4*y1
[ts][sb];
      z2[ts][sb] = interp_d2*x0[ts][sb] + interp_d5*x1[ts][sb] + interp_w2*y0[ts][sb] + interp_w5*y1
[ts][sb];
    }
  }
  return (z0, z1, z2);
}
```

The pseudocode in pseudocode 14 shows the *ajcc_module_4()* function.

Pseudocode 14: ajcc_module_4()

```
ajcc_module_4(ajcc_alpha, ajcc_beta,
              ajcc_dry1, ajcc_dry2,
              ajcc_wet1, ajcc_wet2, ajcc_wet3,
              num_pset, x0, x1, y0, y1)
{
  if (ajcc_core_mode == 0) {
    for (ps = 0; ps < num_pset; ps++) {
      for (pb = 0; pb < ajcc_num_bands_table[ajcc_num_param_bands_id]; pb++) {
        d0[ps][pb] = (1+ajcc_alpha[ps][pb])/2;
        d1[ps][pb] = 0;
        d2[ps][pb] = (1-ajcc_alpha[ps][pb])/2;
        d3[ps][pb] = 0;
        d4[ps][pb] = ajcc_dry1[ps][pb]+ajcc_dry2[ps][pb];
        d5[ps][pb] = 1-ajcc_dry1[ps][pb]-ajcc_dry2[ps][pb];
        w0[ps][pb] = ajcc_beta[ps][pb]/2;
        w1[ps][pb] = 0;
        w2[ps][pb] = -ajcc_beta[ps][pb]/2;
        w3[ps][pb] = 0;
        w4[ps][pb] = -
sqrt(0.5*ajcc_wet1[ps][pb]*ajcc_wet1[ps][pb] + 0.5*ajcc_wet3[ps][pb]*ajcc_wet3[ps][pb]);
        w5[ps][pb] = sqrt(0.5*ajcc_wet1[ps][pb]*ajcc_wet1[ps][pb] + 0.5*ajcc_wet3[ps][pb]*ajcc_wet3[
ps][pb]);
      }
    }
  }
  else {
    for (ps = 0; ps < num_pset; ps++) {
      for (pb = 0; pb < ajcc_num_bands_table[ajcc_num_param_bands_id]; pb++) {
        d0[ps][pb] = ajcc_dry1[ps][pb];
        d1[ps][pb] = 1-ajcc_dry1[ps][pb];
        d2[ps][pb] = 0;
        d3[ps][pb] = 0;
        d4[ps][pb] = 0;
        d5[ps][pb] = 1;
        w0[ps][pb] = sqrt(0.5*(ajcc_wet1[ps][pb]+ajcc_wet3[ps][pb])^2 + 0.5*(ajcc_wet3[ps][pb]+ajcc_
wet2[ps][pb])^2);
        w1[ps][pb] = -
sqrt(0.5*(ajcc_wet1[ps][pb]+ajcc_wet3[ps][pb])^2 + 0.5*(ajcc_wet3[ps][pb]+ajcc_wet2[ps][pb])^2);
        w2[ps][pb] = 0;
        w3[ps][pb] = 0;
        w4[ps][pb] = 0;
        w5[ps][pb] = 0;
      }
    }
  }
  for (sb = 0; sb < num_qmf_subbands; sb++) {
    for (ts = 0; ts < num_qmf_timeslots; ts++) {
      interp_d0 = interpolate_ajcc(d0, num_pset, sb, ts);
      interp_d1 = interpolate_ajcc(d1, num_pset, sb, ts);
      interp_d2 = interpolate_ajcc(d2, num_pset, sb, ts);
      interp_d3 = interpolate_ajcc(d3, num_pset, sb, ts);
      interp_d4 = interpolate_ajcc(d4, num_pset, sb, ts);
      interp_d5 = interpolate_ajcc(d5, num_pset, sb, ts);
      interp_w0 = interpolate_ajcc(w0, num_pset, sb, ts);
      interp_w1 = interpolate_ajcc(w1, num_pset, sb, ts);
      interp_w2 = interpolate_ajcc(w2, num_pset, sb, ts);
      interp_w3 = interpolate_ajcc(w3, num_pset, sb, ts);
      interp_w4 = interpolate_ajcc(w4, num_pset, sb, ts);
      interp_w5 = interpolate_ajcc(w5, num_pset, sb, ts);

      z0[ts][sb] = interp_d0*x0[ts][sb] + interp_d3*x1[ts][sb] + interp_w0*y0[ts][sb] + interp_w3*y1
[ts][sb];
      z1[ts][sb] = interp_d1*x0[ts][sb] + interp_d4*x1[ts][sb] + interp_w1*y0[ts][sb] + interp_w4*y1
[ts][sb];
      z2[ts][sb] = interp_d2*x0[ts][sb] + interp_d5*x1[ts][sb] + interp_w2*y0[ts][sb] + interp_w5*y1
[ts][sb];
    }
  }
  return (z0, z1, z2);
}
```

5.7 Advanced Joint Object Coding (A-JOC)

5.7.1 Introduction

The Advanced Joint Object Coding (A-JOC) tool is a coding tool for improved coding of a large number of audio objects. To gain coding efficiency this tool supports the reconstruction of audio objects out of a lower number of joint input audio objects and low overhead side information.

5.7.2 Interface

5.7.2.1 Input preparation

The output of decoding `var_channel_element` consists of m' matrices $Q'in_{\text{AJOC}}$. To obtain the right order of the m input matrices Qin_{AJOC} (see clause 5.7.2.1), the m' matrices $Q'in_{\text{AJOC}}$ shall be reordered as follows:

1) the value of `b_has_lfe` is taken from the respective `var_channel_element`.

NOTE 1: The LFE channel is treated separately, because this one is not processed by the A-JOC tool, but again combined with the output as described in clause 5.7.2.2.

2) m'_{fb}, the number of fullband input signals, is determined as: $m'_{\text{fb}} = m'$ - (`b_has_lfe` == 1 ? 1 : 0).

NOTE 2: The number m'_{fb} of fullband input signals is the same as the number m of input matrices to the A-JOC processing (see clause 5.7.2.1).

3) The m matrices Qin_{AJOC} are assigned to the m' matrices $Q'in_{\text{AJOC}}$ as specified in pseudocode 14a.

Pseudocode 14a: Input preparation for A-JOC processing

```
if (m´_fb > 3) {
  n_offset = 2 + (m´_fb mod 2);
  for (i = 0; i < n_offset; i++) {
    Qin_AJOC[i] = Q´in_AJOC[m´ - n_offset + i];
  }
  if (b_has_lfe == 1) {
    for (i = 0; i < m´_fb - n_offset; i++) {
      Qin_AJOC[i + n_offset] = Q´in_AJOC[i + 1];
    }
  }
  else {
    for (i = 0; i < m´_fb - n_offset; i++) {
      Qin_AJOC[i + n_offset] = Q´in_AJOC[i];
    }
  }
}
else {
  if (b_has_lfe == 1) {
    for (i = 0; i < m´_fb; i++) {
      Qin_AJOC[i] = Q´in_AJOC[i + 1];
    }
  }
  else {
    for (i = 0; i < m´_fb; i++) {
      Qin_AJOC[i] = Q´in_AJOC[i];
    }
  }
}
```

5.7.2.2 Inputs

Qin $_{\text{AJOC},[a|b|\ldots]}$ m_{AJOC} complex valued QMF matrices for m_{AJOC} objects to be processed by the A-JOC tool

The **Qin** $_{\text{AJOC}}$ matrices each consist of *num_qmf_timeslots* columns and *num_qmf_subbands* rows. m_{AJOC} is the number for *num_dmx_signals* objects in the coded A-JOC domain.

5.7.2.3 Outputs

Qout $_{\mathrm{AJOC},[a|b|\ldots]}$ n_{AJOC} complex valued QMF matrices

Q´out $_{\mathrm{AJOC},[a|b|\ldots]}$ n'_{AJOC} complex valued QMF matrices

The **Qout**$_{\mathrm{AJOC}}$ and **Q´out**$_{\mathrm{AJOC}}$ matrices each consist of *num_qmf_timeslots* columns and *num_qmf_subbands* rows. n_{AJOC} is the number for *num_umx_signals* reconstructed objects and n'_{AJOC} is the number of output matrices after possible re-insertion of an LFE object.

Q´out$_{\mathrm{AJOC}}$ shall be derived from **Qout**$_{\mathrm{AJOC}}$ by inserting the LFE signal at the position indicated by the corresponding A-JOC reconstruction parameter as specified in pseudocode 15.

Pseudocode 15: Output postprocessing after A-JOC processing

```
if (b_has_lfe == 1) {
  pos_lfe = 0;
  if (b_reconstruction_contains_Left_channel) {
    pos_lfe++;
  }
  if (b_reconstruction_contains_Right_channel) {
    pos_lfe++;
  }
  if (b_reconstruction_contains_Centre_channel) {
    pos_lfe++;
  }
  for (i = 0; i < pos_lfe; i++) {
    Q´out_AJOC[i] = Qout_AJOC[i];
  }
  Q´out_AJOC[pos_lfe] = Q´in_AJOC[0];
  for (i = pos_lfe; i < n_AJOC; i++) {
    Q´out_AJOC[i+1] = Qout_AJOC[i];
  }
}
else {
  for (i = 0; i < n_AJOC; i++) {
    Q´out_AJOC[i] = Qout_AJOC[i];
  }
}
```

5.7.2.4 Controls

The control information for the A-JOC tool consists of decoded and dequantized A-JOC side information. The side information contains parameters to control the dequantization process described in clause 5.7.3.3, the interpolation process described in clause 5.7.3.4, the decorrelation process described in clause 5.7.3.5, and coefficients of two submatrices - dry and wet - which are used in the reconstruction process described in clause 5.7.3.6.

5.7.3 Processing

5.7.3.1 Parameter band to QMF subband mapping

The AJOC parameters inside `ajoc_data()` are transmitted separately for each of the *ajoc_num_bands* parameter bands. The number of parameter bands is coded using the element `ajoc_num_bands_code`. The value of this element indicates the number of transmitted parameter bands, *ajoc_num_bands*, as shown in table 78.

The mapping of QMF subbands to parameter bands is shown in table 28.

Table 28: A-JOC parameter band to QMF subband mapping

QMF subband	A-JOC parameter band mapping dependent on *ajoc_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

5.7.3.2 Differential decoding

The pseudocode in pseudocode 16 describes the process to get quantized values $mtx\_dry\_q_o$ and $mtx\_wet\_q_o$ for A-JOC object o from the Huffman decoded values *mix_mtx_dry* and *mix_mtx_wet*, where $0 \leq o < n_{\mathrm{AJOC}}$ and $0 \leq ch < m_{\mathrm{AJOC}}$.

Pseudocode 16: Differential decoding for A-JOC objects

```
// differential decoding for A-JOC object o
// input: array mix_mtx_dry[o][dp][ch][pb]
// array mix_mtx_wet[o][dp][de][pb]
// array mtx_dry_q_prev[o][ch][pb]
// array mtx_wet_q_prev[o][de][pb]
// output: array mtx_dry_q[o][dp][ch][pb]
// array mtx_wet_q[o][dp][de][pb]

for (dp = 0; dp < ajoc_num_dpoints; dp++) {
  // dry matrix
  nquant = (ajoc_quant_select[o] == 1) ? 51 : 101;
  for (ch = 0; ch < num_dmx_signals; ch++) {
    if (ajoc_sparse_select == 1 && ajoc_sparse_mask_dry[o][ch] == 0) {
      for (pb = 0; pb < ajoc_num_bands[o]; pb++) {
        mtx_dry_q[o][dp][ch][pb] = (nquant - 1) / 2;
      }
    }
    else {
      if (diff_type[o][dp][ch] == 0) {    // DIFF_FREQ
        mtx_dry_q[o][dp][ch][0] = mix_mtx_dry[o][dp][ch][0];
        for (pb = 1; pb < ajoc_num_bands[o]; pb++) {
            mtx_dry_q[o][dp][ch][pb] = (mtx_dry_q[o][dp][ch][pb-1] +
                                        mix_mtx_dry[o][dp][ch][pb]) % nquant;
        }
      }
      else {  // DIFF_TIME
        for (pb = 0; pb < ajoc_num_bands[o]; pb++) {
          mtx_dry_q[o][dp][ch][pb] = mtx_dry_q_prev[o][ch][pb] +
                                     mix_mtx_dry[o][dp][ch][pb];
        }
      }
    }
```

```
    mtx_dry_q_prev[o][ch] = mtx_dry_q[o][dp][ch];
  }
  // wet matrix
  nquant = (ajoc_quant_select[o] == 1) ? 21 : 41;
  for (de = 0; de < ajoc_num_decorr; de++) {
    if (ajoc_sparse_select == 1 && ajoc_sparse_mask_wet[o][de] == 0) {
      for (pb = 0; pb < ajoc_num_bands[o]; pb++) {
        mtx_wet_q[o][dp][ch][pb] = (nquant - 1) / 2;
      }
    }
    else {
      if (diff_type[o][dp][de] == 0) {    // DIFF_FREQ
        mtx_wet_q[o][dp][de][0] = mix_mtx_wet[o][dp][de][0];
        for (pb = 1; pb < ajoc_num_bands[o]; pb++) {
          mtx_wet_q[o][dp][de][pb] = (mtx_wet_q[o][dp][de][pb-1] +
                                      mix_mtx_wet[o][dp][de][pb]) % nquant;
        }
      }
      else {  // DIFF_TIME
        for (pb = 0; pb < ajoc_num_bands[o]; pb++) {
          mtx_wet_q[o][dp][de][pb] = mtx_wet_q_prev[o][de][pb] +
                                     mix_mtx_wet[o][dp][de][pb];
        }
      }
    }
    mtx_wet_q_prev[o][de] = mtx_wet_q[o][dp][de];
  }
}
```

The quantized values from the last corresponding data point of the previous AC-4 frame, *mtx_dry_q_prev* and *mtx_wet_q_prev*, are needed when delta coding in the time direction across AC-4 frame boundaries.

5.7.3.3 Dequantization

The dequantized values *mtx_dry_dq* and *mtx_wet_dq* are obtained from *mtx_dry_q* and *mtx_wet_q* using table 29 and table 31 if the quantization mode is set to coarse (`ajoc_quant_select` = 1), and using table 30 and table 32 if the quantization mode is set to fine (`ajoc_quant_select` = 0). For inactive objects, *mtx_dry_dq* and *mtx_wet_dq* are set to 0,0.

The dry values are dequantized into ranges of -5,0048828 to +5,0048828. The wet values are dequantized into ranges of -2,0019531 to +2,0019531. The dequantization is based on a uniform quantization and the quantization steps for the dry values are identical to the dequantization of the advanced coupling parameter described in ETSI TS 103 190-1 [1], clause 5.7.7.7.

Table 29: Coarse dequantization of A-JOC dry values

mtx_dry_q	mtx_dry_dq
0	-5,0048828
1	-4,8046875
2	-4,6044922
3	-4,4042969
4	-4,2041016
5	-4,0039063
6	-3,8037109
7	-3,6035156
8	-3,4033203
9	-3,203125
10	-3,0029297
11	-2,8027344
12	-2,6025391
13	-2,4023438
14	-2,2021484
15	-2,0019531
16	-1,8017578
17	-1,6015625
18	-1,4013672
19	-1,2011719
20	-1,0009766
21	-0,8007813

mtx_dry_q	mtx_dry_dq
22	-0,6005859
23	-0,4003906
24	-0,2001953
25	0
26	0,2001953
27	0,4003906
28	0,6005859
29	0,8007813
30	1,0009766
31	1,2011719
32	1,4013672
33	1,6015625
34	1,8017578
35	2,0019531
36	2,2021484
37	2,4023438
38	2,6025391
39	2,8027344
40	3,0029297
41	3,203125
42	3,4033203
43	3,6035156
44	3,8037109
45	4,0039063
46	4,2041016
47	4,4042969
48	4,6044922
49	4,8046875
50	5,0048828

Table 30: Fine dequantization of A-JOC dry values

mtx_dry_q	mtx_dry_dq
0	-5,00488281
1	-4,90478516
2	-4,8046875
3	-4,70458984
4	-4,60449219
5	-4,50439453
6	-4,40429688
7	-4,30419922
8	-4,20410156
9	-4,10400391
10	-4,00390625
11	-3,90380859
12	-3,80371094
13	-3,70361328
14	-3,60351563
15	-3,50341797
16	-3,40332031
17	-3,30322266
18	-3,203125
19	-3,10302734
20	-3,00292969
21	-2,90283203
22	-2,80273438
23	-2,70263672
24	-2,60253906
25	-2,50244141
26	-2,40234375
27	-2,30224609
28	-2,20214844
29	-2,10205078

mtx_dry_q	mtx_dry_dq
30	-2,00195313
31	-1,90185547
32	-1,80175781
33	-1,70166016
34	-1,6015625
35	-1,50146484
36	-1,40136719
37	-1,30126953
38	-1,20117188
39	-1,10107422
40	-1,00097656
41	-0,90087891
42	-0,80078125
43	-0,70068359
44	-0,60058594
45	-0,50048828
46	-0,40039063
47	-0,30029297
48	-0,20019531
49	-0,10009766
50	0
51	0,100097656
52	0,200195313
53	0,300292969
54	0,400390625
55	0,500488281
56	0,600585938
57	0,700683594
58	0,80078125
59	0,900878906
60	1,000976563
61	1,101074219
62	1,201171875
63	1,301269531
64	1,401367188
65	1,501464844
66	1,6015625
67	1,701660156
68	1,801757813
69	1,901855469
70	2,001953125
71	2,102050781
72	2,202148438
73	2,302246094
74	2,40234375
75	2,502441406
76	2,602539063
77	2,702636719
78	2,802734375
79	2,902832031
80	3,002929688
81	3,103027344
82	3,203125
83	3,303222656
84	3,403320313
85	3,503417969
86	3,603515625
87	3,703613281
88	3,803710938
89	3,903808594
90	4,00390625
91	4,104003906
92	4,204101563
93	4,304199219

mtx_dry_q	mtx_dry_dq
94	4,404296875
95	4,504394531
96	4,604492188
97	4,704589844
98	4,8046875
99	4,904785156
100	5,004882813

Table 31: Coarse dequantization of A-JOC wet values

mtx_wet_q	mtx_wet_dq
0	-2,001953125
1	-1,801757813
2	-1,6015625
3	-1,401367188
4	-1,201171875
5	-1,000976563
6	-0,80078125
7	-0,600585938
8	-0,400390625
9	-0,200195313
10	0
11	0,200195313
12	0,400390625
13	0,600585938
14	0,80078125
15	1,000976563
16	1,201171875
17	1,401367188
18	1,6015625
19	1,801757813
20	2,001953125

Table 32: Fine dequantization of the wet values

mtx_wet_q	mtx_wet_dq
0	-2,00195313
1	-1,90185547
2	-1,80175781
3	-1,70166016
4	-1,6015625
5	-1,50146484
6	-1,40136719
7	-1,30126953
8	-1,20117188
9	-1,10107422
10	-1,00097656
11	-0,90087891
12	-0,80078125
13	-0,70068359
14	-0,60058594
15	-0,50048828
16	-0,40039063
17	-0,30029297
18	-0,20019531
19	-0,10009766
20	0
21	0,100097656
22	0,200195313
23	0,300292969
24	0,400390625
25	0,500488281
26	0,600585938
27	0,700683594
28	0,80078125
29	0,900878906
30	1,000976563
31	1,101074219
32	1,201171875
33	1,301269531
34	1,401367188
35	1,501464844
36	1,6015625
37	1,701660156
38	1,801757813
39	1,901855469
40	2,001953125

5.7.3.4 Parameter time interpolation

Decoded and dequantized advanced joint object coding parameters carried in the bitstream are time-interpolated for further processing. Each frame can carry one or two new parameter sets. In the case of interpolation over multiple frames, a frame can have no new parameter sets. The bitstream element `ajoc_num_dpoints` signals the number 0, 1 or 2 of parameter sets sent with the corresponding frame. The interpolation process for A-JOC differs from parameter interpolation for the A-CPL or A-JCC parameters. The interpolation of A-JOC parameters facilitates synchronization with the object audio metadata. The linear A-JOC parameter interpolation ramp is controlled with a ramp length value, which is signalled via `ajoc_ramp_len`, and a ramp starting point time slot, which is signalled via `ajoc_start_pos`. The maximum supported ramp length is 64 QMF time slots, which can cause an interpolation over multiple frame boundaries. Pseudocode 17 shows the *ajoc_interpolate()* function, which is used in clause 5.7.3.6.1.

Pseudocode 17: A-JOC interpolation

```
ajoc_interpolate(ajoc_param, prev_value, delta_inc, ts, curr_ramp_len, target_ramp_len)
{
  if (curr_ramp_len <= target_ramp_len) {
    interpolated = prev_value + delta_inc;
    prev_value = interpolated;
    curr_ramp_len++;
  }
  else {
    interpolated = prev_value;
  }

  for (dp = 0; dp < ajoc_num_dpoints; dp++) {
    if (ts == ajoc_start_pos[dp]) {
      delta_inc = (ajoc_param[dp] - prev_value) / ajoc_ramp_len[dp];
    }
  }

  return interpolated;
}
```

5.7.3.5 Decorrelator and transient ducker

This clause specifies function `ajoc_decorrelate()` as an extension of the decorrelator specified for advanced coupling in ETSI TS 103 190-1 [1], clause 5.7.7.4.2.

The A-JOC processing includes multiple decorrelation processes where *ajoc_num_decorr* parallel decorrelator instances generate the output signals:

$\mathbf{Qdecorr\_out}_{\mathrm{AJOC,[a|b|...]}}$ y_{AJOC} complex-valued QMF matrices; each one is the output of one of the *ajoc_num_decorr* parallel decorrelator instances

from the decorrelator input signals:

$\mathbf{Qdecorr\_in}_{\mathrm{AJOC,[a|b|...]}}$ u_{AJOC} complex-valued QMF matrices; each one is the input to one of the *ajoc_num_decorr* parallel decorrelator instances.

The $\mathbf{Qdecorr\_in}_{\mathrm{AJOC}}$ and $\mathbf{Qdecorr\_out}_{\mathrm{AJOC}}$ matrices each consist of *num_qmf_timeslots* columns and *num_qmf_subbands* rows.

The decorrelators used in A-JOC processing are identical to the decorrelators of the advanced coupling tool. The processing frequency subbands are grouped as described in ETSI TS 103 190-1 [1], clause 5.7.7.4.1. The coefficients of the three used decorrelators are described in ETSI TS 103 190-1 [1], clause 5.7.7.4.2. As the maximum number of active decorrelators is given by *ajoc_num_decorr = 7*, the three available decorrelators are used in a cyclic way: 0, 2, 1, 0, 2, 1, 0. The output signals of these decorrelators are also ducked as specified in ETSI TS 103 190-1 [1], clause 5.7.7.4.3.

5.7.3.6 Signal reconstruction using matrices

5.7.3.6.1 Processing

For advanced joint object decoding, *num_dmx_signals* input objects are present. Their elements are addressed as follows:

Input signals $x_{\mathrm{ch}}(ts,sb) \in \mathbf{Qin}_{\mathrm{AJOC,ch}}$ for ch = 0,…,*num_dmx_signals* - 1

The *num_umx_signals* output objects are addressed as:

Output signals $z_{\mathrm{o}}(ts,sb) \in \mathbf{Qout}_{\mathrm{AJOC,o}}$ for o = 0,…,*num_umx_signals* - 1

The reconstruction of the output signals requires additional internal signals:

Decorrelator input signals $u_{\mathrm{de}}(ts,sb) \in \mathbf{Qdecorr\_in}_{\mathrm{AJOC,de}}$ for de = 0,…,*ajoc_num_decorr* - 1

Decorrelator output signals $y_{\mathrm{de}}(ts,sb) \in \mathbf{Qdecorr\_out}_{\mathrm{AJOC,de}}$ for de = 0,…,*ajoc_num_decorr* - 1

The *num_umx_signals* output signals can be calculated from the *num_dmx_signals* input signals and the *ajoc_num_decorr* decorrelator output signals using reconstruction matrices **C** with *num_umx_signals* rows and *num_dmx_signals* + *ajoc_num_decorr* columns which are also time variant over the *qmf_timeslots* and frequency dependent on *qmf_subbands*:

$$z_o(ts,sb) = \sum_{ch=0}^{ndmx-1} C_{o,ch}(ts,sb) \times x_{ch}(ts,sb) + \sum_{de=0}^{ndcr-1} C_{o,ndmx+de}(ts,sb) \times y_{de}(ts,sb)$$

where *ndmx* = *num_dmx_signals* and *ndcr* = *ajoc_num_decorr*.

The reconstruction matrices **C**(*ts*,*sb*) can be divided into two sets of submatrices:

The submatrices $\mathbf{Csub1}_{o,ch}(ts,sb)$ factor the input signals $x_{ch}(ts,sb)$ to the output signals $z_o(ts,sb)$ and are called the dry submatrices. The submatrices $\mathbf{Csub2}_{o,de}(ts,sb)$ factor the decorrelator output signals $y_{de}(ts,sb)$ to the output signals $z_o(ts,sb)$ and are called the wet submatrices.

The matrix elements for both types of submatrices are calculated from the A-JOC bitstream via Huffman decoding (see clause 6.3.6.5), differential decoding (see clause 5.7.3.2) and dequantization (see clause 5.7.3.3), followed by an interpolation in time (see clause 5.7.3.4) and the ungrouping from parameter bands to QMF subbands (see clause 5.7.3.1).

The pseudocode in pseudocode 18 shows the reconstruction process.

Pseudocode 18: A-JOC reconstruction

```
ajoc_reconstruct(*z, *x,
                 *mtx_dry_dq, *mtx_wet_dq, *mtx_dry_prev, *mtx_wet_prev,
                 *delta_inc_dry, *deta_inc_wet)
{
  clear_output_matrices(*z, num_qmf_timeslots, num_qmf_subbands);

  /* calculate decorrelation input matrix parameter */
  for (dp = 0; dp < ajoc_num_dpoints; dp++) {
    for (o = 0; o < num_umx_signals; o++) {
      for (pb = 0; pb < ajoc_num_bands[o]; pb++) {
        for (ch = 0; ch < num_dmx_signals; ch++) {
          for (de = 0; de < ajoc_num_decorr; de++) {
            mtx_pre_param[pb][de][ch][dp] +=
              abs(mtx_wet_dq[o][dp][de][pb]) * mtx_dry_dq[o][dp][ch][pb];
          }
        }
      }
    }
  }

  if (b_ajoc_de_process) {
    (mtx_dry_dq, mtx_wet_dq) = ajoc_de_process(mtx_dry_dq, mtx_wet_dq, de_gain);
  }

  for (ts = 0; ts < num_qmf_timeslots; ts++) {
    for (sb = 0; sb < num_qmf_subbands; sb++) {

      /* interpolate */
      for (o = 0; o < num_umx_signals; o++) {
        for (ch = 0; ch < num_dmx_signals; ch++) {
          mtx_dry[ts][sb][o][ch] =
            ajoc_interpolate(mtx_dry_dq[o][][ch][sb_to_pb(sb)],
                             mtx_dry_prev[ts][sb][o][ch],
                             delta_inc_dry[ts][sb][o][ch],
                             ts,
                             curr_ramp_len,
                             target_ramp_len);
        }
        for (de = 0; de < ajoc_num_decorr; de++) {
          mtx_wet[ts][sb][o][de] =
            ajoc_interpolate(mtx_wet_dq[o][][de][sb_to_pb(sb)],
                             mtx_wet_prev[ts][sb][o][de],
                             delta_inc_wet[ts][sb][o][de],
                             ts,
                             curr_ramp_len,
                             target_ramp_len);
        }
      }
```

```
      for (ch = 0; ch < num_dmx_signals; ch++) {
        for (de = 0; de < ajoc_num_decorr; de++) {
          mtx_pre[ts][sb][de][ch] =
            ajoc_interpolate(mtx_pre_param[sb_to_pb(sb)][de][ch],
                             mtx_pre_prev[ts][sb][de][ch],
                             delta_inc_pre[ts][sb][de][ch],
                             ts,
                             curr_ramp_len,
                             target_ramp_len);
        }
      }

      for (de = 0; de < ajoc_num_decorr; de++) {
        /* decorrelation pre-matrix */
        u[ts][sb][de] = 0;
        for (ch = 0; ch < num_dmx_signals; ch++) {
          u[ts][sb][de] += mtx_pre[ts][sb][de][ch] * x[ts][sb][ch];
        }

        /* decorrelation process */
        y[ts][sb][de] = ajoc_decorrelate(u[ts][sb][de]);
      }

      /* reconstruct */
      for (o = 0; o < num_umx_signals; o++) {
        for (ch = 0; ch < num_dmx_signals; ch++) {
          z[ts][sb][o] += x[ts][sb][ch] * mtx_dry[ts][sb][o][ch];
        }
        for (de = 0; de < ajoc_num_decorr; de++) {
          z[ts][sb][o] += y[ts][sb][de] * mtx_wet[ts][sb][o][de];
        }
      }
    }

    /* update curr_ramp_len and target_ramp_len */
    for (dp = 0; dp < ajoc_num_dpoints; dp++) {
      if (ts == ajoc_start_pos[dp]) {
        curr_ramp_len = 0;
        target_ramp_len = ajoc_ramp_len[dp];
      }
    }
  }
}
```

NOTE: *clear_output_matrices()* is a helper function that sets all elements of the output signal matrix *z* to zero.

ajoc_decorrelate() calculates the decorrelator output as outlined in clause 5.7.3.5. This includes usage of the decorrelator input matrix *mtx_pre* as calculated by the pseudocode in pseudocode 18 to create the decorrelator input signals.

sb_to_pb() is a helper function that maps from QMF subbands to A-JOC parameter bands according to table 28.

mtx_dry_dq is the matrix of A-JOC parameter for the dry submatrix for reconstruction with the dimension *mtx_dry_dq[num_umx_signals][ajoc_num_dpoints][num_dmx_signals][ajoc_num_bands[o]]*.

mtx_wet_dq is the matrix of A-JOC parameter for the wet submatrix for reconstruction with the dimension *mtx_wet_dq[num_umx_signals][ajoc_num_dpoints][ajoc_num_decorr][ajoc_num_bands[o]]*.

mtx_dry_prev is the matrix of dry submatrix elements stored from the previous call to *ajoc_reconstruct()* with the dimension *mtx_dry_prev[num_qmf_timeslots][num_qmf_subbands][num_umx_signals][num_dmx_signals]*.

mtx_wet_prev is the matrix of wet submatrix elements stored from the previous call to *ajoc_reconstruct()* with the dimension *mtx_wet_prev[num_qmf_timeslots][num_qmf_subbands][num_umx_signals][ajoc_num_decorr]*.

mtx_pre_prev is the matrix of pre-matrix elements stored from the previous call to *ajoc_reconstruct()* with the dimension *mtx_pre_prev[num_qmf_timeslots][num_qmf_subbands][ajoc_num_decorr][num_dmx_signals]*.

delta_inc_dry is an array of values that stores the incremental delta value to be added by the interpolation process with the dimension *delta_inc_dry[num_qmf_timeslots][num_qmf_subbands][num_umx_signals][num_dmx_signals]*.

delta_inc_wet is an array of values that stores the incremental delta value to be added by the interpolation process with the dimension *delta_inc_wet[num_qmf_timeslots][num_qmf_subbands][num_umx_signals][ajoc_num_decorr]*.

delta_inc_pre is an array of values that stores the incremental delta value to be added by the interpolation process with the dimension *delta_inc_pre[num_qmf_timeslots][num_qmf_subbands][ajoc_num_decorr][num_dmx_signals]*.

b_ajoc_de_process indicates whether the dialogue enhancement operation is activated.

ajoc_de_process() is a function that applies dialogue enhancement and is specified in clause 5.8.2.3. If the decoding mode is full decoding, the parameter *de_gain* is specified in clause 5.8.2.3; if the decoding mode is core decoding, *de_gain* is specified in clause 5.8.2.4.

5.7.3.6.2 Decorrelation input matrix

The input signals for the parallel decorrelators are:

Decorrelator input signals $u_{de}(ts,sb)$ □ **Qdecorr_in**$_{AJOC,de}$ for de = 0, …, *ajoc_num_decorr* - 1

The decorrelation input signals u_{de} can be calculated from the input signals x_{ch} using decorrelation input matrices **D**(ts,sb) with *ajoc_num_decorr* rows and *num_dmx_signals* columns:

$$u_{de}(ts, sb) = \sum_{ch=0}^{num\_dmx\_signals-1} D_{de,ch}(ts, sb) x_{ch}(ts, sb)$$

The decorrelation input matrix can be calculated using the dry submatrix $\mathbf{Csub1}_{o,ch}$ and the wet submatrix $\mathbf{Csub2}_{o,de}$:

$$D(ts, sb) = |Csub2(ts, sb)^T| \times Csub1(ts, sb)$$

5.8 Dialogue enhancement for immersive audio

5.8.1 Introduction

This dialogue enhancement tool specification extends the specification of the dialogue enhancement tool in ETSI TS 103 190-1 [1], clause 5.7.8 to support dialogue enhancement for immersive object audio content and dialogue enhancement for core decoding of A-JCC or A-CPL coded immersive channel audio content.

5.8.2 Processing

5.8.2.1 Dialogue enhancement for core decoding of A-JCC coded 9.X.4 content

This clause specifies the dialogue enhancement processing for core decoding of A-JCC coded 9.X.4 content where the dialogue enhancement operation is performed right after the A-JCC core.

This tool is an extension to the dialogue enhancement tool specified in ETSI TS 103 190-1 [1], clause 5.7.8. core decoding of A-JCC content is specified in clause 5.6.3.5.3.

A-JCC coded 9.X.4 content is content coded in an `immersive_channel_element` where *immersive_codec_mode* is set to ASPX_AJCC and where `b_5fronts` is true. The input signals for the dialogue enhancement processing are a subset of the A-JCC related signals (input and output) specified in clause 5.6.2.

The dialogue enhancement shall be performed by the following process:

$$y(ts, sb) = H_{DE,AJCC,Core}(ts, sb) \times \begin{pmatrix} m(ts, sb) \\ u(ts, sb) \end{pmatrix}$$

for $0 \le ts < num\_qmf\_timeslot$ and for $0 \le sb < num\_qmf\_subbands$.

m(ts,sb) is a vector of three input signals to the A-JCC processing:

$$m = \begin{pmatrix} Qin_{AJCC,A} \\ Qin_{AJCC,B} \\ Qin_{AJCC,C} \end{pmatrix}$$

u(ts,sb) is a vector of three output signals of the A-JCC processing:

$$u = \begin{pmatrix} Qout_{AJCC,L} \\ Qout_{AJCC,R} \\ Qout_{AJCC,C} \end{pmatrix}$$

The matrix $H_{DE,AJCC,Core}$ shall be derived from a matrix $M_{interp}(ts,sb)$ as:

$$H_{DE,AJCC,Core}(ts, sb) = \left(M_{interp}(ts, sb) \middle| I\right)$$

where I is the 3 × 3 identity matrix and | is the horizontal concatenation operator.

The matrix $M_{interp}(ts,sb)$ shall be calculated from two A-JCC coefficients C_L and C_R and a modified dialogue enhancement (DE) matrix:

$$\hat{H}_{DE,Core} = \hat{H}_{DE,MC} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} - I$$

where $\hat{H}_{DE,MC}$ is specified in ETSI TS 103 190-1 [1], clause 5.7.8.6 and I is the 3 × 3 identity matrix. The calculation of $M_{interp}(ts,sb)$ is specified in pseudocode 20. The input parameters *int_type*, *num_ps* and *psts* shall be initialized to:

int_type = (`ajcc_it_lf`, `ajcc_it_rf`)

$$num\_ps = (\texttt{ajcc\_nps\_lf}, \texttt{ajcc\_nps\_rf})$$

$$psts = (\texttt{ajcc\_psts\_lf}, \texttt{ajcc\_psts\_rf})$$

The calculation of the coefficients $C_L = C\_L$ and $C_R = C\_R$ is specified in pseudocode 19.

Pseudocode 19: Calculation of A-JCC coefficients

```
for (ps = 0; ps < num_ps[0]; ps++) {
  for (pb = 0; pb < ajcc_num_param_bands; pb++) {
    C_L[ps][pb] = 1 - ajcc_dry1f_dq[ps][pb] - ajcc_dry2f_dq[ps][pb];
  }
}
for (ps = 0; ps < num_ps[1]; ps++) {
  for (pb = 0; pb < ajcc_num_param_bands; pb++) {
    C_R[ps][pb] = 1 - ajcc_dry3f_dq[ps][pb] - ajcc_dry4f_dq[ps][pb];
  }
}
```

Pseudocode 20: Dialogue enhancement matrix calculation and interpolation

```
joint_interpolation(de_param, coeff_l, coeff_r, int_type, num_ps, psts)
{
  int_type[2] = 0;
  num_ps[2] = 1;
  coeff[0] = coeff_l;
  coeff[1] = coeff_r;
  coeff[2] = 1;
  for (sb = 0; sb < num_qmf_subbands; sb++) {
    ab = sb_to_pb(sb);
    db = sb_to_pb_de(sb);
    for (ch2 = 0; ch2 < 3; ch2++) {
      for (ts = 0; ts < num_qmf_timeslots; ts++) {
        if (int_type[ch2] == 0) {   // Smooth interpolation
          if (num_ps[ch2] == 1) {   // 1 parameter set
            if (ts == 0) {
              for (ch1 = 0; ch1 < 3; ch1++) {
                Mtgt[ch1][ch2] = de_param[db][ch1][ch2] * coeff[ch2][0][ab];
                delta[ch1][ch2] = (Mtgt[ch1][ch2] - Mprev[sb][ch1][ch2])
                    / num_qmf_timeslots;
              }
              coeff_prev[ch2][sb] = coeff[ch2][0][ab];
            }
          } else {                  // 2 parameter sets
            if (ts == 0) {
              for (ch1 = 0; ch1 < 3; ch1++) {
                delta_de = (de_param[db][ch1][ch2] -
                    de_param_prev[sb][ch1][ch2]) / num_qmf_timeslots;
                Mde = de_param_prev[sb][ch1][ch2]
                    + floor(num_qmf_timeslots / 2) * delta_de;
                Mtgt[ch1][ch2] = Mde * coeff[ch2][0][ab];
                delta[ch1][ch2] = (Mtgt[ch1][ch2] - Mprev[sb][ch1][ch2])
                    / floor(num_qmf_timeslots / 2);
              }
            } elseif (ts == floor(num_qmf_timeslots / 2)) {
              for (ch1 = 0; ch1 < 3; ch1++) {
                Mprev[sb][ch1][ch2] = Mtgt[ch1][ch2];
                Mtgt[ch1][ch2] = de_param[db][ch1][ch2]
                    * coeff[ch2][1][ab];
                delta[ch1][ch2] = (Mtgt[ch1][ch2] - Mprev[sb][ch1][ch2])
                    / (num_qmf_timeslots - floor(num_qmf_timeslots / 2));
              }
              coeff_prev[ch2][sb] = coeff[ch2][1][ab];
            }
          }
        } else {                    // Steep interpolation
          if (num_ps[ch2] == 1) {   // 1 parameter set
            if (ts == 0) {
              for (ch1 = 0; ch1 < 3; ch1++) {
                delta_de[ch1][ch2] = (de_param[db][ch1][ch2] -
                    de_param_prev[sb][ch1][ch2]) / num_qmf_timeslots;
                Mde[ch1][ch2] = de_param_prev[sb][ch1][ch2]
                    + psts[ch2][0] * delta_de[ch1][ch2];
                Mtgt[ch1][ch2] = Mde[ch1][ch2] * coeff_prev[ch2][sb];
                delta[ch1][ch2] = (Mtgt[ch1][ch2] - Mprev[sb][ch1][ch2])
```

```
                    / psts[ch2][0];
              }
            } elseif (ts == psts[ch2][0]) {
              for (ch1 = 0; ch1 < 3; ch1++) {
                Mde[ch1][ch2] = Mde[ch1][ch2] + delta_de[ch1][ch2];
                Minterp[ts][sb][ch1][ch2] = Mde[ch1][ch2] * coeff[ch2][0][ab];
                Mtgt[ch1][ch2] = de_param[db][ch1][ch2] * coeff[ch2][0][ab];
                delta[ch1][ch2] = (Mtgt[ch1][ch2] - Minterp[ts][sb][ch1][ch2])
                    / (num_qmf_timeslots - psts[ch2][0] - 1);
              }
              coeff_prev[ch2][sb] = coeff[ch2][0][ab];
              ts++;
            }
          } else {                  // 2 parameter sets
            if (ts == 0) {
              for (ch1 = 0; ch1 < 3; ch1++) {
                delta_de[ch1][ch2] = (de_param[db][ch1][ch2] -
                    de_param_prev[sb][ch1][ch2]) / num_qmf_timeslots;
                Mde[ch1][ch2] = de_param_prev[sb][ch1][ch2]
                    + psts[ch2][0] * delta_de[ch1][ch2];
                Mtgt[ch1][ch2] = Mde[ch1][ch2] * coeff_prev[ch2][sb];
                delta[ch1][ch2] = (Mtgt[ch1][ch2] - Mprev[sb][ch1][ch2])
                    / psts[ch2][0];
              }
            } elseif (ts == psts[ch2][0]) {
              for (ch1 = 0; ch1 < 3; ch1++) {
                Mde[ch1][ch2] = Mde[ch1][ch2] + delta_de[ch1][ch2];
                Minterp[ts][sb][ch1][ch2] = Mde[ch1][ch2]
                    * coeff[ch2][0][ab];
                Mde[ch1][ch2] = de_param_prev[sb][ch1][ch2]
                    + psts[ch2][1] * delta_de[ch1][ch2];
                Mtgt[ch1][ch2] = Mde[ch1][ch2] * coeff[ch2][0][ab];
                delta[ch1][ch2] = (Mtgt[ch1][ch2] - Minterp[ts][sb][ch1][ch2])
                    / (psts[ch2][1] - psts[ch2][0] - 1);
              }
              ts++;
            } elseif (ts == psts[ch2][1]) {
              for (ch1 = 0; ch1 < 3; ch1++) {
                Mde[ch1][ch2] = Mde[ch1][ch2] + delta_de[ch1][ch2];
                Minterp[ts][sb][ch1][ch2] = Mde[ch1][ch2] * coeff[ch2][1][ab];
                Mtgt[ch1][ch2] = de_param[db][ch1][ch2] * coeff[ch2][1][ab];
                delta[ch1][ch2] = (Mtgt[ch1][ch2] - Minterp[ts][sb][ch1][ch2])
                    / (num_qmf_timeslots - psts[ch2][1] - 1);
              }
              coeff_prev[ch2][sb] = coeff[ch2][1][ab];
              ts++;
            }
          }
        }
        for (ch1 = 0; ch1 < 3; ch1++) {
          if (ts == 0) {
            Minterp[ts][sb][ch1][ch2] = Mprev[sb][ch1][ch2] + delta[ch1][ch2];
          } else {
            Minterp[ts][sb][ch1][ch2] = Minterp[ts - 1][sb][ch1][ch2]
                + delta[ch1][ch2];
          }
        }
      }
    }
    for (ch1 = 0; ch1 < 3; ch1++) {
      for (ch2 = 0; ch2 < 3; ch2++) {
        Mprev[sb][ch1][ch2] = Mtgt[ch1][ch2];
        de_param_prev[sb][ch1][ch2] = de_param[db][ch1][ch2];
      }
    }
  }
  return Minterp;
}
```

NOTE: de_param = $\hat{H}_{DE,Core}$, coeff_l = C_L, coeff_r = C_R, Minterp = M_{interp}

The *sb_to_pb()* helper function maps from QMF subbands to A-JCC parameter bands according to clause 5.6.3.1. The *sb_to_pb_de()* helper function maps from QMF subbands to dialogue enhancement parameter bands according to ETSI TS 103 190-1 [1], clause 5.7.8.4.

5.8.2.2 Dialogue enhancement for core decoding of parametric A-CPL coded 9.X.4 content

NOTE: This tool is an extension to the dialogue enhancement tool specified in ETSI TS 103 190-1 [1], clause 5.7.8.

Parametric A-CPL coded 9.X.4 content is content coded in an `immersive_channel_element`, where *immersive_codec_mode* is set to `ASPX_ACPL_2` and where `b_5fronts` is true. The input signals for the dialogue enhancement processing are a subset of the A-CPL input signals specified in ETSI TS 103 190-1 [1], clause 5.7.7.1.

The dialogue enhancement shall be performed by the following process:

$$y(ts, sb) = \mathrm{H}_{\mathrm{DE,ACPL,Core}}(ts, sb) \times m(ts, sb)$$

for $0 \le ts < num\_qmf\_timeslot$ and for $0 \le sb < num\_qmf\_subbands$.

$m(ts,sb)$ is a vector of three input signals to the A-CPL processing:

$$m = \begin{pmatrix} \mathrm{Qin}_{\mathrm{ACPL,a}} \\ \mathrm{Qin}_{\mathrm{ACPL,b}} \\ \mathrm{Qin}_{\mathrm{ACPL,c}} \end{pmatrix}$$

The matrix $H_{\mathrm{DE,ACPL,Core}}$ shall be derived from a matrix $M_{\mathrm{interp}}(ts,sb)$ as

$$\mathrm{H}_{\mathrm{DE,ACPL,Core}}(ts, sb) = \left(\mathrm{M}_{\mathrm{interp}}(ts, sb) \middle| \mathrm{I}\right)$$

where I is the 3 × 3 identity matrix and | is the horizontal matrix concatenation operator.

The matrix $M_{\mathrm{interp}}(ts,sb)$ shall be calculated from two A-CPL coefficients C_{L} and C_{R} and a modified dialogue enhancement matrix:

$$\hat{\mathrm{H}}_{\mathrm{DE,Core}} = \hat{\mathrm{H}}_{\mathrm{DE,MC}} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} - I$$

where $\hat{\mathrm{H}}_{\mathrm{DE,MC}}$ is specified in ETSI TS 103 190-1 [1], clause 5.7.8.6 and I is the 3 × 3 identity matrix. The calculation of $M_{\mathrm{interp}}(ts,sb)$ is specified in pseudocode 20. The input parameters *int_type*, *num_ps* and *psts* shall be initialized to:

int_type = `acpl_interpolation_type`

num_ps = `acpl_num_param_sets`

psts = `acpl_param_timeslot`

Each of these variables is an array of two dequantized elements, where the first one shall be derived from the fifth `acpl_data_1ch` element and the second one from the sixth `acpl_data_1ch` element present in the `immersive_channel_element`.

The calculation of the coefficients $C_{\mathrm{L}} = C\_L$ and $C_{\mathrm{R}} = C\_R$ is specified in pseudocode 21.

Pseudocode 21: Calculation of A-CPL coefficients

```
for (ps = 0; ps < num_ps[0];  ps++) {
  for (pb = 0; pb < acpl_num_param_bands; pb++) {
    C_L[ps][pb] = 0.5 * (1 - acpl_alpha5_dq[ps][pb]);
  }
}
for (ps = 0; ps < num_ps[1];  ps++) {
  for (pb = 0; pb < acpl_num_param_bands; pb++) {
    C_R[ps][pb] = 0.5 * (1 - acpl_alpha6_dq[ps][pb]);
  }
}
```

The *acpl_alpha5_dq* function shall be derived from the `acpl_alpha1` value in the fifth `acpl_data_1ch` element and the *acpl_alpha6_dq* function shall be derived from the `acpl_alpha1` value in the sixth `acpl_data_1ch` element present in the `immersive_channel_element`.

5.8.2.3 Dialogue enhancement for full decoding of A-JOC coded content

NOTE: This tool is an extension to the dialogue enhancement tool specified in ETSI TS 103 190-1 [1], clause 5.7.8.

A-JOC coded content is content coded in an `audio_data_ajoc` element. If dialogue enhancement is enabled for full decoding of A-JOC coded content the dialogue enhancement processing shall be done by a modification of the A-JOC parameters. The function *ajoc_de_process()*, which is specified in pseudocode 22, shall be called during the A-JOC processing as specified in pseudocode 18. The *de_gain* value for full decoding is specified as:

$$\mathrm{de\_gain} = \begin{cases} 10^{G_{DE}/20} & if\, G_{DE} < G_{max} \\ 10^{G_{max}/20} & \text{else} \end{cases}$$

where G_{max} shall be derived from `de_max_gain`.

Pseudocode 22: A-JOC dialogue enhancement processing

```
ajoc_de_process(mtx_dry_dq, mtx_wet_dq, de_gain)
{
  if (de_gain > 1)
  {
    (num_dlg_obj, dlg_idx) = dlg_obj();
    for (d = 0; d < num_dlg_obj; d++)
    {
      for (dp = 0; dp < ajoc_num_dpoint; dp++)
      {
        for (pb = 0; pb < ajoc_num_bands[dlg_idx[d]]; pb++)
        {
          for (ch = 0; ch < num_dmx_signals; ch++)
          {
            mtx_dry_dq[dlg_idx[d]][dp][ch][pb] *= de_gain;
          }
          for (de = 0; de < ajoc_num_decorr; de++)
          {
            mtx_wet_dq[dlg_idx[d]][dp][de][pb] *= de_gain;
          }
        }
      }
    }
  }
  return (mtx_dry_dq, mtx_wet_dq);
}
```

The *dlg_obj()* function is specified in pseudocode 28.

5.8.2.4 Dialogue enhancement for core decoding of A-JOC coded content

NOTE: This tool is an extension to the dialogue enhancement tool specified in ETSI TS 103 190-1 [1], clause 5.7.8.

When decoding A-JOC content in core decoding mode, dialogue enhancement shall be applied according to the matrix equation:

$$y_{ch}(ts, sb) = H_M(ts, sb) \times H_A(ts, sb) \times x_{ch}(ts, sb) + x_{ch}(ts, sb)$$

where $x_{ch}(ts,sb)$ correspond to the signals $Qin_{AJOC,[a|b|\ldots]}$ specified in clause 5.7.2.1 and $y_{ch}(ts,sb)$ is the dialogue enhanced output for *ch* = a,b,… and the QMF timeslot *ts* and the QMF subband *sb*.

The matrix $H_A(ts,sb)$ is representing a partial A-JOC processing and shall be derived from the matrix *mtx_dry* which is calculated by the function *ajoc_reconstruct()* as specified in pseudocode 18. The function *ajoc_de_process()* is specified in pseudocode 22, where *de_gain* for core decoding is specified as:

$$\mathrm{de\_gain} = \begin{cases} 10^{G_{DE}/20} - 1 & if\, G_{DE} < G_{max} \\ 10^{G_{max}/20} - 1 & \text{else} \end{cases}$$

where G_{max} shall be derived from `de_max_gain`. $H_A(ts,sb)$ shall be calculated from *mtx_dry* as specified in pseudocode 23.

Pseudocode 23: Calculation of matrix H_A

```
calculate_H_A(ts, sb)
{
  [num_dlg_obj, dlg_idx] = dlg_obj();
  for (dlg = 0; dlg < num_dlg_obj; dlg++)
  {
    for (ch = 0; ch < num_dmx_signals; ch++)
    {
      H_A[ts][sb][dlg][ch] = mtx_dry[ts][sb][dlg_idx[dlg]][ch];
    }
  }
  return H_A;
}
```

The matrix $H_M(ts,sb)$ shall be the result of an interpolation process between the matrix $H'_{M,prev}$ and H'_M, where H'_M shall be calculated for each codec frame and after the interpolation process stored as $H'_{M,prev}$ for the interpolation process in the next codec frame. The interpolation process is specified as:

$$H_M(ts, sb) = \left(1 - \frac{ts+1}{\mathrm{num\_qmf\_timeslots}}\right) \times H'_{M,prev}(sb) + \frac{ts+1}{\mathrm{num\_qmf\_timeslots}} \times H'_M(sb)$$

where the calculation of H'_M shall be done as specified in pseudocode 24. The variable *de_dlg_dmx_coeff* shall be derived from `de_dlg_dmx_coeff_idx` according to table 82.

Pseudocode 24: Calculation of matrix H'_M

```
calculate_H_M_dash(sb)
{
  (num_dlg_obj, dlg_idx) = dlg_obj();
  for (dlg = 0; dlg < num_dlg_obj; dlg++)
  {
    for (ch = 0; ch < num_dmx_signals; ch++)
    {
      H_M_dash[sb][ch][dlg] = de_dlg_dmx_coeff[dlg][ch];
    }
  }
  return H_M_dash;
}
```

The *dlg_obj()* helper function is specified in pseudocode 28.

5.8.2.5 Dialogue enhancement for non A-JOC coded object audio content

NOTE: This tool is an extension to the dialogue enhancement tool specified in ETSI TS 103 190-1 [1], clause 5.7.8.

Non A-JOC coded object audio content is content coded in `audio_data_objs`. If dialogue enhancement is enabled for decoding of non A-JOC coded object audio content and the processed substream is a dialogue substream, the decoder shall apply a gain factor as follows:

$$\mathrm{de\_gain} = \begin{cases} 10^{G_{DE}/20} & \text{if} G_{DE} < G_{max} \\ 10^{G_{max}/20} & \text{else} \end{cases}$$

to the corresponding audio objects for which `b_dialog` is true. Hence, the decoder should apply the gain factor to the corresponding object essences.

If b_dialog_max_gain is true, G_{max} shall be calculated from dialog_max_gain as:

$$G_{max} = 3 \times (1+\text{dialog\_max\_gain})[\text{dB}]$$

Otherwise, G_{max} shall be 0 dB.

5.9 Object audio metadata timing

5.9.1 Introduction

The AC-4 bitstream supports multiple updates of object properties per codec frame, where each update is contained in one obj_info_block (called block update). The OAMD timing data specifies how updates of the object properties are synchronized to the PCM audio samples of the object audio essence.

5.9.2 Synchronization of object properties

The decoder shall synchronize the object properties, contained in multiple block updates, to the PCM audio samples. The decoder may split up the object audio essence into subblocks of PCM audio samples. These subblocks of samples are associated with properties of the corresponding block update and are consecutively output of the decoder to be processed by an object audio renderer.

The PCM audio sample, at which the corresponding block update starts to be effective, is called *update* PCM *sample*. An update PCM sample shall be calculated for each present block update, which is an offset to the first PCM sample of the actual codec frame. One block update is valid until the update PCM sample of the next received block update.

The update PCM sample value $update\_sample_n$ for the block update n shall be calculated from sample_offset, which is shared by all block updates, and the block_offset_factor of the corresponding block update as $update\_sample_n = \text{sample\_offset} + (32 \times \text{block\_offset\_factor}_n)$.

Figure 10 shows the update PCM samples for received block updates and how block updates are stored in the AC-4 bitstream.

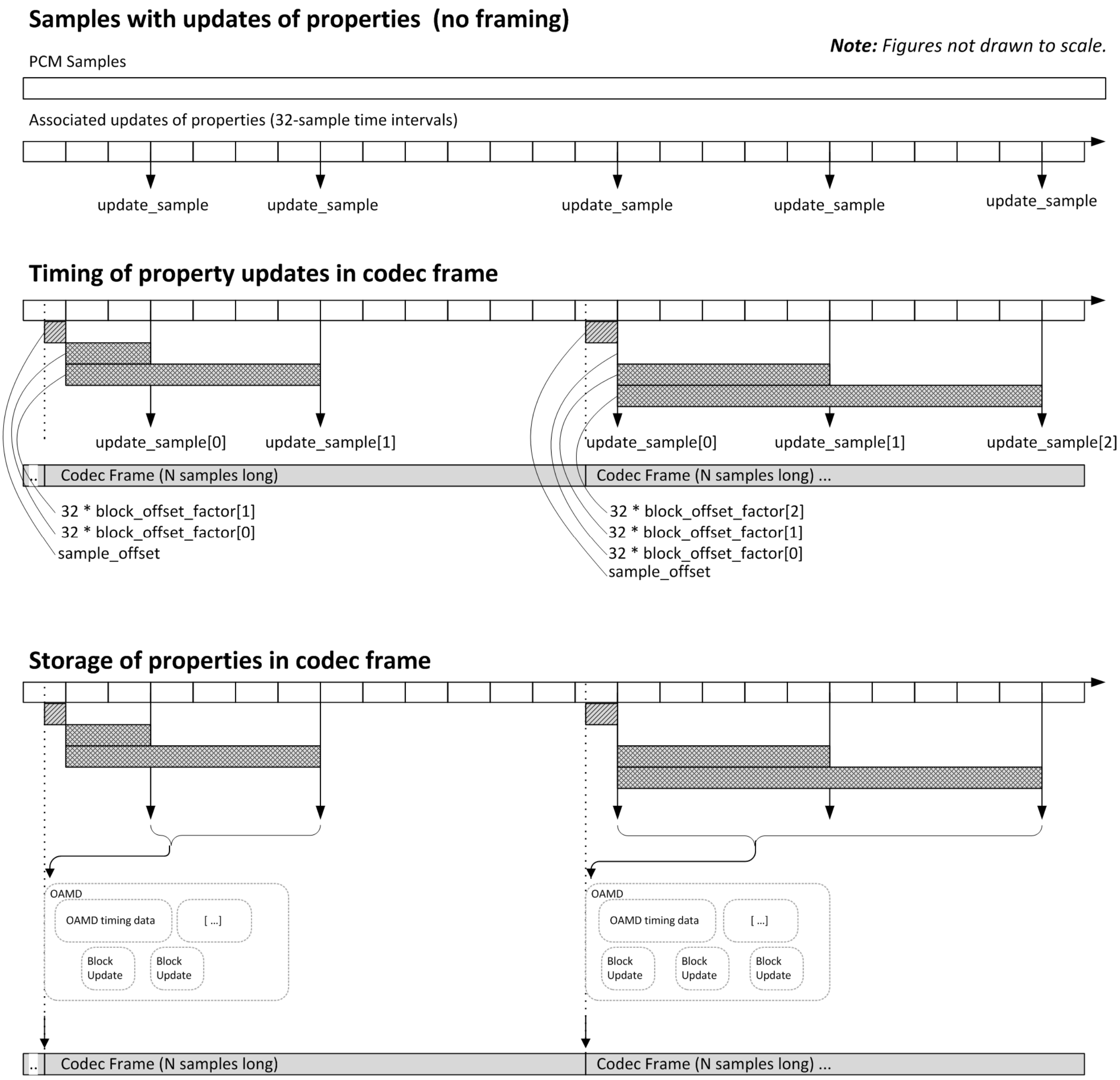


Figure 10: OAMD block updates

5.10 Rendering

5.10.1 Introduction

This topic specifies several different algorithms for rendering audio tracks into output channels corresponding to a chosen speaker layout.

Two different renderers are defined:

Channel based renderer	Downmixes or up-maps input channels to output channels.
ISF **renderer**	Renders the intermediate spatial format.

5.10.2 Channel audio renderer

5.10.2.1 Introduction

This clause describes how decoded audio substreams are rendered to the output channel configuration.

The process is illustrated in figure 11 and is applicable to the processing of each channel audio substream of a presentation.

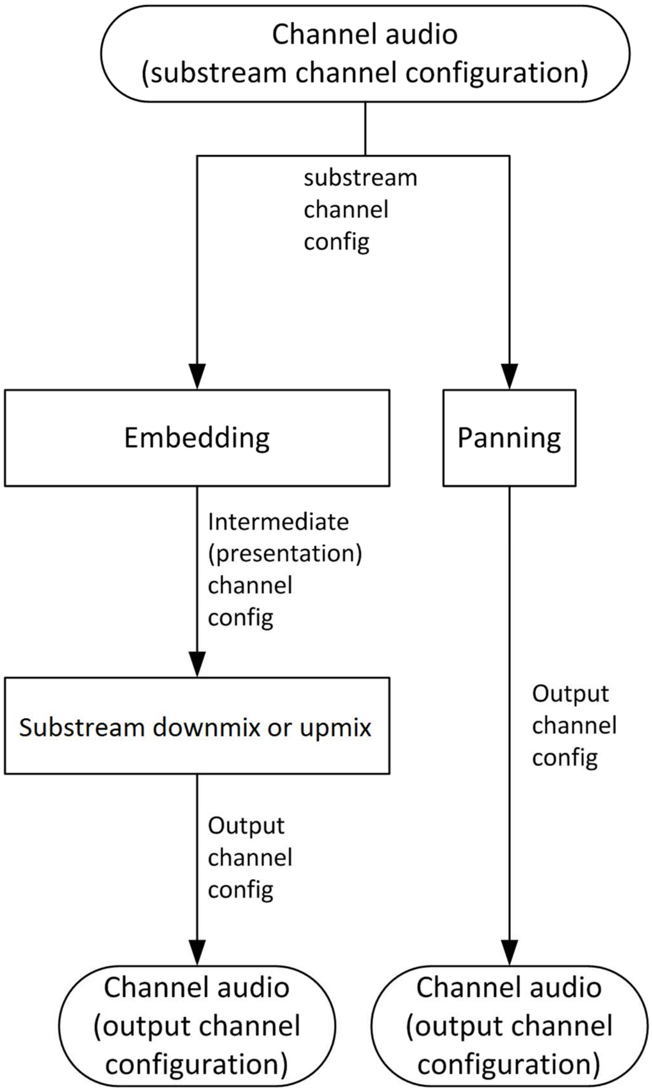


Figure 11: Channel rendering process

The channel rendering process shall perform the following steps:

1) When `pan_dialog` or `pan_associated` is present, the decoder shall pan the dialogue or associated audio signals to the horizontal-plane speakers of the output channel configuration, as described in clause 5.10.2.3.

2) If the presentation is an exclusively channel-based presentation (i.e. `pres_ch_mode` $\neq$ -1 or `pres_ch_mode_core` $\neq$ -1), then all other signals shall be embedded into a channel configuration that reflects the presentation channel mode as indicated by `pres_ch_mode` for full decoding mode or `pres_ch_mode_core` for core decoding mode.

3) The resulting channel configuration may be up-mixed or down-mixed to the output channel configuration as specified in clause 5.10.2.4 for full decoding mode or clause 5.10.2.6 for core decoding mode. This process is described by the rendering matrix specified in clause 5.10.2.2.

NOTE: In some cases, the surround channels might undergo an additional phase shift operation of 90°.

5.10.2.2 General rendering matrix

The calculation of the output signals out_{ch}, where ch is the corresponding channel of the output channel configuration, from the input signals in_{ch}, where ch is the corresponding channel of the input channel mode, is described through the generalized matrix operation:

$$\begin{bmatrix} out_L \\ out_R \\ out_C \\ out_{Ls} \\ out_{Rs} \\ out_{Lb} \\ out_{Rb} \\ out_{Tfl} \\ out_{Tfr} \\ out_{Tbl} \\ out_{Tbr} \\ out_{Lfe} \\ out_{Tsl} \\ out_{Tsr} \\ out_{Tfc} \\ out_{Tbc} \\ out_{Tc} \\ out_{Lfe2} \\ out_{Bfl} \\ out_{Bfr} \\ out_{Bfc} \\ out_{Cb} \\ out_{Lscr} \\ out_{Rscr} \\ out_{Lw} \\ out_{Rw} \end{bmatrix} = \begin{bmatrix} r_{0,0} & r_{0,1} & \cdots & r_{0,25} \\ r_{1,0} & r_{1,1} & \cdots & r_{1,25} \\ \vdots & \vdots & \cdots & \vdots \\ r_{25,0} & r_{25,1} & \cdots & r_{25,25} \end{bmatrix} \cdot \begin{bmatrix} in_L \\ in_R \\ in_C \\ in_{Ls} \\ in_{Rs} \\ in_{Lb} \\ in_{Rb} \\ in_{Tfl} \\ in_{Tfr} \\ in_{Tbl} \\ in_{Tbr} \\ in_{Lfe} \\ in_{Tsl} \\ in_{Tsr} \\ in_{Tfc} \\ in_{Tbc} \\ in_{Tc} \\ in_{Lfe2} \\ in_{Bfl} \\ in_{Bfr} \\ in_{Bfc} \\ in_{Cb} \\ in_{Lscr} \\ in_{Rscr} \\ in_{Lw} \\ in_{Rw} \end{bmatrix}$$

NOTE: The channel configurations are specified in clause A.3.

If the presentation is an exclusively channel-based presentation: for full decoding mode the input channel mode is indicated by `pres_ch_mode`, for core decoding mode the input channel mode is indicated by `pres_ch_mode_core`. Otherwise (i.e. the `pres_ch_mode`, or `pres_ch_mode_core` respectively, is -1): the input channel mode is indicated by the channel mode of the current substream, the `ch_mode` field, specified in clause 6.3.2.7.2. The coefficients denote $r_{\text{out,in}}$.

Input signals that are not present in the input channel mode should be silenced signals. Output signals that are not present in the output channel configuration should be discarded.

5.10.2.3 Panning of a stereo or mono signal

If a channel audio substream is a dialogue substream and `b_pan_dialog_present` is true, n `pan_dialog` value(s) are present (if the input channel configuration is mono, n = 1; otherwise, n = 2). If a channel audio substream is a dialogue substream and `b_pan_dialog_present` is false, then the `pan_dialog` value(s) shall default as follows: If n=1, the default value shall be `pan_dialog`=0; otherwise, the default values shall be `pan_dialog[0]`=330 and `pan_dialog[1]`=30.

If a channel audio substream is an associated audio substream and the input channel configuration is mono, one `pan_associated` value is present.

Each `pan_dialog` or `pan_associated` value indicates a panning value α_i as specified in ETSI TS 103 190-1 [1], clause 4.3.12.4.9.

If the output channel configuration is mono, the decoder shall ignore present `pan_dialog` or `pan_associated` value(s) and the signal shall be passed through the panning process unmodified.

For all other output channel configurations the decoder shall apply the present `pan_dialog` or `pan_associated` value(s) by performing the following consecutive steps:

1) If the output channel configuration is stereo, the value α shall be derived from α_i as:

$$\alpha = \begin{cases} \alpha_i & if\, \alpha_i \leq 30° \vee \alpha_i \geq 330° \\ 30° & if\, \alpha_i \geq 30° \wedge \alpha_i \leq 150° \\ 330° & if\, \alpha_i \geq 210° \wedge \alpha_i < 330° \\ 180° - \alpha_i & if\, \alpha_i > 150° \wedge \alpha_i \leq 180° \\ 540° - \alpha_i & if\, \alpha_i > 180° \wedge \alpha_i < 210° \end{cases}$$

If the output channel configuration is not stereo, then $\alpha = \alpha_i$.

2) Derive the two adjacent speakers *A* and *B* from the horizontal-plane subset of the output speaker configuration utilizing table 33, in a way that speaker position angles α_A and α_B correspond to:

$$\alpha_A < \alpha < \begin{cases} 360° & if\, \alpha_B = 0° \\ \alpha_B & else \end{cases}$$

3) Calculate:

$$r = \frac{\alpha - \alpha_A}{\alpha_B - \alpha_A}$$

4) Calculate mix matrix coefficients $r_{j,m} = cos(r \times 90°)$ and $r_{k,m} = sin(r \times 90°)$, where *m* is 0 for mono input or in [0, 1] for stereo input. The values for *j* and *k* depend on the output configuration and the corresponding adjacent speakers *A* and *B* as shown in table 33.

EXAMPLE: If *A* is speaker R at -45 degrees, and *B* is speaker Rw at -60 degrees in a 9.X.Y configuration, *j* will be 1 and *k* will be 25. For a stereo input, elements $r_{1,0}$ and $r_{1,1}$ of the rendering matrix (see clause 5.10.2.2) would be populated with the result of the *cos()* calculation, while $r_{25,0}$ and $r_{25,1}$ would be populated with the result of the *sin()* calculation.

Table 33: Horizontal-plane subset of the output speaker configuration and corresponding values for j and k

Output speaker configuration	Speakers	Corresponding values for j and k
9.X.Y	L, R, C, Ls, Rs, Lb, Rb, Lw, Rw	0, 1, 2, 3, 4, 5, 6, 24, 25
7.X.Y	L, R, C, Ls, Rs, Lb, Rb	0, 1, 2, 3, 4, 5, 6
5.X.Y	L, R, C, Ls, Rs	0, 1, 2, 3, 4
2.0	L, R	0, 1

5.10.2.4 Substream downmix or upmix for full decoding

If the embedding process specified in step two in clause 5.10.2.1 is performed, the substream to be processed is present in the presentation channel mode `pres_ch_mode` (specified in clause 6.3.3.1.27); otherwise, the current channel mode is indicated by `ch_mode` (specified in clause 6.3.2.7.2). The decoder shall calculate the signals of the output channel configuration from the signals of the input channel mode, i.e. the current channel mode, utilizing the generalized rendering matrix specified in clause 5.10.2.2.

The decoder shall derive the coefficients $r_{out,in}$ of the rendering matrix from:

- static coefficients specified in this documentation; and
- customized downmix coefficients present in the bitstream.

Derivation of coefficients is specified in clause 5.10.2.5.

Table 34 shows how to derive the matrix coefficients by referencing specific tables in clause 5.10.2.5 depending on the output channel configuration.

Table 34: Mix matrix coefficients for different output channel configurations

Output channel configuration	9.X.4	9.X.2	9.X.0	7.X.4	7.X.2	7.X.0	5.X.4	5.X.2	5.X.0
reference	Table 35	Table 36	Table 37	Table 38	Table 39	Table 40	Table 41	Table 42	Table 43

NOTE: Table A.27 shows which speakers are contained in each channel configuration.

5.10.2.5 Matrix coefficients for channel-based renderer for full decoding

This clause contains one table of coefficients $r_{\mathrm{out,in}}$ for the rendering matrix for each output channel configuration referenced by table 34.

The matrix coefficients depend on the input channel configuration, as specified in clause 5.10.2.4. The configuration is either indicated by the presentation channel mode `pres_ch_mode` or by the `ch_mode`.

NOTE 1: Coefficients have a default value of -∞ dB. The following tables specify non-default coefficients.

NOTE 2: When the LFE channel contains a signal (X = 1), then $r_{11,11}$ = 0 dB; otherwise, $r_{11,11}$ = -∞ dB.

Table 35: Channel rendering coefficients for output to 9.X.4 for full decoding

Input channel configuration	Coefficients
9.X.4	$r_{i,i}$ = 0 dB for i = {0…10, 22, 23}
9.X.2	$r_{i,i}$ = 0 dB for i = {0…6, 22, 23}, $r_{7,12} = r_{9,12} = r_{8,13} = r_{10,13}$ = -3 dB
9.X.0	$r_{i,i}$ = 0 dB for i = {0…6, 22, 23}
7.X.4	$r_{i,i}$ = 0 dB for i = {0…10}
7.X.2	$r_{i,i}$ = 0 dB for i = {0…6}, $r_{7,12} = r_{9,12} = r_{8,13} = r_{10,13}$ = -3 dB
7.X.0	$r_{i,i}$ = 0 dB for i = {0…6}
5.X.4	$r_{i,i}$ = 0 dB for i = {0…4, 7…10}
5.X.2	$r_{i,i}$ = 0 dB for i = {0…4}, $r_{7,12} = r_{9,12} = r_{8,13} = r_{10,13}$ = -3 dB
5.X.0	$r_{i,i}$ = 0 dB for i = {0…4}
3.0.0	$r_{i,i}$ = 0 dB for i = {0…2}
2.0.0	$r_{i,i}$ = 0 dB for i = {0…1}
1.0.0	$r_{i,i}$ = 0 dB for i = 2

Table 36: Channel rendering coefficients for output to 9.X.2 for full decoding

Input channel configuration	Coefficients
9.X.4	$r_{i,i}$ = 0 dB for i = {0…6, 22..23}, $r_{12,7} = r_{12,9} = r_{13,8} = r_{13,10}$ = -3 dB
9.X.2	$r_{i,i}$ = 0 dB for i = {0…6, 12…13, 22…23}
9.X.0	$r_{i,i}$ = 0 dB for i = {0…6, 22…23}
7.X.4	$r_{i,i}$ = 0 dB for i = {0…6}, $r_{12,7} = r_{12,9} = r_{13,8} = r_{13,10}$ = -3 dB
7.X.2	$r_{i,i}$ = 0 dB for i = {0…6, 12…13}
7.X.0	$r_{i,i}$ = 0 dB for i = {0…6}
5.X.4	$r_{i,i}$ = 0 dB for i = {0…4}, $r_{12,7} = r_{12,9} = r_{13,8} = r_{13,10}$ = -3 dB
5.X.2	$r_{i,i}$ = 0 dB for i = {0…4, 12…13}
5.X.0	$r_{i,i}$ = 0 dB for i = {0…4}
3.0.0	$r_{i,i}$ = 0 dB for i = {0…2}
2.0.0	$r_{i,i}$ = 0 dB for i = {0…1}
1.0.0	$r_{i,i}$ = 0 dB for i = 2

Table 37: Channel rendering coefficients for output to 9.X.0 for full decoding

Input channel configuration	Coefficients
9.X.4	$r_{i,i}$=0 dB for i={0…6, 22…23}, $r_{3,7}$=$r_{4,8}$=$r_{3,9}$=$r_{4,10}$=-3 dB
9.X.2	$r_{i,i}$=0 dB for i={0…6, 22…23}, $r_{3,12}$=$r_{4,13}$=-3 dB
9.X.0	$r_{i,i}$=0 dB for i={0…6, 22…23}
7.X.4	$r_{i,i}$=0 dB for i={0…6}, $r_{3,7}$=$r_{4,8}$=$r_{3,9}$=$r_{4,10}$=-3 dB
7.X.2	$r_{i,i}$=0 dB for i={0…6}, $r_{3,12}$=$r_{4,13}$=-3 dB
7.X.0	$r_{i,i}$=0 dB for i={0…6}
5.X.4	$r_{i,i}$=0 dB for i={0…4}, $r_{3,7}$=$r_{4,8}$=$r_{3,9}$=$r_{4,10}$=-3 dB
5.X.2	$r_{i,i}$=0 dB for i={0…4}, $r_{3,12}$=$r_{4,13}$=-3 dB
5.X.0	$r_{i,i}$=0 dB for i={0…4}
3.0.0	$r_{i,i}$=0 dB for i={0…2}
2.0.0	$r_{i,i}$=0 dB for i={0…1}
1.0.0	$r_{i,i}$=0 dB for i=2

Table 38: Channel rendering coefficients for output to 7.X.4 for full decoding

Input channel configuration	Coefficients
9.X.4	$r_{i,i}$ = 0 dB for i = {0…10}, $r_{0,22}$ = $r_{1,23}$ = 0dB
9.X.2	$r_{i,i}$ = 0 dB for i = {0…6}, $r_{0,22}$ = $r_{1,23}$ = 0 dB, $r_{7,12}$ = $r_{9,12}$ = $r_{8,13}$ = $r_{10,13}$ = -3 dB
9.X.0	$r_{i,i}$ = 0 dB for i = {0…6}, $r_{0,22}$ = $r_{1,23}$ = 0 dB
7.X.4	$r_{i,i}$ = 0 dB for i = {0…10}
7.X.2	$r_{i,i}$ = 0 dB for i = {0…6}, $r_{7,12}$ = $r_{9,12}$ = $r_{8,13}$ = $r_{10,13}$ = -3 dB
7.X.0	$r_{i,i}$ = 0 dB for i = {0…6}
5.X.4	$r_{i,i}$ = 0 dB for i = {0…4, 7…10}
5.X.2	$r_{i,i}$ = 0 dB for i = {0…4}, $r_{7,12}$ = $r_{9,12}$ = $r_{8,13}$ = $r_{10,13}$ = -3 dB
5.X.0	$r_{i,i}$ = 0 dB for i = {0…4}
3.0.0	$r_{i,i}$ = 0 dB for i = {0…2}
2.0.0	$r_{i,i}$ = 0 dB for i = {0…1}
1.0.0	$r_{i,i}$ = 0 dB for i = 2

Table 39: Channel rendering coefficients for output to 7.X.2 for full decoding

Input channel configuration	Coefficients
9.X.4	$r_{i,i}$ = 0 dB for i = {0…6}, $r_{0,22}$ = $r_{1,23}$ = *gain_f1*, $r_{2,22}$ = $r_{2,23}$ = *gain_f2*, $r_{12,7}$ = $r_{12,9}$ = $r_{13,8}$ = $r_{13,10}$ = *gain_t1*
9.X.2	$r_{i,i}$ = 0 dB for i = {0…6, 12, 13}, $r_{0,22}$ = $r_{1,23}$ = *gain_f1*, $r_{2,22}$ = $r_{2,23}$ = *gain_f2*
9.X.0	$r_{i,i}$ = 0 dB for i = {0…6}, $r_{0,22}$ = $r_{1,23}$ = 0 dB
7.X.4	$r_{i,i}$ = 0 dB for i = {0…6}, $r_{12,7}$ = $r_{12,9}$ = $r_{13,8}$ = $r_{13,10}$ = *gain_t1*
7.X.2	$r_{i,i}$ = 0 dB for i = {0…6, 12…13}
7.X.0	$r_{i,i}$ = 0 dB for i = {0…6}
5.X.4	$r_{i,i}$ = 0 dB for i = {0…4}, $r_{12,7}$ = $r_{12,9}$ = $r_{13,8}$ = $r_{13,10}$ = -3 dB
5.X.2	$r_{i,i}$ = 0 dB for i = {0…4, 12…13}
5.X.0	$r_{i,i}$ = 0 dB for i = {0…4}
3.0.0	$r_{i,i}$ = 0 dB for i = {0…2}
2.0.0	$r_{i,i}$ = 0 dB for i = {0…1}
1.0.0	$r_{i,i}$ = 0 dB for i = 2

Table 40: Channel rendering coefficients for output to 7.X.0 for full decoding

Input channel configuration	Coefficients
9.X.4	$r_{i,i}$ = 0 dB for i = {0…6}, $r_{0,22} = r_{1,23}$ = *gain_f1*, $r_{2,22} = r_{2,23}$ = *gain_f2*, $r_{0,7} = r_{1,8}$ = *gain_t2a*, $r_{3,7} = r_{4,8}$ = *gain_t2b*, $r_{5,7} = r_{6,8}$ = *gain_t2c*, $r_{0,9} = r_{1,10}$ = *gain_t2d*, $r_{3,9} = r_{4,10}$ = *gain_t2e*, $r_{5,9} = r_{6,10}$ = *gain_t2f*
9.X.2	$r_{i,i}$ = 0 dB for i = {0…6}, $r_{0,22} = r_{1,23}$ = *gain_f1*, $r_{2,22} = r_{2,23}$ = *gain_f2*, $r_{0,12} = r_{1,13}$ = *gain_t2a*, $r_{3,12} = r_{4,13}$ = *gain_t2b*, $r_{5,12} = r_{6,13}$ = *gain_t2c*
9.X.0	$r_{i,i}$ = 0 dB for i = {0…6}, $r_{0,22} = r_{1,23}$ = 0 dB
7.X.4	$r_{i,i}$ = 0 dB for i = {0…6}, $r_{0,7} = r_{1,8}$ = *gain_t2a*, $r_{3,7} = r_{4,8}$ = *gain_t2b*, $r_{5,7} = r_{6,8}$ = *gain_t2c*, $r_{0,9} = r_{1,10}$ = *gain_t2d*, $r_{3,9} = r_{4,10}$ = *gain_t2e*, $r_{5,9} = r_{6,10}$ = *gain_t2f*
7.X.2	$r_{i,i}$ = 0 dB for i = {0…6}, $r_{0,12} = r_{1,13}$ = *gain_t2a*, $r_{3,12} = r_{4,13}$ = *gain_t2b*, $r_{5,12} = r_{6,13}$ = *gain_t2c*
7.X.0	$r_{i,i}$ = 0 dB for i = {0…6}
5.X.4	$r_{i,i}$ = 0 dB for i = {0…4}, $r_{3,7} = r_{4,8} = r_{3,9} = r_{4,10}$ = -3 dB
5.X.2	$r_{i,i}$ = 0 dB for i = {0…4}, $r_{3,12} = r_{4,13}$ = -3 dB
5.X.0	$r_{i,i}$ = 0 dB for i = {0…4}
3.0.0	$r_{i,i}$ = 0 dB for i = {0…2}
2.0.0	$r_{i,i}$ = 0 dB for i = {0…1}
1.0.0	$r_{i,i}$ = 0 dB for i = 2

Table 41: Channel rendering coefficients for output to 5.X.4 for full decoding

Input channel configuration	Coefficients
9.X.4	$r_{i,i}$ = 0 dB for i = {0…2, 7…10}, $r_{0,22} = r_{1,23}$ = *gain_f1*, $r_{2,22} = r_{2,23}$ = *gain_f2*, $r_{3,3} = r_{3,5} = r_{4,4} = r_{4,6}$ = *gain_b*
9.X.2	$r_{i,i}$ = 0 dB for i = {0…2}, $r_{0,22} = r_{1,23}$ = *gain_f1*, $r_{2,22} = r_{2,23}$ = *gain_f2*, $r_{3,3} = r_{3,5} = r_{4,4} = r_{4,6}$ = *gain_b*, $r_{7,12} = r_{9,12} = r_{8,13} = r_{10,13}$ = -3 dB
9.X.0	$r_{i,i}$ = 0 dB for i = {0…2}, $r_{0,22} = r_{1,23}$ = 0 dB, $r_{3,3} = r_{3,5} = r_{4,4} = r_{4,6}$ = -3 dB
7.X.4	$r_{i,i}$ = 0 dB for i = {0…2, 7…10}, $r_{3,3} = r_{3,5} = r_{4,4} = r_{4,6}$ = *gain_b*
7.X.2	$r_{i,i}$ = 0 dB for i = {0…2}, $r_{3,3} = r_{3,5} = r_{4,4} = r_{4,6}$ = *gain_b*, $r_{7,12} = r_{9,12} = r_{8,13} = r_{10,13}$ = -3 dB
7.X.0	$r_{i,i}$ = 0 dB for i = {0…2}, $r_{3,3} = r_{3,5} = r_{4,4} = r_{4,6}$ = -3 dB
5.X.4	$r_{i,i}$ = 0 dB for i = {0…4, 7…10}
5.X.2	$r_{i,i}$ = 0 dB for i = {0…4}, $r_{7,12} = r_{9,12} = r_{8,13} = r_{10,13}$ = -3 dB
5.X.0	$r_{i,i}$ = 0 dB for i = {0…4}
3.0.0	$r_{i,i}$ = 0 dB for i = {0…2}
2.0.0	$r_{i,i}$ = 0 dB for i = {0…1}
1.0.0	$r_{i,i}$ = 0 dB for i = 2

Table 42: Channel rendering coefficients for output to 5.X.2 for full decoding

Input channel configuration	Coefficients
9.X.4	$r_{i,i}$ = 0 dB for i = {0…2}, $r_{0,22} = r_{1,23}$ = *gain_f1*, $r_{2,22} = r_{2,23}$ = *gain_f2*, $r_{3,3} = r_{3,5} = r_{4,4} = r_{4,6}$ = *gain_b*, $r_{12,7} = r_{12,9} = r_{13,8} = r_{13,10}$ = *gain_t1*
9.X.2	$r_{i,i}$ = 0 dB for i = {0…2, 12, 13}, $r_{0,22} = r_{1,23}$ = *gain_f1*, $r_{2,22} = r_{2,23}$ = *gain_f2*, $r_{3,3} = r_{3,5} = r_{4,4} = r_{4,6}$ = *gain_b*
9.X.0	$r_{i,i}$ = 0 dB for i = {0…2}, $r_{0,22} = r_{1,23}$ = 0 dB, $r_{3,3} = r_{3,5} = r_{4,4} = r_{4,6}$ = -3 dB
7.X.4	$r_{i,i}$ = 0 dB for i = {0…2}, $r_{3,3} = r_{3,5} = r_{4,4} = r_{4,6}$ = *gain_b*, $r_{12,7} = r_{12,9} = r_{13,8} = r_{13,10}$ = *gain_t1*
7.X.2	$r_{i,i}$ = 0 dB for i = {0…2, 12…13}, $r_{3,3} = r_{3,5} = r_{4,4} = r_{4,6}$ = *gain_b*
7.X.0	$r_{i,i}$ = 0 dB for i = {0…2}, $r_{3,3} = r_{3,5} = r_{4,4} = r_{4,6}$ = -3 dB
5.X.4	$r_{i,i}$ = 0 dB for i = {0…4}, $r_{12,7} = r_{12,9} = r_{13,8} = r_{13,10}$ = *gain_t1*
5.X.2	$r_{i,i}$ = 0 dB for i = {0…4, 12…13}
5.X.0	$r_{i,i}$ = 0 dB for i = {0…4}
3.0.0	$r_{i,i}$ = 0 dB for i = {0…2}
2.0.0	$r_{i,i}$ = 0 dB for i = {0…1}
1.0.0	$r_{i,i}$ = 0 dB for i = 2

Table 43: Channel rendering coefficients for output to 5.X.0 for full decoding

Input channel configuration	Coefficients
9.X.4	$r_{i,i}$ = 0 dB for i = {0…2}, $r_{0,22}$ = $r_{1,23}$ = *gain_f1*, $r_{2,22}$ = $r_{2,23}$ = *gain_f2*, $r_{3,3}$ = $r_{3,5}$ = $r_{4,4}$ = $r_{4,6}$ = *gain_b*, $r_{0,7}$ = $r_{1,8}$ = *gain_t2a*, $r_{3,7}$ = $r_{4,8}$ = *gain_t2b*, $r_{0,9}$ = $r_{1,10}$ = *gain_t2d*, $r_{3,9}$ = $r_{4,10}$ = *gain_t2e*
9.X.2	$r_{i,i}$ = 0 dB for i = {0…2}, $r_{0,22}$ = $r_{1,23}$ = *gain_f1*, $r_{2,22}$ = $r_{2,23}$ = *gain_f2*, $r_{3,3}$ = $r_{3,5}$ = $r_{4,4}$ = $r_{4,6}$ = *gain_b*, $r_{0,12}$ = $r_{1,13}$ = *gain_t2a*, $r_{3,12}$ = $r_{4,13}$ = *gain_t2b*
9.X.0	$r_{i,i}$ = 0 dB for i = {0…2}, $r_{0,22}$ = $r_{1,23}$ = 0 dB, $r_{3,3}$ = $r_{3,5}$ = $r_{4,4}$ = $r_{4,6}$ = -3 dB
7.X.4	$r_{i,i}$ = 0 dB for i = {0…2}, $r_{3,3}$ = $r_{3,5}$ = $r_{4,4}$ = $r_{4,6}$ = *gain_b*, $r_{0,7}$ = $r_{1,8}$ = *gain_t2a*, $r_{3,7}$ = $r_{4,8}$ = *gain_t2b*, $r_{0,9}$ = $r_{1,10}$ = *gain_t2d*, $r_{3,9}$ = $r_{4,10}$ = *gain_t2e*
7.X.2	$r_{i,i}$ = 0 dB for i = {0…2}, $r_{3,3}$ = $r_{3,5}$ = $r_{4,4}$ = $r_{4,6}$ = *gain_b*, $r_{0,12}$ = $r_{1,13}$ = *gain_t2a*, $r_{3,12}$ = $r_{4,13}$ = *gain_t2b*
7.X.0	$r_{i,i}$ = 0 dB for i = {0…2}, $r_{3,3}$ = $r_{3,5}$ = $r_{4,4}$ = $r_{4,6}$ = -3 dB
5.X.4	$r_{i,i}$ = 0 dB for i = {0…4}, $r_{0,7}$ = $r_{1,8}$ = *gain_t2a*, $r_{3,7}$ = $r_{4,8}$ = *gain_t2b*, $r_{0,9}$ = $r_{1,10}$ = *gain_t2d*, $r_{3,9}$ = $r_{4,10}$ = *gain_t2e*
5.X.2	$r_{i,i}$ = 0 dB for i = {0…4}, $r_{0,12}$ = $r_{1,13}$ = *gain_t2a*, $r_{3,12}$ = $r_{4,13}$ = *gain_t2b*
5.X.0	$r_{i,i}$ = 0 dB for i = {0…4}
3.0.0	$r_{i,i}$ = 0 dB for i = {0…2}
2.0.0	$r_{i,i}$ = 0 dB for i = {0…1}
1.0.0	$r_{i,i}$ = 0 dB for i = 2

5.10.2.6 Substream downmix or upmix for core decoding

If the embedding process specified in step two in clause 5.10.2.1 is performed, the substream to be processed is present in the presentation channel mode for core decoding `pres_ch_mode_core` (specified in clause 6.3.3.1.28); otherwise, the current channel mode is indicated by `ch_mode` (specified in clause 6.3.2.7.2). The decoder shall calculate the signals of the output channel configuration from the signals of the input channel mode, i.e. the current channel mode, utilizing the generalized rendering matrix specified in clause 5.10.2.2.

The decoder shall derive the coefficients $r_{\text{out,in}}$ of the rendering matrix from:

- static coefficients specified in the present documentation; and
- customized downmix coefficients present in the bitstream.

Derivation of coefficients specified in clause 5.10.2.7.

Table 44 shows how to derive the matrix coefficients by referencing specific tables in clause 5.10.2.7 depending on the output channel configuration.

Table 44: Mix matrix coefficients for different output channel configurations in core decoding mode

Output channel configuration	Location of coefficient definition
5.X.2	Table 45
5.X.0	Table 46

5.10.2.7 Matrix coefficients for channel-based renderer for core decoding

This clause contains one table of coefficients $r_{\text{out,in}}$ for the rendering matrix for each output channel configuration referenced by table 44.

The matrix coefficients depend on the input channel configuration, as specified in clause 5.10.2.6 either indicated by the presentation channel mode for core decoding `pres_ch_mode_core` or by the `ch_mode`.

NOTE 1: Coefficients have a default value of $-\infty$ dB. The following tables specify non-default coefficients.

NOTE 2: When the LFE channel contains a signal (X = 1), then $r_{11,11}$ = 0 dB; otherwise, $r_{11,11}$ = $-\infty$ dB.

NOTE 3: The coefficients depend on additional conditions specified in the second column.

Table 45: Channel rendering coefficients for output to 5.X.2 for core decoding

Input channel configuration	Condition	Coefficients
9.X.X, 7.X.X	Always	$r_{i,i}$ = 0 dB for i = {0…2}
	`top_channels_present` = 3	$r_{12,12} = r_{13,13}$ = *gain_t1*+3 dB
	`top_channels_present` = {1,2}	$r_{12,12} = r_{13,13}$ = +3 dB
	`b_4_back_channels_present` = 1	$r_{3,3} = r_{4,4}$ = *gain_b*+3 dB
	`b_4_back_channels_present` = 0	$r_{3,3} = r_{4,4}$ = +3 dB
5.X.0	Always	$r_{i,i}$ = 0 dB for i = {0…4}
3.0.0	Always	$r_{i,i}$ = 0 dB for i = 0…2}
2.0.0	Always	$r_{i,i}$ = 0 dB for i = {0…1}
1.0.0	Always	$r_{i,i}$ = 0 dB for i = 2

Table 46: Channel rendering coefficients for output to 5.X.0 for core decoding

Input channel configuration	Condition	Coefficients
9.X.X, 7.X.X	Always	$r_{i,i}$ = 0 dB for i = {0…2}
	`top_channels_present`={1…3}	$r_{0,12} = r_{1,13}$ = *gain_t2a*+3 dB, $r_{3,12} = r_{4,13}$ = *gain_t2b*+3 dB
	`b_4_back_channels_present`=1	$r_{3,3} = r_{4,4}$ = *gain_b*+3 dB
	`b_4_back_channels_present`=0	$r_{3,3} = r_{4,4}$ = +3 dB
5.X.0	Always	$r_{i,i}$ = 0 dB for i = {0…4}
3.0.0	Always	$r_{i,i}$ = 0 dB for i = {0…2}
2.0.0	Always	$r_{i,i}$ = 0 dB for i = {0…1}
1.0.0	Always	$r_{i,i}$ = 0 dB for i = 2

5.10.3 Intermediate spatial format rendering

5.10.3.1 Introduction

Spatial audio scenes may be represented using a variety of multichannel soundfield formats, including object formats or traditional multichannel formats.

A spatial audio scene is comprised at the source of many individual sound objects, potentially with different radiation characteristics, which may be encoded and transmitted in some interstitial format for eventual playback by rendering to a given speaker array or other audio output device at the sink.

The present document adopts the term ISF to describe any such interstitial format into which object audio may be coded, and from which the same audio may be rendered to any given speaker array. An ISF preserves the flexibility to "decode" this soundfield with minimal spatial distortion to almost any three-dimensional speaker array.

EXAMPLE: Spherical harmonics can be used to build an ISF, transmitting the soundfield using spherical harmonic components.

The present document specifies a new class of ISF that provides equivalent spatial resolution using fewer audio signals, by assuming the placement of playback speakers in relatively few horizontally aligned layers. The specified ISF represents the soundfield as several "stacked rings" of signals, with each set similar to a circular harmonic component representation.

5.10.3.2 Conventions

The stacked ring ISF format is denoted by SR*M.U.L.Z*, using four numbers to represent the number of ISF object audio essences in the middle, upper, lower, and zenith rings as shown in figure 12.

EXAMPLE: SR*9.5.0.1* represents a signal with 9 signals for the middle ring, 5 for the upper ring, 1 for the zenith, and none for the lower ring.

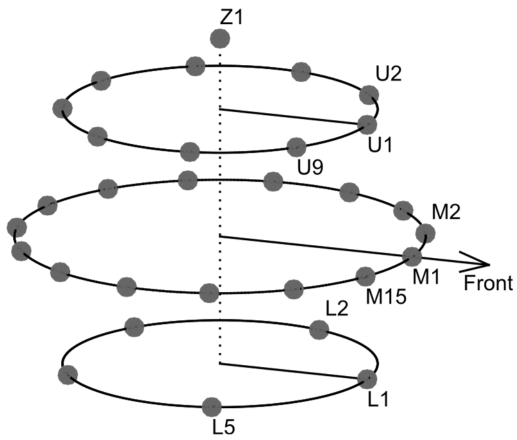


Figure 12: Schematic representation of the ISF stacked ring format

5.10.3.3 Interface

5.10.3.3.1 Inputs

$x_{1,\ldots,N\_in}$ *N_in* decoded ISF object audio essences.

5.10.3.3.2 Outputs

$y_{1,\ldots,N\_out}$ *N_out* speaker feeds output from the ISF tool.

N_out is the number of speakers in the output channel configuration as defined in clause 5.10.2.5.

5.10.3.3.3 Controls

`isf_config` The configuration of ISF object audio essences as defined in clause 5.10.3.2.

5.10.3.4 Processing

The ISF rendering algorithm maps ISF object audio essences to speaker feeds with coefficients configured by the output channel configuration.

To process ISF object audio essences, the decoder shall:

1) Select the matrix **M** for processing from clause A.2.1. The selection is dependent on the output channel configuration.

2) Assign the decoded object audio essences successively to elements of column vector $t = [M_1\ldots, U_1\ldots, L_1\ldots, Z]$.

NOTE 1: The exact layout of *t* is determined by `isf_config` (see table 61). LFE channels stays unassigned, i.e. they bypass the ISF tool.

3) Transform the signal into speaker feeds by $y = M \times t$.

NOTE 2: The ISF tool is operated at the output channel configuration (see clause 4.8.4).

5.11 Accurate frame rate control

This tool provides accurate control over media timing when frame lengths are fractional.

As specified in ETSI TS 103 190-1 [1], clause 4.3.3.2.6 AC-4 offers control over audio coding frame rate.

Frame rates in [29,97; 59,94; 119,88] frames per second result in the nominal fractional decoded frame lengths of 1 601,6, 800,8, and 400,4 samples, respectively, at the sample rate converter output (see ETSI TS 103 190-1 [1], clause 6.2.15). In practice, the sample rate converter will not return fractional samples, but instead frames of integer lengths straddling the fractional frame length.

EXAMPLE: At a frame rate of $29{,}97[\text{Hz}] = 30 \times \frac{1000}{1001}[\text{Hz}]$, each frame measures $\frac{48000}{30} \times \frac{1001}{1000} = 8 \times \frac{1001}{5}$ samples. A standard sample rate converter will return frames of 1 601 and 1 602 samples in a sequence repeating after five frames, corresponding to five distinct phases of the resampler.

If the phase of the sample rate converter is locked to the stream, it will produce the same number of output sample from a frame independent of where decoding started. This is achieved by locking the phase of the sample rate converter to φ_t according to table 47, with:

$$\phi_t = \begin{cases} \text{cnt}\,mod 5 & if\,\text{cnt} \neq 0 \\ (\phi_{t\text{-}1} + 1) mod 5 & if\,(\text{cnt} = 0) \wedge (\text{this is not the first frame}) \\ 0 & else \end{cases}$$

where *cnt* is set to the `sequence_counter`.

The decoder shall link the number of output samples per frame to the `sequence_counter` according to the equation above and table 47.

When a source change is detected according to ETSI TS 103 190-1 [1], clause 4.3.3.2.2, and it results in a different output sequence or a jump in the phase of the current output sequence this change shall only be applied at the time the first frame of the new source is returned. In other words, the change in resampler phase sequence shall be delayed with the signal so that the change only becomes active once the first sample from the new output sequence reaches the resampler output.

Table 47: Mapping sequence index to number of resampler output samples

Frame rate [Hz]	Exact frame length [samples]	φ_t	Internal block size	Number of output samples	Remainder
29,97	1 601,6	0	1 536	1 601	0,6
		1	1 536	1 602	0,2
		2	1 536	1 601	0,8
		3	1 536	1 602	0,4
		4	1 536	1 602	0
59,94	800,8	0	768	800	0,8
		1	768	801	0,6
		2	768	801	0,4
		3	768	801	0,2
		4	768	801	0
119,88	400,4	0	384	400	0,4
		1	384	400	0,8
		2	384	401	0,2
		3	384	400	0,6
		4	384	401	0

6 Bitstream syntax

6.1 Introduction

The present document re-uses syntactic elements specified in ETSI TS 103 190-1 [1], clause 4.2. Most syntactic elements are used unchanged as indicated in table 48. Some syntactic elements are amended as indicated in table 49. This clause specifies amended and newly defined syntactic elements.

Table 48: Syntactic elements specified in ETSI TS 103 190-1 [1]

Syntactic element	Location in ETSI TS 103 190-1 [1]
raw_ac4_frame	Clause 4.2.1 (table 2)
variable_bits	Clause 4.2.2 (table 3)
presentation_version	Clause 4.2.3.3 (table 6)
frame_rate_multiply_info	Clause 4.2.3.4 (table 7)
emdf_info	Clause 4.2.3.5 (table 8)
ac4_substream_info	Clause 4.2.3.6 (table 9)
content_type	Clause 4.2.3.7 (table 10)
emdf_payloads_substream_info	Clause 4.2.3.10 (table 13)
substream_index_table	Clause 4.2.3.11 (table 14)
ac4_hsf_ext_substream	Clause 4.2.4.2 (table 17)
emdf_payloads_substream	Clause 4.2.4.3 (table 18)
single_channel_element	Clause 4.2.6.1 (table 20)
mono_data	Clause 4.2.6.2 (table 21)
channel_pair_element	Clause 4.2.6.3 (table 22)
stereo_data	Clause 4.2.6.4 (table 23)
3_0_channel_element	Clause 4.2.6.5 (table 24)
5_X_channel_element	Clause 4.2.6.6 (table 25)
two_channel_data	Clause 4.2.6.7 (table 26)
three_channel_data	Clause 4.2.6.8 (table 27)
four_channel_data	Clause 4.2.6.9 (table 28)
five_channel_data	Clause 4.2.6.10 (table 29)
three_channel_info	Clause 4.2.6.11 (table 30)
four_channel_info	Clause 4.2.6.12 (table 31)
five_channel_info	Clause 4.2.6.13 (table 32)
7_X_channel_element	Clause 4.2.6.14 (table 33)
sf_info	Clause 4.2.7.1 (table 34)
sf_info_lfe	Clause 4.2.7.2 (table 35)
sf_data	Clause 4.2.7.3 (table 36)
asf_transform_info	Clause 4.2.8.1 (table 37)
asf_psy_info	Clause 4.2.8.2 (table 38)
asf_section_data	Clause 4.2.8.3 (table 39)
asf_spectral_data	Clause 4.2.8.4 (table 40)
asf_scalefac_data	Clause 4.2.8.5 (table 41)
asf_snf_data	Clause 4.2.8.6 (table 42)
ssf_data	Clause 4.2.9.1 (table 43)
ssf_granule	Clause 4.2.9.2 (table 44)
ssf_st_data	Clause 4.2.9.3 (table 45)
ssf_ac_data	Clause 4.2.9.4 (table 46)
chparam_info	Clause 4.2.10.1 (table 47)
sap_data	Clause 4.2.10.2 (table 48)
companding_control	Clause 4.2.11 (table 49)
aspx_config	Clause 4.2.12.1 (table 50)
aspx_data_1ch	Clause 4.2.12.2 (table 51)
aspx_data_2ch	Clause 4.2.12.3 (table 52)
aspx_framing	Clause 4.2.12.4 (table 53)
aspx_delta_dir	Clause 4.2.12.5 (table 54)
aspx_hfgen_iwc_1ch	Clause 4.2.12.6 (table 55)
aspx_hfgen_iwc_2ch	Clause 4.2.12.7 (table 56)
aspx_ec_data	Clause 4.2.12.8 (table 57)
aspx_huff_data	Clause 4.2.12.9 (table 58)

Syntactic element	Location in ETSI TS 103 190-1 [1]
acpl_config_1ch	Clause 4.2.13.1 (table 59)
acpl_config_2ch	Clause 4.2.13.2 (table 60)
acpl_data_1ch	Clause 4.2.13.3 (table 61)
acpl_data_2ch	Clause 4.2.13.4 (table 62)
acpl_framing_data	Clause 4.2.13.5 (table 63)
acpl_ec_data	Clause 4.2.13.6 (table 64)
acpl_huff_data	Clause 4.2.13.7 (table 65)
drc_frame	Clause 4.2.14.5 (table 70)
drc_config	Clause 4.2.14.6 (table 71)
drc_decoder_mode_config	Clause 4.2.14.7 (table 72)
drc_compression_curve	Clause 4.2.14.8 (table 73)
drc_data	Clause 4.2.14.9 (table 74)
drc_gains	Clause 4.2.14.10 (table 75)
de_config	Clause 4.2.14.12 (table 77)
emdf_payload_config	Clause 4.2.14.14 (table 79)
emdf_protection	Clause 4.2.14.15 (table 80)

Table 49: Amended syntactic elements specified in ETSI TS 103 190-1 [1]

Syntactic element	Location in ETSI TS 103 190-1 [1]	Location in the present document
ac4_toc	Clause 4.2.3.1 (table 4)	Clause 6.2.1.1
ac4_presentation_info	Clause 4.2.3.2 (table 5)	Clause 6.2.1.2
presentation_config_ext_info	Clause 4.2.3.8 (table 11)	Clause 6.2.1.5
ac4_hsf_ext_substream_info	Clause 4.2.3.9 (table 12)	Clause 6.2.1.14
ac4_substream	Clause 4.2.4.1 (table 16)	Clause 6.2.2.2
audio_data	Clause 4.2.5 (table 19)	Clause 6.2.3.1
metadata	Clause 4.2.14.1 (table 66)	Clause 6.2.7.1
basic_metadata	Clause 4.2.14.2 (table 67)	Clause 6.2.7.2
further_loudness_info	Clause 4.2.14.3 (table 68)	Clause 6.2.7.3
extended_metadata	Clause 4.2.14.4 (table 69)	Clause 6.2.7.4
dialog_enhancement	Clause 4.2.14.11 (table 76)	Clause 6.2.7.5
de_data	Clause 4.2.14.13 (table 78)	Clause 6.2.7.6

6.2 Syntax specification

6.2.1 AC-4 frame info

6.2.1.1 ac4_toc

Syntax	No of bits

```
ac4_toc()
{
  bitstream_version; ...................................................................... 2
  if (bitstream_version == 3) {
    bitstream_version += variable_bits(2);
  }
  sequence_counter; ...................................................................... 10
  b_wait_frames; .......................................................................... 1
  if (b_wait_frames) {
    wait_frames; ............................................................................ 3
    if (wait_frames > 0) {
      br_code; ................................................................................ 2
    }
  }
  fs_index; ................................................................................... 1
  frame_rate_index; ...................................................................... 4
  b_iframe_global; ........................................................................ 1
  b_single_presentation; ............................................................... 1
  if (b_single_presentation) {
    n_presentations = 1;
  }
  else {
```

Syntax	No of bits

```
    b_more_presentations; ............................................ 1
    if (b_more_presentations) {
      n_presentations = variable_bits(2) + 2;
    }
    else {
      n_presentations = 0;
    }
  }
  payload_base = 0;
  b_payload_base; .................................................... 1
  if (b_payload_base) {
    payload_base_minus1; ............................................. 5
    payload_base = payload_base_minus1 + 1;
    if (payload_base == 0x20) {
      payload_base += variable_bits(3);
    }
  }
  if (bitstream_version <= 1) {
    for (i = 0; i < n_presentations; i++) {
      ac4_presentation_info();
    }
  }
  else {
    b_program_id; .................................................... 1
    if (b_program_id) {
      short_program_id; .............................................. 16
      b_program_uuid_present; ........................................ 1
      if (b_program_uuid_present) {
        program_uuid; ................................................ 16 * 8
      }
    }
    for (i = 0; i < n_presentations; i++) {
      ac4_presentation_v1_info();
    }
    for (j = 0; j < total_n_substream_groups; j++) {
      ac4_substream_group_info();
    }
  }
  substream_index_table();
  byte_align; ........................................................ 0…7
}
```

6.2.1.2 ac4_presentation_info

Syntax	No of bits

```
ac4_presentation_info()
{
  b_single_substream; ................................................ 1
  if (b_single_substream != 1) {
    presentation_config; ............................................. 3
    if (presentation_config == 7) {
      presentation_config += variable_bits(2);
    }
  }
  presentation_version();
  if (b_single_substream != 1 and presentation_config == 6) {
    b_add_emdf_substreams = 1;
  }
  else {
    md_compat; ....................................................... 3
    b_presentation_id; ............................................... 1
    if (b_presentation_id) {
      presentation_id = variable_bits(2);
    }
    frame_rate_multiply_info();
    emdf_info();
    if (b_single_substream == 1) {
      ac4_substream_info();
    }
    else {
      b_hsf_ext; ..................................................... 1
      switch (presentation_config) {
        case 0:
          ac4_substream_info();
```

Syntax	No of bits

```
          if (b_hsf_ext) {
            ac4_hsf_ext_substream_info(1);
          }
          ac4_substream_info();
          break;
        case 1:
          ac4_substream_info();
          if (b_hsf_ext) {
            ac4_hsf_ext_substream_info(1);
          }
          ac4_substream_info();
          break;
        case 2:
          ac4_substream_info();
          if (b_hsf_ext) {
            ac4_hsf_ext_substream_info(1);
          }
          ac4_substream_info();
          break;
        case 3:
          ac4_substream_info();
          if (b_hsf_ext) {
            ac4_hsf_ext_substream_info(1);
          }
          ac4_substream_info();
          ac4_substream_info();
          break;
        case 4:
          ac4_substream_info();
          if (b_hsf_ext) {
            ac4_hsf_ext_substream_info(1);
          }
          ac4_substream_info();
          ac4_substream_info();
          break;
        case 5:
          ac4_substream_info();
          if (b_hsf_ext) {
            ac4_hsf_ext_substream_info(1);
          }
          break;
        default:
          presentation_config_ext_info();
          break;
      }
    }
    b_pre_virtualized; ............................................................... 1
    b_add_emdf_substreams; ........................................................... 1
  }
  if (b_add_emdf_substreams) {
    n_add_emdf_substreams; ........................................................... 2
    if (n_add_emdf_substreams == 0) {
      n_add_emdf_substreams = variable_bits(2) + 4;
    }
    for (i = 0; i < n_add_emdf_substreams; i++) {
      emdf_info();
    }
  }
}
```

6.2.1.3 ac4_presentation_v1_info

Syntax	No of bits

```
ac4_presentation_v1_info()
{
  b_single_substream_group; ........................................................ 1
  if (b_single_substream_group != 1) {
    presentation_config; ........................................................... 3
    if (presentation_config == 7) {
      presentation_config += variable_bits(2);
    }
  }
  if (bitstream_version != 1) {
    presentation_version();
```

Syntax	No of bits

```
  }
  if (b_single_substream_group != 1 and presentation_config == 6) {
    b_add_emdf_substreams = 1;
  }
  else {
    if (bitstream_version != 1) {
      md_compat; ........................................................ 3
    }
    b_presentation_id; .................................................. 1
    if (b_presentation_id) {
      presentation_id = variable_bits(2);
    }
    frame_rate_multiply_info();
    frame_rate_fractions_info();
    emdf_info();
    b_presentation_filter; .............................................. 1
    if (b_presentation_filter) {
      b_enable_presentation; ............................................ 1
    }
    if (b_single_substream_group == 1) {
      ac4_sgi_specifier();
      n_substream_groups = 1;
    }
    else {
      b_multi_pid; ...................................................... 1
      switch (presentation_config) {
        case 0:
          /* Music and Effects + Dialogue */
          ac4_sgi_specifier();
          ac4_sgi_specifier();
          n_substream_groups = 2;
          break;
        case 1:
          /* Main + DE */
          ac4_sgi_specifier();
          ac4_sgi_specifier();
          n_substream_groups = 1;
          break;
        case 2:
          /* Main + Associated Audio */
          ac4_sgi_specifier();
          ac4_sgi_specifier();
          n_substream_groups = 2;
          break;
        case 3:
          /* Music and Effects + Dialogue + Associated Audio */
          ac4_sgi_specifier();
          ac4_sgi_specifier();
          ac4_sgi_specifier();
          n_substream_groups = 3;
          break;
        case 4:
          /* Main + DE + Associated Audio */
          ac4_sgi_specifier();
          ac4_sgi_specifier();
          ac4_sgi_specifier();
          n_substream_groups = 2;
          break;
        case 5:
          /* Arbitrary number of roles and substream groups */
          n_substream_groups_minus2; .................................... 2
          n_substream_groups = n_substream_groups_minus2 + 2;
          if (n_substream_groups == 5) {
            n_substream_groups += variable_bits(2);
          }
          for (sg = 0; sg < n_substream_groups; sg++) {
            ac4_sgi_specifier();
          }
          break;
        default:
          /* EMDF and other data */
          presentation_config_ext_info();
          break;
      }
    }
    b_pre_virtualized; .................................................. 1
    b_add_emdf_substreams; .............................................. 1
```

Syntax	No of bits
`    ac4_presentation_substream_info();`	
`  }`	
`  if (b_add_emdf_substreams) {`	
`    **n_add_emdf_substreams**;`	2
`    if (n_add_emdf_substreams == 0) {`	
`      n_add_emdf_substreams = variable_bits(2) + 4;`	
`    }`	
`    for (i = 0; i < n_add_emdf_substreams; i++) {`	
`      emdf_info();`	
`    }`	
`  }`	
`}`	

6.2.1.4 frame_rate_fractions_info

```
Syntax                                                              No of bits
frame_rate_fractions_info()
{
  frame_rate_fraction = 1;
  if (frame_rate_index in [5, 6, 7, 8, 9]) {
    if (frame_rate_factor == 1) {
      b_frame_rate_fraction; ............................................ 1
      if (b_frame_rate_fraction == 1) {
        frame_rate_fraction = 2;
      }
    }
  }
  if (frame_rate_index in [10, 11, 12]) {
    b_frame_rate_fraction; .............................................. 1
    if (b_frame_rate_fraction == 1) {
      b_frame_rate_fraction_is_4; ....................................... 1
      if (b_frame_rate_fraction_is_4 == 1) {
        frame_rate_fraction = 4;
      }
      else {
        frame_rate_fraction = 2;
      }
    }
  }
}
```

6.2.1.5 presentation_config_ext_info

```
Syntax                                                              No of bits
presentation_config_ext_info()
{
  n_skip_bytes; ......................................................... 5
  b_more_skip_bytes; .................................................... 1
  if (b_more_skip_bytes) {
    n_skip_bytes += variable_bits(2) << 5;
  }
  if (bitstream_version == 1 and presentation_config == 7) {
    n_bits_read = ac4_presentation_v1_info();
    if (n_bits_read % 8) {
      n_skip_bits = 8 - (n_bits_read % 8);
      reserved; ................................................ n_skip_bits
      n_bits_read += n_skip_bits;
    }
    n_skip_bytes = n_skip_bytes - (n_bits_read / 8);
  }
  for (i = 0; i < n_skip_bytes; i++) {
    reserved; ........................................................... 8
  }
}
```

6.2.1.6 ac4_substream_group_info

Syntax	No of bits
ac4_substream_group_info()	
{	
b_substreams_present;	1
b_hsf_ext;	1
b_single_substream;	1
if (b_single_substream) {	
n_lf_substreams = 1;	
}	
else {	
n_lf_substreams_minus2;	2
n_lf_substreams = n_lf_substreams_minus2 + 2;	
if (n_lf_substreams == 5) {	
n_lf_substreams += variable_bits(2);	
}	
}	
b_channel_coded;	1
if (b_channel_coded) {	
b_ajoc = 0;	
for (sus = 0; sus < n_lf_substreams; sus++) {	
if (bitstream_version == 1) {	
sus_ver;	1
}	
else {	
sus_ver = 1;	
}	
ac4_substream_info_chan(b_substreams_present);	
if (b_hsf_ext) {	
ac4_hsf_ext_substream_info(b_substreams_present);	
}	
}	
}	
else {	
b_oamd_substream;	1
if (b_oamd_substream) {	
oamd_substream_info(b_substreams_present);	
}	
for (sus = 0; sus < n_lf_substreams; sus++) {	
b_ajoc;	1
if (b_ajoc) {	
ac4_substream_info_ajoc(b_substreams_present);	
if (b_hsf_ext) {	
ac4_hsf_ext_substream_info(b_substreams_present);	
}	
}	
else {	
ac4_substream_info_obj(b_substreams_present);	
if (b_hsf_ext) {	
ac4_hsf_ext_substream_info(b_substreams_present);	
}	
}	
}	
}	
b_content_type;	1
if (b_content_type) {	
content_type();	
}	
}	

6.2.1.7 ac4_sgi_specifier

Syntax	No of bits
ac4_sgi_specifier()	
{	
if (bitstream_version == 1) {	
ac4_substream_group_info();	
}	
else {	
group_index;	3
if (group_index == 7) {	
group_index += variable_bits(2);	
}	

Syntax	No of bits

``   return group_index;
  }
}```	

```
    return group_index;
  }
}
```

6.2.1.8 ac4_substream_info_chan

Syntax	No of bits

```
ac4_substream_info_chan(b_substreams_present)
{
  channel_mode; ............................................................ 1/2/4/7/8/9
  if (channel_mode == 0b111111111) {
    channel_mode += variable_bits(2);
  }
  if (channel_mode in [0b11111100, 0b11111101, 0b111111100, 0b111111101]) {
    b_4_back_channels_present; ............................................. 1
    b_centre_present; ...................................................... 1
    top_channels_present; .................................................. 2
  }
  if (fs_index == 1) {
    b_sf_multiplier; ....................................................... 1
    if (b_sf_multiplier) {
      sf_multiplier; ....................................................... 1
    }
  }
  b_bitrate_info; .......................................................... 1
  if (b_bitrate_info) {
    bitrate_indicator; ..................................................... 3/5
  }
  if (channel_mode in [0b1111010, 0b1111011, 0b1111100, 0b1111101]) {
    add_ch_base; ........................................................... 1
  }
  for (i = 0; i < frame_rate_factor; i++) {
    b_audio_ndot; .......................................................... 1
  }
  if (b_substreams_present == 1) {
    substream_index; ....................................................... 2
    if (substream_index == 3) {
      substream_index += variable_bits(2);
    }
  }
}
```

6.2.1.9 ac4_substream_info_ajoc

Syntax	No of bits

```
ac4_substream_info_ajoc(b_substreams_present)
{
  b_lfe; ................................................................... 1
  b_static_dmx; ............................................................ 1
  if (b_static_dmx) {
    n_fullband_dmx_signals = 5;
  }
  else {
    n_fullband_dmx_signals_minus1; ......................................... 4
    n_fullband_dmx_signals = n_fullband_dmx_signals_minus1 + 1;
    bed_dyn_obj_assignment(n_fullband_dmx_signals);
  }
  b_oamd_common_data_present; .............................................. 1
  if (b_oamd_common_data_present) {
    oamd_common_data();
  }
  n_fullband_upmix_signals_minus1; ......................................... 4
  n_fullband_upmix_signals = n_fullband_upmix_signals_minus1 + 1;
  if (n_fullband_upmix_signals == 16) {
    n_fullband_upmix_signals += variable_bits(3);
  }
  bed_dyn_obj_assignment(n_fullband_upmix_signals);
  if (fs_index == 1) {
    b_sf_multiplier; ....................................................... 1
    if (b_sf_multiplier) {
```

Syntax	No of bits

```
      sf_multiplier; ...................................................................... 1
    }
  }
  b_bitrate_info; ......................................................................... 1
  if (b_bitrate_info) {
    bitrate_indicator; .................................................................. 3/5
  }
  for (i = 0; i < frame_rate_factor; i++) {
    b_audio_ndot; ......................................................................... 1
  }
  if (b_substreams_present == 1) {
    substream_index; ...................................................................... 2
    if (substream_index == 3) {
      substream_index += variable_bits(2);
    }
  }
  sus_ver = 1;
}
```

6.2.1.10 bed_dyn_obj_assignment

Syntax	No of bits

```
bed_dyn_obj_assignment(n_signals)
{
  /* This function provides an object count and type arrays in n_objs, obj_type[], b_lfe[] and
b_ajoc_coded[] */
  /* obj_type = {BED, DYN, ISF} */
  n_objs = 0;
  b_dyn_objects_only; ..................................................................... 1
  if (b_dyn_objects_only == 0) {
    b_isf; ................................................................................ 1
    if (b_isf) {
      isf_config; ......................................................................... 3
      n_isf = [4, 8, 10, 14, 15, 30][isf_config];
      for (i = 0; i < n_isf; i++) {
        obj_type[n_objs] = ISF;
        b_lfe[n_objs] = 0;
        b_ajoc_coded[n_objs] = 1;
        n_objs += 1;
      }
    }
    else {
      b_ch_assign_code; ................................................................... 1
      if (b_ch_assign_code) {
        bed_chan_assign_code; ............................................................. 3
        for (i = 0; i < [2, 3, 5, 7, 9, 7, 9, 11][bed_chan_assign_code]; i++) {
          obj_type[n_objs] = BED;
          b_lfe[n_objs] = 0;
          b_ajoc_coded[n_objs] = 1;
          n_objs += 1;
        }
      }
      else {
        b_channel_assignment_flags_present; ............................................... 1
        if (b_channel_assignment_flags_present) {
          b_nonstd_bed_channel_assignment_flags_present; .................................. 1
          if (b_nonstd_bed_channel_assignment_flags_present) {
            nonstd_bed_channel_assignment_flag[]; ........................................ 17
            for (i = 0; i < 17; i++) {
              if (nonstd_bed_channel_assignment_flag[16 - i] != 0) {
                if (i != 3 and i != 16) {
                  obj_type[n_objs] = BED;
                  b_lfe[n_objs] = 0;
                  b_ajoc_coded[n_objs] = 1;
                  n_objs += 1;
                }
              }
            }
          }
          else {
            std_bed_channel_assignment_flag[]; ........................................... 10
            for (i = 0; i < 10; i++) {
              if (std_bed_channel_assignment_flag[9 - i] != 0) {
                if (i != 2 and i != 9) {
```

```
Syntax                                                                                No of bits
                  for (j = 0; j < [2, 1, 1, 2, 2, 2, 2, 2, 2, 1][i]; j++) {
                    obj_type[n_objs] = BED;
                    b_lfe[n_objs] = 0;
                    b_ajoc_coded[n_objs] = 1;
                    n_objs += 1;
                  }
                }
              }
            }
          }
        }
        else {
          if (n_signals > 1) {
            bed_ch_bits = ceil(log2(n_signals));
            n_bed_signals_minus1; .................................................. bed_ch_bits
            n_bed_signals = n_bed_signals_minus1 + 1;
          }
          else {
            n_bed_signals = 1;
          }
          for (b = 0; b < n_bed_signals; b++) {
            nonstd_bed_channel_assignment; ..................................................... 4
            if (nonstd_bed_channel_assignment != 3) {
              obj_type[n_objs] = BED;
              b_lfe[n_objs] = 0;
              b_ajoc_coded[n_objs] = 1;
              n_objs += 1;
            }
          }
        }
      }
    }
  }
}
```

6.2.1.11 ac4_substream_info_obj

```
Syntax                                                                                No of bits
ac4_substream_info_obj(b_substreams_present)
{
  /* This function provides an object count and type arrays in n_objs, obj_type[], b_lfe[] and
b_ajoc_coded[] */
  /* obj_type = {BED, DYN, ISF} */
  n_objs = 0;
  n_objects_code; ........................................................................ 3
  num_objects = [0, 1, 2, 3, 5, 7][n_objects_code];
  b_dynamic_objects; ..................................................................... 1
  if (b_dynamic_objects) {
    b_lfe; ............................................................................... 1
    for (i = 0; i < num_objects; i++) {
      if (b_lfe and i == 0) {
        obj_type[n_objs] = BED;
        b_lfe[n_objs] = 1;
      }
      else {
        obj_type[n_objs] = DYN;
        b_lfe[n_objs] = 0;
      }
      b_ajoc_coded[n_objs] = 0;
      n_objs += 1;
    }
  }
  else {
    b_bed_objects; ....................................................................... 1
    if (b_bed_objects) {
      b_bed_start; ....................................................................... 1
      if (b_bed_start) {
        b_ch_assign_code; ................................................................ 1
        if (b_ch_assign_code) {
          bed_chan_assign_code; .......................................................... 3
          for (i = 0; i < [2, 3, 6, 8, 10, 8, 10, 12][bed_chan_assign_code]; i++) {
            obj_type[n_objs] = BED;
            if (i == 3) {
              b_lfe[n_objs] = 1;
```

Syntax	No of bits

```
          }
          else {
            b_lfe[n_objs] = 0;
          }
          b_ajoc_coded[n_objs] = 0;
          n_objs += 1;
        }
      }
      else {
        b_nonstd_bed_channel_assignment_flags_present; ...................................... 1
        if (b_nonstd_bed_channel_assignment_flags_present) {
          nonstd_bed_channel_assignment_flag[]; ............................................. 17
          for (i = 0; i < 17; i++) {
            if (nonstd_bed_channel_assignment_flag[16 - i] != 0) {
              obj_type[n_objs] = BED;
              if (i == 3 or i == 16) {
                b_lfe[n_objs] = 1;
              }
              else {
                b_lfe[n_objs] = 0;
              }
              b_ajoc_coded[n_objs] = 0;
              n_objs += 1;
            }
          }
        }
        else {
          std_bed_channel_assignment_flag[]; ................................................ 10
          for (i = 0; i < 10; i++) {
            if (std_bed_channel_assignment_flag[9 - i] != 0) {
              for (j = 0; j < [2, 1, 1, 2, 2, 2, 2, 2, 2, 1][i]; j++) {
                obj_type[n_objs] = BED;
                if (i == 2 or i == 9) {
                  b_lfe[n_objs] = 1;
                }
                else {
                  b_lfe[n_objs] = 0;
                }
                b_ajoc_coded[n_objs] = 0;
                n_objs += 1;
              }
            }
          }
        }
      }
    }
  }
  else {
    b_isf; .................................................................................. 1
    if (b_isf) {
      b_isf_start; .......................................................................... 1
      if (b_isf_start) {
        isf_config; ......................................................................... 3
        n_isf = [4, 8, 10, 14, 15, 30][isf_config];
        for (i = 0; i < n_isf; i++) {
          obj_type[n_objs] = ISF;
          b_lfe[n_objs] = 0;
          b_ajoc_coded[n_objs] = 0;
          n_objs += 1;
        }
      }
    }
    else {
      res_bytes; ............................................................................ 4
      reserved_data; ........................................................... 8 * res_bytes
    }
  }
}
if (fs_index == 1) {
  b_sf_multiplier; .......................................................................... 1
  if (b_sf_multiplier) {
    sf_multiplier; .......................................................................... 1
  }
}
b_bitrate_info; ............................................................................. 1
if (b_bitrate_info) {
  bitrate_indicator; ...................................................................... 3/5
```

Syntax	No of bits

```
  }
  for (i = 0; i < frame_rate_factor; i++) {
    b_audio_ndot; ............................................................................ 1
  }
  if (b_substreams_present == 1) {
    substream_index; ......................................................................... 2
    if (substream_index == 3) {
      substream_index += variable_bits(2);
    }
  }
  sus_ver = 1;
}
```

6.2.1.12 ac4_presentation_substream_info

Syntax	No of bits

```
ac4_presentation_substream_info()
{
  b_alternative; ........................................................................... 1
  b_pres_ndot; ............................................................................. 1
  substream_index; ......................................................................... 2
  if (substream_index == 3) {
    substream_index += variable_bits(2);
  }
}
```

6.2.1.13 oamd_substream_info

Syntax	No of bits

```
oamd_substream_info(b_substreams_present)
{
  b_oamd_ndot; ............................................................................. 1
  if (b_substreams_present == 1) {
    substream_index; ......................................................................... 2
    if (substream_index == 3) {
      substream_index += variable_bits(2);
    }
  }
}
```

6.2.1.14 ac4_hsf_ext_substream_info

Syntax	No of bits

```
ac4_hsf_ext_substream_info(b_substreams_present)
{
  if (b_substreams_present == 1) {
    substream_index; ......................................................................... 2
    if (substream_index == 3) {
      substream_index += variable_bits(2);
    }
  }
}
```

6.2.2 AC-4 substreams

6.2.2.1 Introduction

The `ac4_substream_data()` element for a specific substream index depends on the type of info element that refers to this specific substream. The mapping for the info elements defined in the present document is given in table 50, which is an extension of ETSI TS 103 190-1 [1], table 15.

Table 50: ac4_substream_data mapping

Info element type referencing the substream	ac4_substream_data element
ac4_substream_info()	ac4_substream()
ac4_substream_info_chan()	
ac4_substream_info_obj()	
ac4_substream_info_ajoc()	
ac4_hsf_ext_substream_info()	ac4_hsf_ext_substream()
emdf_payloads_substream_info()	emdf_payloads_substream()
ac4_presentation_substream_info()	ac4_presentation_substream()
oamd_substream_info()	oamd_substream()

The `ac4_hsf_ext_substream()` and `emdf_payloads_substream()` elements are specified in ETSI TS 103 190-1 [1]; `ac4_substream()`, `ac4_presentation_substream()`, and `oamd_substream()` are specified in the present document.

6.2.2.2 ac4_substream

Syntax	No of bits

```
ac4_substream()
{
  audio_size_value; ............................................................... 15
  audio_size = audio_size_value;
  b_more_bits; ...................................................................... 1
  if (b_more_bits) {
    audio_size += variable_bits(7) << 15;
  }
  if (b_channel_coded) {
    audio_data_chan(channel_mode, b_audio_ndot);
  }
  else {
    if (b_ajoc) {
      audio_data_ajoc(n_fullband_upmix_signals, b_static_dmx, n_fullband_dmx_signals, b_lfe,
b_audio_ndot);
    }
    else {
      audio_data_objs(n_objects, b_lfe, b_audio_ndot);
    }
  }
  fill_bits; ...................................................................... VAR
  byte_align; ..................................................................... 0…7
  metadata(b_alternative, b_ajoc, b_audio_ndot, channel_mode, sus_ver);
  byte_align; ..................................................................... 0…7
}
```

NOTE 1: *n_objects* is derived from `n_objects_code` according to clause 6.3.2.10.2

NOTE 2: If *channel_mode* has not been set by a preceding info element, it shall be considered undefined and be represented by a negative numeric value.

6.2.2.3 ac4_presentation_substream

Syntax	No of bits

```
ac4_presentation_substream()
{
  if (b_alternative) {
    b_name_present; ................................................................. 1
    if (b_name_present) {
      b_length; ..................................................................... 1
      if (b_length) {
        name_len; ................................................................... 5
      }
      else {
        name_len = 32;
      }
      presentation_name; ................................................... name_len*8
    }
    n_targets_minus1; ............................................................... 2
    n_targets = n_targets_minus1 + 1;
    if (n_targets == 4) {
      n_targets += variable_bits(2);
```

```
Syntax                                                                          No of bits
    }
    for (t = 0; t < n_targets; t++) {
      target_level; ........................................................... 3
      target_device_category[]; ............................................... 4
      b_tdc_extension; ........................................................ 1
      if (b_tdc_extension == 1) {
        reserved_bits; ........................................................ 4
      }
      b_ducking_depth_present; ................................................ 1
      if (b_ducking_depth_present) {
        max_ducking_depth; .................................................... 6
      }
      b_loud_corr_target; ..................................................... 1
      if (b_loud_corr_target) {
        loud_corr_target; ..................................................... 5
      }
      for (sus = 0; sus < n_substreams_in_presentation; sus++) {
        b_active; ............................................................. 1
        if (b_active) {
          alt_data_set_index; ................................................. 1
          if (alt_data_set_index == 1) {
            alt_data_set_index += variable_bits(2);
          }
        }
      }
    }
  }
  b_additional_data; .......................................................... 1
  if (b_additional_data) {
    add_data_bytes_minus1; .................................................... 4
    add_data_bytes = add_data_bytes_minus1 + 1;
    if (add_data_bytes == 16) {
      add_data_bytes += variable_bits(2);
    }
    byte_align; ............................................................. 0…7
    add_data_bits = add_data_bytes * 8;
    immersive_audio_indicator; ................................................ 1
    add_data_bits = add_data_bits - 1;
    if (pres_ch_mode == -1) {
      b_oamd_common_timing; ................................................... 1
      add_data_bits = add_data_bits - 1;
    }
    b_advanced_de_data_present; ............................................... 1
    add_data_bits = add_data_bits - 1;
    if (b_advanced_de_data_present) {
      /* if the advanced DE data element is not present in an I-frame, then A-DE is disabled */
      advanced_de_data_bits = advanced_de_data();
      add_data_bits = add_data_bits - advanced_de_data_bits;
    }
    add_data; ......................................................... add_data_bits
  }
  dialnorm_bits; .............................................................. 7
  b_further_loudness_info; .................................................... 1
  if (b_further_loudness_info) {
    further_loudness_info(1, 1);
  }
  drc_metadata_size_value; .................................................... 5
  drc_metadata_size = drc_metadata_size_value;
  b_more_bits; ................................................................ 1
  if (b_more_bits == 1) {
    drc_metadata_size += variable_bits(3) << 5;
  }
  drc_frame(b_pres_ndot);
  if (n_substream_groups > 1) {
    b_substream_group_gains_present; .......................................... 1
    if (b_substream_group_gains_present == 1) {
      b_keep; ................................................................. 1
      if (b_keep == 0) {
        for (sg = 0; sg < n_substream_groups; sg++) {
          sg_gain[sg]; ........................................................ 6
        }
      }
    }
  }
  b_associated; ............................................................... 1
  if (b_associated == 1) {
    b_scale_main; ............................................................. 1
```

Syntax	No of bits

```
    if (b_scale_main == 1) {
      scale_main; ............................................................ 8
    }
    b_scale_main_centre; ..................................................... 1
    if (b_scale_main_centre == 1) {
      scale_main_centre; ..................................................... 8
    }
    b_scale_main_front; ...................................................... 1
    if (b_scale_main_front == 1) {
      scale_main_front; ...................................................... 8
    }
    b_associate_is_mono; ..................................................... 1
    if (b_associate_is_mono == 1) {
      pan_associated; ........................................................ 8
    }
  }
  custom_dmx_data(pres_ch_mode, pres_ch_mode_core, b_pres_4_back_channels_present,
pres_top_channel_pairs, b_pres_has_lfe);
  b_objects = (pres_ch_mode == -1);
  loud_corr(pres_ch_mode, pres_ch_mode_core, b_objects);
  byte_align; ............................................................. 0…7
}
```

6.2.2.4 oamd_substream

Syntax	No of bits

```
oamd_substream()
{
  b_oamd_common_data_present; .............................................. 1
  if (b_oamd_common_data_present) {
    oamd_common_data();
  }
  b_oamd_timing_present; ................................................... 1
  if (b_oamd_timing_present) {
    oamd_timing_data();
  }
  if (b_alternative == 0) {
    oamd_dyndata_multi(n_objs, num_obj_info_blocks, b_oamd_ndot, obj_type[n_objs], b_lfe[n_objs],
b_ajoc_coded[n_objs]);
  }
  byte_align; ............................................................. 0…7
}
```

6.2.2.5 advanced_de_data

Syntax	No of bits

```
advanced_de_data()
{
  b_advanced_de_config_present; ............................................ 1
  advanced_de_data_bits = 1;
  if (b_advanced_de_config_present) {
    /* If not sent in an I-frame, then they will be reset to their defaults */
    advanced_de_compr_tc_attack; ........................................... 6
    advanced_de_compr_tc_release; .......................................... 6
    advanced_de_compr_ratio; ............................................... 4
    advanced_de_data_bits += 16;
  }
  advanced_de_compr_thresh; ................................................ 6
  advanced_de_compr_gain; .................................................. 5
  advanced_de_data_bits += 11;
  return advanced_de_data_bits;
}
```

6.2.3 Audio data

6.2.3.1 audio_data_chan

Syntax	No of bits

```
audio_data_chan(channel_mode, b_iframe)
{
  switch (channel_mode) {
    case mono:
      single_channel_element(b_iframe);
      break;
    case stereo:
      channel_pair_element(b_iframe);
      break;
    case 3.0:
      3_0_channel_element(b_iframe);
      break;
    case 5.0:
      5_X_channel_element(0, b_iframe);
      break;
    case 5.1:
      5_X_channel_element(1, b_iframe);
      break;
    case 7.X:
      7_X_channel_element(channel_mode, b_iframe);
      break;
    case 7.0.4:
      immersive_channel_element(0, 0, b_iframe);
      break;
    case 7.1.4:
      immersive_channel_element(1, 0, b_iframe);
      break;
    case 9.0.4:
      immersive_channel_element(0, 1, b_iframe);
      break;
    case 9.1.4:
      immersive_channel_element(1, 1, b_iframe);
      break;
    case 22.2:
      22_2_channel_element(b_iframe);
      break;
    default:
      break;
  }
}
```

6.2.3.2 audio_data_objs

Syntax	No of bits

```
audio_data_objs(n_objects, b_lfe, b_iframe)
{
  if (b_lfe) {
    mono_data(1);
  }
  if (n_objects != 0) {
    channel_mode = objs_to_channel_mode(n_objects);
    audio_data_chan(channel_mode, b_iframe);
  }
}
```

6.2.3.3 objs_to_channel_mode

Syntax	No of bits

```
objs_to_channel_mode(n_objects)
{
  switch (n_objects) {
    case 1:
      return mono;
      break;
```

Syntax	No of bits

```
    case 2:
      return stereo;
      break;
    case 3:
      return 3.0;
      break;
    case 5:
      return 5.0;
      break;
    default:
      break;
  }
}
```

6.2.3.4 audio_data_ajoc

Syntax	No of bits

```
audio_data_ajoc(n_fb_upmix_signals, b_static_dmx, n_fb_dmx_signals, b_lfe, b_iframe)
{
  if (b_static_dmx == 1) {
    audio_data_chan(b_lfe ? 5.1 : 5.0, b_iframe);
  }
  else {
    n_dmx_signals = n_fb_dmx_signals + (b_lfe?1:0);
    for (s = 0; s < n_dmx_signals; s++) {
      is_lfe[s] = 0;
    }
    if (b_lfe) {
      is_lfe[0] = 1;
    }
    b_some_signals_inactive; ........................................................ 1
    if (b_some_signals_inactive) {
      dmx_active_signals_mask[]; ............................................... n_fb_dmx_signals
    }
    var_channel_element(b_iframe, n_fb_dmx_signals, b_lfe);
    b_dmx_timing; ........................................................................ 1
    if (b_dmx_timing) {
      oamd_timing_data();
    }
    oamd_dyndata_single(n_dmx_signals, num_obj_info_blocks, b_iframe, b_alternative,
obj_type_dmx[n_dmx_signals], is_lfe[n_dmx_signals]);
    b_oamd_extension_present; ...................................................... 1
    if (b_oamd_extension_present) {
      skip_bits = (variable_bits(3) + 1) * 8;
      skip_bits = skip_bits - ajoc_bed_info();
      skip_data; ........................................................................ skip_bits
    }
  }
  ajoc(n_fb_dmx_signals, n_fb_upmix_signals);
  ajoc_dmx_de_data(n_fb_dmx_signals, n_fb_upmix_signals);
  b_umx_timing; ............................................................................ 1
  if (b_umx_timing == 1) {
    oamd_timing_data();
  }
  else {
    b_derive_timing_from_dmx; ...................................................... 1
  }
  n_umx_signals = n_fb_umx_signals + (b_lfe?1:0);
  for (s = 0; s < n_umx_signals; s++) {
    is_lfe[s] = 0;
  }
  if (b_lfe) {
    is_lfe[0] = 1;
  }
  oamd_dyndata_single(n_umx_signals, num_obj_info_blocks, b_iframe, b_alternative,
obj_type_umx[n_umx_signals], is_lfe[n_umx_signals]);
}
```

6.2.3.5 ajoc_dmx_de_data

Syntax	No of bits

```
ajoc_dmx_de_data(num_dmx_signals, num_umx_signals)
{
  b_dmx_de_cfg; ............................................................ 1
  b_keep_dmx_de_coeffs; .................................................... 1
  if (b_dmx_de_cfg) {
    de_max_gain; ........................................................... 2
    de_main_dlg_flag[]; ........................................ num_umx_signals
  }
  if (b_keep_dmx_de_coeffs == 0) {
    for (dio = 0; dio < num_dlg_obj; dio++) {
      for (dmxo = 0; dmxo < num_dmx_signals; dmxo++) {
        de_dlg_dmx_coeff_idx[dio][dmxo]; ................................. VAR
      }
    }
  }
}
```

6.2.3.6 ajoc_bed_info

Syntax	No of bits

```
ajoc_bed_info()
{
  b_obj_without_bed_info_present; .......................................... 1
  if (b_obj_without_bed_info_present) {
    num_obj_with_bed_render_info; .......................................... 3
  }
}
```

6.2.4 Channel elements

6.2.4.1 immersive_channel_element

Syntax	No of bits

```
immersive_channel_element(b_lfe, b_5fronts, b_iframe)
{
  immersive_codec_mode_code; ............................................. VAR
  if (b_iframe == 1) {
    immers_cfg(immersive_codec_mode);
  }
  if (b_lfe == 1) {
    mono_data(1);
  }
  if (immersive_codec_mode == ASPX_AJCC) {
    companding_control(5);
  }
  core_5ch_grouping; ....................................................... 2
  switch (core_5ch_grouping) {
    case 0:
      2ch_mode; ............................................................ 1
      two_channel_data();
      two_channel_data();
      mono_data(0);
      break;
    case 1:
      three_channel_data();
      two_channel_data();
      break;
    case 2:
      four_channel_data();
      mono_data(0);
      break;
    case 3:
      five_channel_data();
      break;
  }
  if (core_channel_config == 7CH_STATIC) {
```

Syntax	No of bits

```
    b_use_sap_add_ch; ................................................................... 1
    if (b_use_sap_add_ch == 1) {
      chparam_info();
      chparam_info();
    }
    two_channel_data();
  }
  if (immersive_codec_mode == ASPX_SCPL) {
    aspx_data_2ch();
    aspx_data_2ch();
    aspx_data_1ch();
    if (b_5fronts == 1) {
      aspx_data_2ch();
      aspx_data_2ch();
    }
    else {
      aspx_data_2ch();
    }
    aspx_data_2ch();
    aspx_data_2ch();
  }
  else {
    if (immersive_codec_mode in [ASPX_ACPL_1, ASPX_ACPL_2, ASPX_AJCC]) {
      aspx_data_2ch();
      aspx_data_2ch();
      if (core_channel_config == 7CH_STATIC) {
        aspx_data_2ch();
      }
      aspx_data_1ch();
    }
  }
  if (immersive_codec_mode == ASPX_AJCC) {
    ajcc_data(b_5fronts);
  }
  if (immersive_codec_mode in [SCPL, ASPX_SCPL, ASPX_ACPL_1]) {
    two_channel_data();
    two_channel_data();
    chparam_info();
    chparam_info();
    chparam_info();
    chparam_info();
    if (b_5fronts == 1) {
      two_channel_data();
      chparam_info();
      chparam_info();
    }
  }
  if (immersive_codec_mode in [ASPX_ACPL_1, ASPX_ACPL_2]) {
    acpl_data_1ch();
    acpl_data_1ch();
    acpl_data_1ch();
    acpl_data_1ch();
    if (b_5fronts == 1) {
      acpl_data_1ch();
      acpl_data_1ch();
    }
  }
}
```

NOTE: *immersive_codec_mode* is derived from `immersive_codec_mode_code` as specified in clause 6.3.5.1.
core_channel_config is derived from *immersive_codec_mode* as specified in clause 6.3.5.2.

6.2.4.2 immers_cfg

Syntax	No of bits

```
immers_cfg(immersive_codec_mode)
{
  if (immersive_codec_mode != SCPL) {
    aspx_config();
  }
  if (immersive_codec_mode == ASPX_ACPL_1) {
    acpl_config_1ch(PARTIAL);
  }
  if (immersive_codec_mode == ASPX_ACPL_2) {
```

Syntax	No of bits

acpl_config_1ch(FULL);
}
}	

6.2.4.3 22_2_channel_element

Syntax	No of bits

```
22_2_channel_element(b_iframe)
{
  22_2_codec_mode; ........................................................ 1
  if (b_iframe) {
    if (22_2_codec_mode == ASPX) {
      aspx_config();
    }
  }
  mono_data(1);
  mono_data(1);
  for (cp = 0; cp < 11; cp++) {
    two_channel_data();
  }
  if (22_2_codec_mode == ASPX) {
    for (cp = 0; cp < 11; cp++) {
      aspx_data_2ch(b_iframe);
    }
  }
}
```

6.2.4.4 var_channel_element

Syntax	No of bits

```
var_channel_element(b_iframe, n_dmx_signals, b_has_lfe)
{
  var_codec_mode; ......................................................... 1
  b_isodd = n_dmx_signals % 2;
  n_pairs = floor(n_dmx_signals / 2);
  if (var_codec_mode == ASPX) {
    if (b_iframe) {
      aspx_config();
    }
    if (n_dmx_signals <= 5) {
      companding_control(n_dmx_signals);
    }
  }
  if (b_has_lfe) {
    mono_data(1);
  }
  if (b_isodd) {
    if (n_dmx_signals == 1) {
      mono_data(0);
    }
    else {
      for (p = 0; p < n_pairs - 1; p++) {
        two_channel_data();
      }
      var_coding_config; .................................................. 1
      if (var_coding_config == 0) {
        two_channel_data();
        mono_data(0);
      }
      else {
        three_channel_data();
      }
    }
  }
  else {
    for (p = 0; p < n_pairs; p++) {
      two_channel_data();
    }
  }
  if (var_codec_mode == ASPX) {
```

Syntax	No of bits

```
    for (p = 0; p < n_pairs; p++) {
      aspx_data_2ch();
    }
    if (b_isodd) {
      aspx_data_1ch();
    }
  }
}
```

6.2.5 Advanced joint object coding (A-JOC)

6.2.5.1 ajoc

Syntax	No of bits

```
ajoc(num_dmx_signals, num_umx_signals)
{
  ajoc_num_decorr; ........................................................................... 3
  ajoc_ctrl_info(num_dmx_signals, ajoc_num_decorr, num_umx_signals);
  ajoc_data(num_dmx_signals, num_umx_signals);
}
```

6.2.5.2 ajoc_ctrl_info

Syntax	No of bits

```
ajoc_ctrl_info(num_dmx_signals, ajoc_num_decorr, num_umx_signals)
{
  for (d = 0; d < ajoc_num_decorr; d++) {
    ajoc_decorr_enable[d]; ................................................................... 1
  }
  for (o = 0; o < num_umx_signals; o++) {
    ajoc_object_present[o]; .................................................................. 1
  }
  ajoc_data_point_info();
  if (ajoc_num_dpoints) {
    for (o = 0; o < num_umx_signals; o++) {
      if (ajoc_object_present[o]) {
        ajoc_num_bands_code[o]; .............................................................. 3
        ajoc_quant_select[o]; ................................................................ 1
        ajoc_sparse_select[o]; ............................................................... 1
        if (ajoc_sparse_select[o] == 1) {
          for (ch = 0; ch < num_dmx_signals; ch++) {
            ajoc_mix_mtx_dry_present[o][ch]; ................................................. 1
          }
          for (d = 0; d < ajoc_num_decorr; d++) {
            if (ajoc_decorr_enable[d]) {
              ajoc_mix_mtx_wet_present[o][d]; ................................................ 1
            }
            else {
              ajoc_mix_mtx_wet_present[o][d] = 0;
            }
          }
        }
      }
    }
  }
}
```

6.2.5.3 ajoc_data

Syntax	No of bits

```
ajoc_data(num_dmx_signals, num_umx_signals)
{
  ajoc_b_nodt; ........................................................................ 1
  for (o = 0; o < num_umx_signals; o++) {
    for (dp = 0; dp < ajoc_num_dpoints; dp++) {
      for (ch = 0; ch < num_dmx_signals; ch++) {
        mix_mtx_dry[o][dp][ch] = 0;
      }
      for (de = 0; de < ajoc_num_decorr; de++) {
        mix_mtx_wet[o][dp][de] = 0;
      }
    }
    if (ajoc_object_present[o]) {
      for (dp = 0; dp < ajoc_num_dpoints; dp++) {
        b_dfonly = (dp == 0 && ajoc_b_nodt);
        nb = ajoc_num_bands[o];
        qs = ajoc_quant_select[o];
        switch (ajoc_sparse_select[o]) {
          case 0:
            for (ch = 0; ch < num_dmx_signals; ch++) {
              mix_mtx_dry[o][dp][ch] = ajoc_huff_data(DRY, nb, qs, b_dfonly);
            }
            for (de = 0; de < ajoc_num_decorr; de++) {
              mix_mtx_wet[o][dp][de] = ajoc_huff_data(WET, nb, qs, b_dfonly);
            }
            break;
          case 1:
            for (ch = 0; ch < num_dmx_signals; ch++) {
              if (ajoc_mix_mtx_dry_present[o][ch]) {
                mix_mtx_dry[o][dp][ch] = ajoc_huff_data(DRY, nb, qs, b_dfonly);
              }
            }
            for (de = 0; de < ajoc_num_decorr; de++) {
              if (ajoc_mix_mtx_wet_present[o][de]) {
                mix_mtx_wet[o][dp][de] = ajoc_huff_data(WET, nb, qs, b_dfonly);
              }
            }
            break;
        }
      }
    }
  }
}
```

NOTE: The *ajoc_num_bands* values are derived using clause 6.3.6.2.3.

6.2.5.4 ajoc_data_point_info

Syntax	No of bits

```
ajoc_data_point_info()
{
  ajoc_num_dpoints; ................................................................... 2
  for (dp = 0; dp < ajoc_num_dpoints; dp++) {
    ajoc_start_pos[dp]; ............................................................... 5
    ajoc_ramp_len_minus1[dp]; ......................................................... 6
  }
}
```

6.2.5.5 ajoc_huff_data

Syntax	No of bits

```
ajoc_huff_data(data_type, data_bands, quant_select, b_dfonly)
{
  if (b_dfonly) {
    diff_type = 0;
  }
  else {
    diff_type; ........................................................................ 1
```

Syntax	No of bits

`` }
  if (diff_type == 0) {
    ajoc_hcb = get_ajoc_hcb(data_type, quant_select, F0);
    ajoc_hcw; .... VAR
    a_huff_data[0] = huff_decode(ajoc_hcb, ajoc_hcw);
    ajoc_hcb = get_ajoc_hcb(data_type, quant_select, DF);
    for (i = 1; i < data_bands; i++) {
      ajoc_hcw; .... VAR
      a_huff_data[i] = huff_decode_diff(ajoc_hcb, ajoc_hcw);
    }
  }
  else {
    ajoc_hcb = get_ajoc_hcb(data_type, quant_select, DT);
    for (i = 0; i < data_bands; i++) {
      ajoc_hcw; .... VAR
      a_huff_data[i] = huff_decode_diff(ajoc_hcb, ajoc_hcw);
    }
  }
  return a_huff_data;
} ```	

NOTE: The function *get_ajoc_hcb()* is defined in clause 6.3.6.5.2.	

6.2.6 Advanced Joint Channel Coding (A-JCC)

6.2.6.1 ajcc_data

```
Syntax                                                                  No of bits
ajcc_data(b_5fronts)
{
  b_no_dt; ............................................................... 1
  ajcc_num_param_bands_id; ............................................... 2
  num_bands = ajcc_num_bands_table[ajcc_num_param_bands_id];
  if (b_5fronts == 1) {
    ajcc_qm_f; ........................................................... 1
    ajcc_qm_b; ........................................................... 1
  }
  else {
    ajcc_core_mode; ...................................................... 1
    ajcc_qm_ab; .......................................................... 1
    ajcc_qm_dw; .......................................................... 1
  }
  if (b_5fronts == 1) {
    ajcc_nps_lf = ajcc_framing_data();
    ajcc_nps_rf = ajcc_framing_data();
    ajcc_nps_lb = ajcc_framing_data();
    ajcc_nps_rb = ajcc_framing_data();
    ajcc_dry1f = ajced(DRY, num_bands, ajcc_qm_f, b_no_dt, ajcc_nps_lf);
    ajcc_dry2f = ajced(DRY, num_bands, ajcc_qm_f, b_no_dt, ajcc_nps_lf);
    ajcc_dry3f = ajced(DRY, num_bands, ajcc_qm_f, b_no_dt, ajcc_nps_rf);
    ajcc_dry4f = ajced(DRY, num_bands, ajcc_qm_f, b_no_dt, ajcc_nps_rf);
    ajcc_dry1b = ajced(DRY, num_bands, ajcc_qm_b, b_no_dt, ajcc_nps_lb);
    ajcc_dry2b = ajced(DRY, num_bands, ajcc_qm_b, b_no_dt, ajcc_nps_lb);
    ajcc_dry3b = ajced(DRY, num_bands, ajcc_qm_b, b_no_dt, ajcc_nps_rb);
    ajcc_dry4b = ajced(DRY, num_bands, ajcc_qm_b, b_no_dt, ajcc_nps_rb);
    ajcc_wet1f = ajced(WET, num_bands, ajcc_qm_f, b_no_dt, ajcc_nps_lf);
    ajcc_wet2f = ajced(WET, num_bands, ajcc_qm_f, b_no_dt, ajcc_nps_lf);
    ajcc_wet3f = ajced(WET, num_bands, ajcc_qm_f, b_no_dt, ajcc_nps_lf);
    ajcc_wet4f = ajced(WET, num_bands, ajcc_qm_f, b_no_dt, ajcc_nps_rf);
    ajcc_wet5f = ajced(WET, num_bands, ajcc_qm_f, b_no_dt, ajcc_nps_rf);
    ajcc_wet6f = ajced(WET, num_bands, ajcc_qm_f, b_no_dt, ajcc_nps_rf);
    ajcc_wet1b = ajced(WET, num_bands, ajcc_qm_b, b_no_dt, ajcc_nps_lb);
    ajcc_wet2b = ajced(WET, num_bands, ajcc_qm_b, b_no_dt, ajcc_nps_lb);
    ajcc_wet3b = ajced(WET, num_bands, ajcc_qm_b, b_no_dt, ajcc_nps_lb);
    ajcc_wet4b = ajced(WET, num_bands, ajcc_qm_b, b_no_dt, ajcc_nps_rb);
    ajcc_wet5b = ajced(WET, num_bands, ajcc_qm_b, b_no_dt, ajcc_nps_rb);
    ajcc_wet6b = ajced(WET, num_bands, ajcc_qm_b, b_no_dt, ajcc_nps_rb);
  }
  else {
    ajcc_nps_l = ajcc_framing_data();
    ajcc_nps_r = ajcc_framing_data();
    ajcc_alpha1 = ajced(ALPHA, num_bands, ajcc_qm_ab, b_no_dt, ajcc_nps_l);
    ajcc_alpha2 = ajced(ALPHA, num_bands, ajcc_qm_ab, b_no_dt, ajcc_nps_r);
```

Syntax	No of bits

```
    ajcc_beta1 = ajced(BETA, num_bands, ajcc_qm_ab, b_no_dt, ajcc_nps_l);
    ajcc_beta2 = ajced(BETA, num_bands, ajcc_qm_ab, b_no_dt, ajcc_nps_r);
    ajcc_dry1 = ajced(DRY, num_bands, ajcc_qm_dw, b_no_dt, ajcc_nps_l);
    ajcc_dry2 = ajced(DRY, num_bands, ajcc_qm_dw, b_no_dt, ajcc_nps_l);
    ajcc_dry3 = ajced(DRY, num_bands, ajcc_qm_dw, b_no_dt, ajcc_nps_r);
    ajcc_dry4 = ajced(DRY, num_bands, ajcc_qm_dw, b_no_dt, ajcc_nps_r);
    ajcc_wet1 = ajced(WET, num_bands, ajcc_qm_dw, b_no_dt, ajcc_nps_l);
    ajcc_wet2 = ajced(WET, num_bands, ajcc_qm_dw, b_no_dt, ajcc_nps_l);
    ajcc_wet3 = ajced(WET, num_bands, ajcc_qm_dw, b_no_dt, ajcc_nps_l);
    ajcc_wet4 = ajced(WET, num_bands, ajcc_qm_dw, b_no_dt, ajcc_nps_r);
    ajcc_wet5 = ajced(WET, num_bands, ajcc_qm_dw, b_no_dt, ajcc_nps_r);
    ajcc_wet6 = ajced(WET, num_bands, ajcc_qm_dw, b_no_dt, ajcc_nps_r);
  }
}
```

6.2.6.2 ajcc_framing_data

Syntax	No of bits

```
ajcc_framing_data()
{
  ajcc_interpolation_type; ............................................................ 1
  ajcc_num_param_sets_code; ........................................................... 1
  if (ajcc_interpolation_type == 1) {
    for (ps = 0; ps < ajcc_num_param_sets_code + 1; ps++) {
      ajcc_param_timeslot[ps]; ........................................................ 5
    }
  }
  return ajcc_num_param_sets_code + 1;
}
```

6.2.6.3 ajced

Syntax	No of bits

```
ajced(data_type, data_bands, quant_mode, b_no_dt, num_ps)
{
  for (ps = 0; ps < num_ps; ps++) {
    a_param_set[ps] = ajcc_huff_data(data_type, data_bands, quant_mode, b_no_dt);
  }
  return a_param_set;
}
```

6.2.6.4 ajcc_huff_data

Syntax	No of bits

```
ajcc_huff_data(data_type, data_bands, quant_mode, b_no_dt)
{
  if (b_no_dt == 1) {
    diff_type = 0;
  }
  else {
    diff_type; ........................................................................ 1
  }
  if (diff_type == 0) {
    ajcc_hcb = get_ajcc_hcb(data_type, quant_mode, F0);
    ajcc_hcw; ....................................................................... VAR
    a_huff_data[0] = huff_decode(ajcc_hcb, ajcc_hcw);
    ajcc_hcb = get_ajcc_hcb(data_type, quant_mode, DF);
    for (band = 1; band < data_bands; band++) {
      ajcc_hcw; ..................................................................... VAR
      a_huff_data[band] = huff_decode_diff(ajcc_hcb, ajcc_hcw);
    }
  }
  else {
    ajcc_hcb = get_ajcc_hcb(data_type, quant_mode, DT);
    for (band = 0; band < data_bands; band++) {
      ajcc_hcw; ..................................................................... VAR
      a_huff_data[band] = huff_decode_diff(ajcc_hcb, ajcc_hcw);
```

Syntax	No of bits

```
    }
  }
  return a_huff_data;
}
```

6.2.7 Metadata

6.2.7.1 metadata

Syntax	No of bits

```
metadata(b_alternative, b_ajoc, b_iframe, channel_mode, sus_ver)
{
  basic_metadata(channel_mode, sus_ver);
  extended_metadata(channel_mode, sus_ver);
  if (b_alternative and b_ajoc == 0) {
    oamd_dyndata_single(n_objs, num_obj_info_blocks, b_iframe, b_alternative, obj_type[n_objs],
b_lfe[n_objs]);
  }
  tools_metadata_size_value; ........................................................... 7
  tools_metadata_size = tools_metadata_size_value;
  b_more_bits; ......................................................................... 1
  if (b_more_bits) {
    tools_metadata_size += variable_bits(3) << 7;
  }
  if (sus_ver == 0) {
    drc_frame(b_iframe);
  }
  dialog_enhancement(b_iframe);
  b_emdf_payloads_substream; ........................................................... 1
  if (b_emdf_payloads_substream) {
    emdf_payloads_substream();
  }
}
```

6.2.7.2 basic_metadata

Syntax	No of bits

```
basic_metadata(channel_mode, sus_ver)
{
  if (sus_ver == 0) {
    dialnorm_bits; ..................................................................... 7
  }
  b_more_basic_metadata; ............................................................... 1
  if (b_more_basic_metadata) {
    if (sus_ver == 0) {
      b_further_loudness_info; ......................................................... 1
      if (b_further_loudness_info) {
        further_loudness_info(sus_ver, 0);
      }
    }
    else {
      b_substream_loudness_info; ....................................................... 1
      if (b_substream_loudness_info) {
        substream_loudness_bits; ....................................................... 8
        b_further_substream_loudness_info; ............................................. 1
        if (b_further_substream_loudness_info) {
          further_loudness_info(sus_ver, 0);
        }
      }
    }
    if (channel_mode == stereo) {
      b_prev_dmx_info; ................................................................. 1
      if (b_prev_dmx_info) {
        pre_dmixtyp_2ch; ............................................................... 3
        phase90_info_2ch; .............................................................. 2
      }
    }
    if (channel_mode > stereo) {
      if (sus_ver == 0) {
```

Syntax	No of bits
b_stereo_dmx_coeff;	1
if (b_stereo_dmx_coeff) {	
loro_centre_mixgain;	3
loro_surround_mixgain;	3
b_loro_dmx_loud_corr;	1
if (b_loro_dmx_loud_corr) {	
loro_dmx_loud_corr;	5
}	
b_ltrt_mixinfo;	1
if (b_ltrt_mixinfo) {	
ltrt_centre_mixgain;	3
ltrt_surround_mixgain;	3
}	
b_ltrt_dmx_loud_corr;	1
if (b_ltrt_dmx_loud_corr) {	
ltrt_dmx_loud_corr;	5
}	
if (channel_mode_contains_Lfe()) {	
b_lfe_mixinfo;	1
if (b_lfe_mixinfo) {	
lfe_mixgain;	5
}	
}	
preferred_dmx_method;	2
}	
}	
if (channel_mode == 5_X) {	
b_predmixtyp_5ch;	1
if (b_predmixtyp_5ch) {	
pre_dmixtyp_5ch;	3
}	
b_preupmixtyp_5ch;	1
if (b_preupmixtyp_5ch) {	
pre_upmixtyp_5ch;	4
}	
}	
if (channel_mode == 7_X) {	
b_upmixtyp_7ch;	1
if (b_upmixtyp_7ch) {	
if (3/4/0) {	
pre_upmixtyp_3_4;	2
}	
else {	
if (3/2/2) {	
pre_upmixtyp_3_2_2;	1
}	
}	
}	
}	
phase90_info_mc;	2
b_surround_attenuation_known;	1
b_lfe_attenuation_known;	1
}	
b_dc_blocking;	1
if (b_dc_blocking) {	
dc_block_on;	1
}	
}	
}	

6.2.7.3 further_loudness_info

Syntax	No of bits
further_loudness_info(sus_ver, b_presentation_ldn)	
{	
if (b_presentation_ldn or sus_ver == 0) {	
loudness_version;	2
if (loudness_version == 3) {	
extended_loudness_version;	4
loudness_version += extended_loudness_version;	
}	
loud_prac_type;	4
if (loud_prac_type != 0) {	
b_loudcorr_dialgate;	1

Syntax	No of bits

```
    if (b_loudcorr_dialgate) {
      dialgate_prac_type; ........................................................ 3
    }
    b_loudcorr_type; ............................................................. 1
  }
}
else {
  b_loudcorr_dialgate; ........................................................... 1
}
b_loudrelgat; .................................................................... 1
if (b_loudrelgat) {
  loudrelgat; .................................................................... 11
}
b_loudspchgat; ................................................................... 1
if (b_loudspchgat) {
  loudspchgat; ................................................................... 11
  dialgate_prac_type; ............................................................ 3
}
b_loudstrm3s; .................................................................... 1
if (b_loudstrm3s) {
  loudstrm3s; .................................................................... 11
}
b_max_loudstrm3s; ................................................................ 1
if (b_max_loudstrm3s) {
  max_loudstrm3s; ................................................................ 11
}
b_truepk; ........................................................................ 1
if (b_truepk) {
  truepk; ........................................................................ 11
}
b_max_truepk; .................................................................... 1
if (b_max_truepk) {
  max_truepk; .................................................................... 11
}
if (b_presentation_ldn or sus_ver == 0) {
  b_prgmbndy; .................................................................... 1
  if (b_prgmbndy) {
    prgmbndy = 1;
    prgmbndy_bit = 0;
    while (prgmbndy_bit == 0) {
      prgmbndy <<= 1;
      prgmbndy_bit; .............................................................. 1
    }
    b_end_or_start; .............................................................. 1
    b_prgmbndy_offset; ........................................................... 1
    if (b_prgmbndy_offset) {
      prgmbndy_offset; ........................................................... 11
    }
  }
}
b_lra; ........................................................................... 1
if (b_lra) {
  lra; ........................................................................... 10
  lra_prac_type; ................................................................. 3
}
b_loudmntry; ..................................................................... 1
if (b_loudmntry) {
  loudmntry; ..................................................................... 11
}
b_max_loudmntry; ................................................................. 1
if (b_max_loudmntry) {
  max_loudmntry; ................................................................. 11
}
if (sus_ver >= 1) {
  b_rtllcomp; .................................................................... 1
  if (b_rtllcomp) {
    rtll_comp; ................................................................... 8
  }
  b_extension; ................................................................... 1
  if (b_extension) {
    e_bits_size; ................................................................. 5
    if (e_bits_size == 31) {
      e_bits_size += variable_bits(4);
    }
    extensions_bits; .................................................... e_bits_size
  }
}
```

Syntax	No of bits
` else {`	
`   b_extension;`	1
`   if (b_extension) {`	
`     e_bits_size;`	5
`     if (e_bits_size == 31) {`	
`       e_bits_size += variable_bits(4);`	
`     }`	
`     b_rtllcomp;`	1
`     if (b_rtllcomp) {`	
`       rtll_comp;`	8
`       extensions_bits;`	e_bits_size - 9
`     }`	
`     else {`	
`       extensions_bits;`	e_bits_size - 1
`     }`	
`   }`	
` }`	
`}`	

6.2.7.4 extended_metadata

Syntax	No of bits
`extended_metadata(channel_mode, sus_ver)`	
`{`	
`  if (sus_ver >= 1) {`	
`    b_dialog;`	1
`  }`	
`  else {`	
`    if (b_associated) {`	
`      b_scale_main;`	1
`      if (b_scale_main) {`	
`        scale_main;`	8
`      }`	
`      b_scale_main_centre;`	1
`      if (b_scale_main_centre) {`	
`        scale_main_centre;`	8
`      }`	
`      b_scale_main_front;`	1
`      if (b_scale_main_front) {`	
`        scale_main_front;`	8
`      }`	
`      if (channel_mode == mono) {`	
`        pan_associated;`	8
`      }`	
`    }`	
`  }`	
`  if (b_dialog) {`	
`    b_dialog_max_gain;`	1
`    if (b_dialog_max_gain) {`	
`      dialog_max_gain;`	2
`    }`	
`    b_pan_dialog_present;`	1
`    if (b_pan_dialog_present) {`	
`      if (channel_mode == mono) {`	
`        pan_dialog;`	8
`      }`	
`      else {`	
`        pan_dialog[0];`	8
`        pan_dialog[1];`	8
`        pan_signal_selector;`	2
`      }`	
`    }`	
`  }`	
`  b_channels_classifier;`	1
`  if (b_channels_classifier) {`	
`    if (channel_mode_contains_c()) {`	
`      b_c_active;`	1
`      if (b_c_active) {`	
`        b_c_has_dialog;`	1
`      }`	
`    }`	
`    if (channel_mode_contains_lr()) {`	
`      b_l_active;`	1
`      if (b_l_active) {`	

Syntax	No of bits
`        b_l_has_dialog;`	1
`      }`	
`      b_r_active;`	1
`      if (b_r_active) {`	
`        b_r_has_dialog;`	1
`      }`	
`    }`	
`    if (channel_mode_contains_LsRs()) {`	
`      b_ls_active;`	1
`      b_rs_active;`	1
`    }`	
`    if (channel_mode_contains_LbRb()) {`	
`      b_lb_active;`	1
`      b_rb_active;`	1
`    }`	
`    if (channel_mode_contains_LwRw()) {`	
`      b_lw_active;`	1
`      b_rw_active;`	1
`    }`	
`    if (channel_mode_contains_TflTfr()) {`	
`      b_tfl_active;`	1
`      b_tfr_active;`	1
`    }`	
`    if (channel_mode_contains_Lfe()) {`	
`      b_lfe_active;`	1
`    }`	
`  }`	
`  b_event_probability;`	1
`  if (b_event_probability) {`	
`    event_probability;`	4
`  }`	
`}`	
NOTE: For *sus_ver* == 0, *b_associated* is determined as specified in ETSI TS 103 190-1 [1], clause 4.3.12.4.1, and *b_dialog* is determined as specified in ETSI TS 103 190-1 [1], clause 4.3.12.4.2.	

6.2.7.5 dialog_enhancement

Syntax	No of bits
`dialog_enhancement(b_iframe)`	
`{`	
`  b_de_data_present;`	1
`  if (b_de_data_present) {`	
`    if (b_iframe) {`	
`      de_config();`	
`    }`	
`    else {`	
`      b_de_config_flag;`	1
`      if (b_de_config_flag) {`	
`        de_config();`	
`      }`	
`    }`	
`    de_data(de_method, de_nr_channels, b_iframe, 0);`	
`    if (ch_mode == 13 || ch_mode == 14) {`	
`      b_de_simulcast;`	1
`      if (b_de_simulcast) {`	
`        de_data(de_method, de_nr_channels, b_iframe, 1);`	
`      }`	
`    }`	
`  }`	
`}`	

6.2.7.6 de_data

Syntax	No of bits
`de_data(de_method, de_nr_channels, b_iframe, b_de_simulcast)`	
`{`	
`  if (de_nr_channels > 0) {`	
`    if ((de_method == 1 or de_method == 3) and de_nr_channels > 1) {`	
`      if (b_de_simulcast == 0) {`	
`        if (b_iframe) {`	

Syntax	No of bits

```
          de_keep_pos_flag = 0;
        }
        else {
          de_keep_pos_flag; .......................................................... 1
        }
        if (de_keep_pos_flag == 0) {
          de_mix_coef1_idx; .......................................................... 5
          if (de_nr_channels == 3) {
            de_mix_coef2_idx; ........................................................ 5
          }
        }
      }
    }
    if (b_iframe) {
      de_keep_data_flag = 0;
    }
    else {
      de_keep_data_flag; ............................................................. 1
    }
    if (de_keep_data_flag == 0) {
      if ((de_method == 0 or de_method == 2) and de_nr_channels == 2) {
        de_ms_proc_flag; ............................................................. 1
      }
      else {
        de_ms_proc_flag = 0;
      }
      for (ch = 0; ch < de_nr_channels - de_ms_proc_flag; ch++) {
        if (b_iframe != 0 and ch == 0) {
          de_par_code; ............................................................. VAR
          de_par[0][0] = de_abs_huffman(de_method % 2, de_par_code);
          ref_val = de_par[0][0];
          de_par_prev[0][0] = de_par[0][0];
          for (band = 1; band < de_nr_bands; band++) {
            de_par_code; ........................................................... VAR
            de_par[0][band] = ref_val + de_diff_huffman(de_method % 2, de_par_code);
            ref_val = de_par[0][band];
            de_par_prev[0][band] = de_par[0][band];
          }
        }
        else {
          for (band = 0; band < de_nr_bands; band++) {
            if (b_iframe) {
              de_par_code; ......................................................... VAR
              de_par[ch][band] = ref_val + de_diff_huffman(de_method % 2, de_par_code);
              ref_val = de_par[0][band];
            }
            else {
              de_par_code; ......................................................... VAR
              de_par[ch][band] = de_par_prev[ch][band] + de_diff_huffman(de_method % 2,
de_par_code);
            }
            de_par_prev[ch][band] = de_par[ch][band];
          }
        }
        ref_val = de_par[ch][0];
      }
      if (de_method >= 2) {
        de_signal_contribution; ...................................................... 5
      }
    }
  }
}
```

6.2.8 Object Audio Metadata (OAMD)

6.2.8.1 oamd_common_data

Syntax	No of bits

```
oamd_common_data()
{
  b_default_screen_size_ratio; ....................................................... 1
  if (b_default_screen_size_ratio == 0) {
```

Syntax	No of bits

```
    master_screen_size_ratio_code; ............................................ 5
  }
  b_bed_object_chan_distribute; ............................................... 1
  b_additional_data; .......................................................... 1
  if (b_additional_data) {
    add_data_bytes_minus1; .................................................... 1
    add_data_bytes = add_data_bytes_minus1 + 1;
    if (add_data_bytes == 2) {
      add_data_bytes += variable_bits(2);
    }
    add_data_bits = add_data_bytes * 8;
    bits_used = trim();
    add_data_bits = add_data_bits - bits_used;
    if (add_data_bits) {
      bits_used = bed_render_info();
      add_data_bits = add_data_bits - bits_used;
    }
    if (add_data_bits) {
      bits_used = headphone();
      add_data_bits = add_data_bits - bits_used;
    }
    add_data; ....................................................... add_data_bits
  }
}
```

6.2.8.2 oamd_timing_data

Syntax	No of bits

```
oamd_timing_data()
{
  oa_sample_offset_type; .................................................... 1/2
  if (oa_sample_offset_type == 0b10) {
    oa_sample_offset_code; .................................................. 1/2
  }
  else {
    if (oa_sample_offset_type == 0b11) {
      oa_sample_offset; ....................................................... 5
    }
  }
  num_obj_info_blocks; ........................................................ 3
  for (blk = 0; blk < num_obj_info_blocks; blk++) {
    block_offset_factor; ...................................................... 6
    ramp_duration_code; ....................................................... 2
    if (ramp_duration_code == 0b11) {
      b_use_ramp_table; ....................................................... 1
      if (b_use_ramp_table) {
        ramp_duration_table; .................................................. 4
      }
      else {
        ramp_duration; ....................................................... 11
      }
    }
  }
}
```

6.2.8.3 oamd_dyndata_single

Syntax	No of bits

```
oamd_dyndata_single(n_objs, n_blocks, b_iframe, b_alternative, obj_type[n_objs], b_lfe[n_objs])
{
  for (i = 0; i < n_objs; i++) {
    if (obj_type[i] == DYN and b_lfe[i] == 0) {
      b_dynamic_object = 1;
    }
    else {
      b_dynamic_object = 0;
    }
    for (b = 0; b < n_blocks; b++) {
      object_info_block((b_iframe != 0) and (b == 0), b_dynamic_object);
    }
```

Syntax	No of bits
` }`	
` if (b_alternative) {`	
`  **b_ducking_disabled**;`	1
`  **object_sound_category**;`	2
`  if (object_sound_category == 3) {`	
`   object_sound_category += variable_bits(2);`	
`  }`	
`  **n_alt_data_sets**;`	2
`  if (n_alt_data_sets == 3) {`	
`   n_alt_data_sets += variable_bits(2);`	
`  }`	
`  for (s = 0; s < n_alt_data_sets; s++) {`	
`   **b_keep**;`	1
`   if (b_keep == 0) {`	
`    n_data_points = n_objs;`	
`    if (obj_type[0] == ISF) {`	
`     n_data_points = 1;`	
`    }`	
`    else {`	
`     **b_common_data**;`	1
`     if (b_common_data) {`	
`      n_data_points = 1;`	
`     }`	
`    }`	
`    for (dp = 0; dp < n_data_points; dp++) {`	
`     if (obj_type[dp] == BED || obj_type[dp] == ISF) {`	
`      **b_alt_gain**;`	1
`      if (b_alt_gain) {`	
`       **alt_obj_gain**;`	6
`      }`	
`     }`	
`     else {`	
`      if (obj_type[dp] == DYN) {`	
`       **b_alt_gain**;`	1
`       if (b_alt_gain) {`	
`        **alt_obj_gain**;`	6
`       }`	
`       if (b_lfe[dp] == 0) {`	
`        **b_alt_position**;`	1
`        if (b_alt_position) {`	
`         **alt_pos3D_X**;`	6
`         **alt_pos3D_Y**;`	6
`         **alt_pos3D_Z_sign**;`	1
`         **alt_pos3D_Z**;`	4
`        }`	
`       }`	
`      }`	
`     }`	
`    }`	
`   }`	
`   **b_additional_data**;`	1
`   if (b_additional_data) {`	
`    skip_bits = (variable_bits(2) + 1) * 8;`	
`    skip_bits = skip_bits - ext_prec_alt_pos(n_objs, b_keep, obj_type, b_lfe);`	
`    **skip_data**;`	skip_bits
`   }`	
`  }`	
` }`	
`}`	

6.2.8.4 oamd_dyndata_multi

Syntax	No of bits
`oamd_dyndata_multi(n_objs, n_blocks, b_iframe, obj_type[n_objs], b_lfe[n_objs], b_ajoc_coded[n_objs])`	
`{`	
` for (i = 0; i < n_objs; i++) {`	
`  if (b_ajoc_coded[i] == 0) {`	
`   if (obj_type[i] == DYN and b_lfe[i] == 0) {`	
`    b_dynamic_object = 1;`	
`   }`	
`   else {`	
`    b_dynamic_object = 0;`	
`   }`	

Syntax	No of bits

`for (b = 0; b < n_blocks; b++) {`
`  object_info_block((b_iframe != 0) and (b == 0), b_dynamic_object);`
`}`
`}`
`}`
`}`	

6.2.8.5 object_info_block

```
Syntax                                                                                  No of bits
object_info_block(b_no_delta, b_dynamic_object)
{
  b_object_not_active; ..................................................................... 1
  if (b_object_not_active) {
    object_basic_info_status = DEFAULT;
  }
  else {
    if (b_no_delta) {
      object_basic_info_status = ALL_NEW;
    }
    else {
      b_basic_info_reuse; .................................................................. 1
      if (b_basic_info_reuse) {
        object_basic_info_status = REUSE;
      }
      else {
        object_basic_info_status = ALL_NEW;
      }
    }
  }
  if (object_basic_info_status == ALL_NEW) {
    object_basic_info();
  }
  if (b_object_not_active) {
    object_render_info_status = DEFAULT;
  }
  else {
    if (b_dynamic_object) {
      if (b_no_delta) {
        object_render_info_status = ALL_NEW;
      }
      else {
        b_render_info_reuse; ............................................................... 1
        if (b_render_info_reuse) {
          object_render_info_status = REUSE;
        }
        else {
          b_render_info_partial_reuse; ..................................................... 1
          if (b_render_info_partial_reuse) {
            object_render_info_status = PART_REUSE;
          }
          else {
            object_render_info_status = ALL_NEW;
          }
        }
      }
    }
    else {
      object_render_info_status = DEFAULT;
    }
  }
  if (object_render_info_status == ALL_NEW or object_render_info_status == PART_REUSE) {
    object_render_info(object_render_info_status, b_no_delta);
  }
  b_add_table_data; ........................................................................ 1
  if (b_add_table_data) {
    add_table_data_size_minus1; ............................................................ 4
    atd_size = add_table_data_size_minus1 + 1;
    used_bits = add_per_object_md(b_dynamic_object, b_object_not_active);
    remain_bits = 8 * atd_size - used_bits;
    add_table_data; .............................................................. remain_bits
  }
}
```

6.2.8.6 object_basic_info

Syntax	No of bits
object_basic_info()	
{	
b_default_basic_info_md;	1
if (b_default_basic_info_md == 0) {	
basic_info_md;	1/2
if (basic_info_md == 0b0 or basic_info_md == 0b10) {	
object_gain_code;	1/2
if (object_gain_code == 0b0) {	
object_gain_value;	6
}	
}	
if (basic_info_md == 0b10 or basic_info_md == 0b11) {	
object_priority_code;	5
}	
}	
}	

6.2.8.7 object_render_info

Syntax	No of bits
object_render_info(object_render_info_status, b_no_delta)	
{	
if (object_render_info_status == ALL_NEW) {	
b_obj_render_otherprops_present = 1;	
b_obj_render_zone_present = 1;	
b_obj_render_position_present = 1;	
}	
else {	
/* object_render_info_mask section */	
b_obj_render_otherprops_present;	1
b_obj_render_zone_present;	1
b_obj_render_position_present;	1
}	
if (b_obj_render_position_present) {	
if (b_no_delta) {	
b_diff_pos_coding = 0;	
}	
else {	
b_diff_pos_coding;	1
}	
if (b_diff_pos_coding) {	
diff_pos3D_X;	3
diff_pos3D_Y;	3
diff_pos3D_Z;	3
}	
else {	
pos3D_X;	6
pos3D_Y;	6
pos3D_Z_sign;	1
pos3D_Z;	4
}	
}	
if (b_obj_render_zone_present) {	
b_grouped_zone_defaults;	1
if (b_grouped_zone_defaults == 0) {	
group_zone_flag[];	3
if (group_zone_flag[2]) {	
zone_mask;	3
}	
if (group_zone_flag[1]) {	
b_enable_elevation = 0;	
}	
if (group_zone_flag[0]) {	
b_object_snap = 1;	
}	
}	
}	
if (b_obj_render_otherprops_present) {	

```
Syntax                                                                No of bits
    b_grouped_other_defaults; ........................................ 1
    if (b_grouped_other_defaults == 0) {
      group_other_mask; .............................................. 4
      if (group_other_mask & 0b0001) {
        object_width_mode; ........................................... 1
        if (object_width_mode == 0) {
          object_width_code; ......................................... 5
        }
        else {
          object_width_X_code; ....................................... 5
          object_width_Y_code; ....................................... 5
          object_width_Z_code; ....................................... 5
        }
      }
      if (group_other_mask & 0b0010) {
        object_screen_factor_code; ................................... 3
        object_depth_factor; ......................................... 2
      }
      else {
        object_screen_factor_code = 0;
      }
      if (group_other_mask & 0b0100) {
        b_obj_at_infinity; ........................................... 1
        if (b_obj_at_infinity) {
          obj_distance = inf;
        }
        else {
          obj_distance_factor_code; .................................. 4
        }
      }
      if (group_other_mask & 0b1000) {
        object_div_mode; ............................................. 2
        if (object_div_mode == 0b00) {
          object_div_table; .......................................... 2
        }
        else {
          if (object_div_mode & 0b10) {
            object_div_code; ......................................... 6
          }
        }
      }
    }
  }
}
```

6.2.8.8 bed_render_info

```
Syntax                                                                No of bits
bed_render_info()
{
  b_bed_render_info; ................................................. 1
  if (b_bed_render_info) {
    b_stereo_dmx_coeff; .............................................. 1
    if (b_stereo_dmx_coeff) {
      stereo_dmx_coeff();
    }
    b_cdmx_data_present; ............................................. 1
    if (b_cdmx_data_present) {
      b_cdmx_w_to_f; ................................................. 1
      if (b_cdmx_w_to_f) {
        gain_w_to_f_code; ............................................ 3
      }
      b_cdmx_b4_to_b2; ............................................... 1
      if (b_cdmx_b4_to_b2) {
        gain_b4_to_b2_code; .......................................... 3
      }
      b_tm_ch_present; ............................................... 1
      if (b_tm_ch_present) {
        b_cdmx_t2_to_f_s_b; .......................................... 1
        if (b_cdmx_t2_to_f_s_b) {
          tool_t2_to_f_s_b();
        }
        b_cdmx_t2_to_f_s; ............................................ 1
        if (b_cdmx_t2_to_f_s) {
```

Syntax	No of bits
`      tool_t2_to_f_s();`	
`    }`	
`  }`	
`  **b_tb_ch_present**;`	1
`  if (b_tb_ch_present) {`	
`    **b_cdmx_tb_to_f_s_b**;`	1
`    if (b_cdmx_tb_to_f_s_b) {`	
`      tool_tb_to_f_s_b();`	
`    }`	
`    **b_cdmx_tb_to_f_s**;`	1
`    if (b_cdmx_tb_to_f_s) {`	
`      tool_tb_to_f_s();`	
`    }`	
`  }`	
`  **b_tf_ch_present**;`	1
`  if (b_tf_ch_present) {`	
`    **b_cdmx_tf_to_f_s_b**;`	1
`    if (b_cdmx_tf_to_f_s_b) {`	
`      tool_tf_to_f_s_b();`	
`    }`	
`    **b_cdmx_tf_to_f_s**;`	1
`    if (b_cdmx_tf_to_f_s) {`	
`      tool_tf_to_f_s();`	
`    }`	
`  }`	
`  if (b_tb_ch_present || b_tf_ch_present) {`	
`    **b_cdmx_tfb_to_tm**;`	1
`    if (b_cdmx_tfb_to_tm) {`	
`      **gain_tfb_to_tm_code**;`	3
`    }`	
`  }`	
`}`	
`}`	
`}`	

6.2.8.8a stereo_dmx_coeff

Syntax	No of bits
stereo_dmx_coeff()	
{	
loro_centre_mixgain;	3
loro_surround_mixgain;	3
b_ltrt_mixinfo;	1
if ((b_ltrt_mixinfo) == (1)) {	
ltrt_centre_mixgain;	3
ltrt_surround_mixgain;	3
}	
b_lfe_mixinfo;	1
if ((b_lfe_mixinfo) == (1)) {	
lfe_mixgain;	5
}	
preferred_dmx_method;	2
}	

6.2.8.9 trim

Syntax	No of bits
trim()	
{	
b_trim_present;	1
if (b_trim_present) {	
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) {	

Syntax	No of bits

```
            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
            }
          }
        }
      }
    }
  }
}
```

6.2.8.9a headphone

Syntax	No of bits

```
headphone()
{
  b_headphone; .............................................................. 1
  if (b_headphone) {
    hp_operation_mode; ...................................................... 3
    if ((hp_operation_mode == 0b001) || (hp_operation_mode == 0b010)) {
      b_head_track_disable_all; ............................................. 1
    }
  }
}
```

6.2.8.10 add_per_object_md

Syntax	No of bits

```
add_per_object_md(b_object_not_active, b_dynamic_object)
{
  b_obj_trim_disable; ....................................................... 1
  if (b_object_not_active == 0) {
    if (b_dynamic_object) {
      b_ext_prec_pos; ....................................................... 1
      if (b_ext_prec_pos) {
        ext_prec_pos();
      }
    }
  }
  b_headphone; .............................................................. 1
  if (b_headphone == 1) {
    hp_render_mode_obj; ..................................................... 2
    b_head_track_disable_obj; ............................................... 1
  }
}
```

6.2.8.11 ext_prec_pos

Syntax	No of bits

```
ext_prec_pos()
{
  ext_prec_pos_presence[]; .................................................. 3
  if (ext_prec_pos_presence[2]) {
```

Syntax	No of bits
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
}	
}	

6.2.8.12 ext_prec_alt_pos

Syntax	No of bits
ext_prec_alt_pos(n_objs, b_keep, obj_type[n_objs], b_lfe[n_objs])	
{	
if (b_keep == 0) {	
for (obj = 0; obj < n_objs; obj++) {	
if ((obj_type[obj] == DYN) && (b_lfe[obj] == 0)) {	
b_ext_prec_alt_pos;	1
if (b_ext_prec_alt_pos) {	
ext_prec_pos();	
}	
}	
}	
}	
}	

6.2.8.13 tool_tb_to_f_s_b

Syntax	No of bits
tool_tb_to_f_s_b()	
{	
b_top_back_to_front;	1
if (b_top_back_to_front == 1) {	
gain_t2d_code;	3
gain_t2e_code = 7;	
}	
else {	
b_top_back_to_side;	1
if (b_top_back_to_side == 1) {	
gain_t2e_code;	3
}	
else {	
gain_t2f_code;	3
gain_t2e_code = 7;	
}	
}	
}	

6.2.8.14 tool_tb_to_f_s

Syntax	No of bits
tool_tb_to_f_s()	
{	
b_top_back_to_front;	1
if (b_top_back_to_front == 1) {	
gain_t2d_code;	3
gain_t2e_code = 7;	
}	
else {	
gain_t2e_code;	3
}	
}	

6.2.8.15 tool_tf_to_f_s_b

Syntax	No of bits
tool_tf_to_f_s_b()	
{	
b_top_front_to_front;	1
if (b_top_front_to_front == 1) {	
gain_t2a_code;	3
gain_t2b_code = 7;	
}	
else {	
b_top_front_to_side;	1
if (b_top_front_to_side == 1) {	
gain_t2b_code;	3
}	
else {	
gain_t2c_code;	3
gain_t2b_code = 7;	
}	
}	
}	

6.2.8.16 tool_tf_to_f_s

Syntax	No of bits
tool_tf_to_f_s()	
{	
b_top_front_to_front;	1
if (b_top_front_to_front == 1) {	
gain_t2a_code;	3
gain_t2b_code = 7;	
}	
else {	
gain_t2b_code;	3
}	
}	

6.2.9 Presentation data

6.2.9.1 loud_corr

Syntax	No of bits

```
loud_corr(pres_ch_mode, pres_ch_mode_core, b_objects)
{
  b_obj_loud_corr = 0;
  if (b_objects == 1) {
    b_obj_loud_corr; ............................................................. 1
  }
  if (pres_ch_mode > 4 or b_obj_loud_corr == 1) {
    b_corr_for_immersive_out; .................................................... 1
  }
  if (pres_ch_mode > 1 or b_obj_loud_corr == 1) {
    b_loro_loud_comp; ............................................................ 1
    if (b_loro_loud_comp == 1) {
      loro_dmx_loud_corr; ........................................................ 5
    }
    b_ltrt_loud_comp; ............................................................ 1
    if (b_ltrt_loud_comp == 1) {
      ltrt_dmx_loud_corr; ........................................................ 5
    }
  }
  if (pres_ch_mode > 4 or b_obj_loud_corr == 1) {
    b_loud_comp; ................................................................. 1
    if (b_loud_comp == 1) {
      loud_corr_5_X; ............................................................. 5
    }
    if (b_corr_for_immersive_out == 1) {
      b_loud_comp; ............................................................... 1
      if (b_loud_comp == 1) {
        loud_corr_5_X_2; ......................................................... 5
      }
      b_loud_comp; ............................................................... 1
      if (b_loud_comp == 1) {
        loud_corr_7_X; ........................................................... 5
      }
    }
  }
  if (pres_ch_mode > 10 or b_obj_loud_corr == 1) {
    if (b_corr_for_immersive_out == 1) {
      b_loud_comp; ............................................................... 1
      if (b_loud_comp == 1) {
        loud_corr_7_X_4; ......................................................... 5
      }
      b_loud_comp; ............................................................... 1
      if (b_loud_comp == 1) {
        loud_corr_7_X_2; ......................................................... 5
      }
      b_loud_comp; ............................................................... 1
      if (b_loud_comp == 1) {
        loud_corr_5_X_4; ......................................................... 5
      }
    }
  }
  if (pres_ch_mode_core >= 5) {
    b_loud_comp; ................................................................. 1
    if (b_loud_comp == 1) {
      loud_corr_core_5_X_2; ...................................................... 5
    }
  }
  if (pres_ch_mode_core >= 3) {
    b_loud_comp; ................................................................. 1
    if (b_loud_comp == 1) {
      loud_corr_core_5_X; ........................................................ 5
    }
    b_loud_comp; ................................................................. 1
    if (b_loud_comp == 1) {
      loud_corr_core_loro; ....................................................... 5
      loud_corr_core_ltrt; ....................................................... 5
    }
  }
  if (b_obj_loud_corr == 1) {
    b_loud_comp; ................................................................. 1
    if (b_loud_comp == 1) {
```

Syntax	No of bits
``` loud_corr_9_X_4; ```	5
``` } } } ```	

6.2.9.2 custom_dmx_data

```
Syntax                                                                 No of bits
custom_dmx_data(pres_ch_mode, pres_ch_mode_core, b_pres_4_back_channels_present,
pres_top_channel_pairs, b_pres_has_lfe)
{
  bs_ch_config = -1;
  if (pres_ch_mode in [11, 12, 13, 14]) {
    if (pres_top_channel_pairs == 2) {
      if (pres_ch_mode >= 13 and b_pres_4_back_channels_present == 1) {
        bs_ch_config = 0;
      }
      if (pres_ch_mode <= 12) {
        if (b_pres_4_back_channels_present == 1) {
          bs_ch_config = 1;
        }
        else {
          bs_ch_config = 2;
        }
      }
    }
    if (pres_top_channel_pairs == 1) {
      if (pres_ch_mode >= 13 and b_pres_4_back_channels_present == 1) {
        bs_ch_config = 3;
      }
      if (pres_ch_mode <= 12) {
        if (b_pres_4_back_channels_present == 1) {
          bs_ch_config = 4;
        }
        else {
          bs_ch_config = 5;
        }
      }
    }
  }
  if (bs_ch_config >= 0) {
    b_cdmx_data_present; ................................................ 1
    if (b_cdmx_data_present == 1) {
      n_cdmx_configs_minus1; ............................................ 2
      n_cdmx_configs = n_cdmx_configs_minus1 + 1;
      for (dc = 0; dc < n_cdmx_configs; dc++) {
        if (bs_ch_config == 2 or bs_ch_config == 5) {
          out_ch_config[dc]; ............................................ 1
        }
        else {
          out_ch_config[dc]; ............................................ 3
        }
        cdmx_parameters(bs_ch_config, out_ch_config[dc]);
      }
    }
  }
  if (pres_ch_mode >= 3 or pres_ch_mode_core >= 3) {
    b_stereo_dmx_coeff; ................................................. 1
    if (b_stereo_dmx_coeff == 1) {
      loro_centre_mixgain; .............................................. 3
      loro_surround_mixgain; ............................................ 3
      b_ltrt_mixinfo; ................................................... 1
      if (b_ltrt_mixinfo == 1) {
        ltrt_centre_mixgain; ............................................ 3
        ltrt_surround_mixgain; .......................................... 3
      }
      if (b_pres_has_lfe == 1) {
        b_lfe_mixinfo; .................................................. 1
        if (b_lfe_mixinfo == 1) {
          lfe_mixgain; .................................................. 5
        }
      }
      preferred_dmx_method; ............................................. 2
    }
```

Syntax	No of bits
} }	

6.2.9.3 cdmx_parameters

Syntax	No of bits

```
cdmx_parameters(bs_ch_config, out_ch_config)
{
  if (bs_ch_config == 0 or bs_ch_config == 3) {
    tool_scr_to_c_l();
  }
  if (bs_ch_config < 2) {
    switch (out_ch_config) {
      case 0:
        tool_t4_to_f_s();
        tool_b4_to_b2();
        break;
      case 1:
        tool_t4_to_t2();
        tool_b4_to_b2();
        break;
      case 2:
        tool_b4_to_b2();
        break;
      case 3:
        tool_t4_to_f_s_b();
        break;
      case 4:
        tool_t4_to_t2();
        break;
    }
  }
  if (bs_ch_config == 2) {
    switch (out_ch_config) {
      case 0:
        tool_t4_to_f_s();
        break;
      case 1:
        tool_t4_to_t2();
        break;
    }
  }
  if (3 <= bs_ch_config <= 4) {
    switch (out_ch_config) {
      case 0:
        tool_t2_to_f_s();
        tool_b4_to_b2();
        break;
      case 1:
        tool_b4_to_b2();
        break;
      case 2:
        tool_b4_to_b2();
        break;
      case 3:
        tool_t2_to_f_s_b();
        break;
    }
  }
  if (bs_ch_config == 5) {
    switch (out_ch_config) {
      case 0:
        tool_t2_to_f_s();
        break;
    }
  }
}
```

6.2.9.4 tool_scr_to_c_l

Syntax	No of bits
`tool_scr_to_c_l()`	
`{`	
`  b_put_screen_to_c;`	1
`  if (b_put_screen_to_c == 1) {`	
`    gain_f1_code;`	3
`  }`	
`  else {`	
`    gain_f2_code;`	3
`  }`	
`}`	

6.2.9.5 tool_b4_to_b2

Syntax	No of bits
`tool_b4_to_b2()`	
`{`	
`  gain_b_code;`	3
`}`	

6.2.9.6 tool_t4_to_t2

Syntax	No of bits
`tool_t4_to_t2()`	
`{`	
`  gain_t1_code;`	3
`}`	

6.2.9.7 tool_t4_to_f_s_b

Syntax	No of bits
`tool_t4_to_f_s_b()`	
`{`	
`  b_top_front_to_front;`	1
`  if (b_top_front_to_front == 1) {`	
`    gain_t2a_code;`	3
`    gain_t2b_code = 7;`	
`  }`	
`  else {`	
`    b_top_front_to_side;`	1
`    if (b_top_front_to_side == 1) {`	
`      gain_t2b_code;`	3
`    }`	
`    else {`	
`      gain_t2c_code;`	3
`      gain_t2b_code = 7;`	
`    }`	
`  }`	
`  b_top_back_to_front;`	1
`  if (b_top_back_to_front == 1) {`	
`    gain_t2d_code;`	3
`    gain_t2e_code = 7;`	
`  }`	
`  else {`	
`    b_top_back_to_side;`	1
`    if (b_top_back_to_side == 1) {`	
`      gain_t2e_code;`	3
`    }`	
`    else {`	
`      gain_t2f_code;`	3
`      gain_t2e_code = 7;`	
`    }`	
`  }`	
`}`	

6.2.9.8 tool_t4_to_f_s

Syntax	No of bits

```
tool_t4_to_f_s()
{
  b_top_front_to_front; ................................ 1
  if (b_top_front_to_front == 1) {
    gain_t2a_code; ..................................... 3
    gain_t2b_code = 7;
  }
  else {
    gain_t2b_code; ..................................... 3
  }
  b_top_back_to_front; ................................. 1
  if (b_top_back_to_front == 1) {
    gain_t2d_code; ..................................... 3
    gain_t2e_code = 7;
  }
  else {
    gain_t2e_code; ..................................... 3
  }
}
```

6.2.9.9 tool_t2_to_f_s_b

Syntax	No of bits

```
tool_t2_to_f_s_b()
{
  b_top_to_front; ...................................... 1
  if (b_top_to_front == 1) {
    gain_t2a_code; ..................................... 3
    gain_t2b_code = 7;
  }
  else {
    b_top_to_side; ..................................... 1
    if (b_top_to_side == 1) {
      gain_t2b_code; ................................... 3
    }
    else {
      gain_t2c_code; ................................... 3
      gain_t2b_code = 7;
    }
  }
}
```

6.2.9.10 tool_t2_to_f_s

Syntax	No of bits

```
tool_t2_to_f_s()
{
  b_top_to_front; ...................................... 1
  if (b_top_to_front == 1) {
    gain_t2a_code; ..................................... 3
    gain_t2b_code = 7;
  }
  else {
    gain_t2b_code; ..................................... 3
  }
}
```

6.3 Description of bitstream elements

6.3.1 Introduction

The description of bitstream elements is analogous to the description in ETSI TS 103 190-1 [1]. Elements that have been described in ETSI TS 103 190-1 [1], clause 4.3, are not repeated in this clause unless their meaning has changed.

6.3.2 AC-4 frame information

6.3.2.1 ac4_toc - AC-4 table of contents

6.3.2.1.1 bitstream_version

This 2-bit code, which is extendable by `variable_bits()`, indicates the bitstream version. A decoder implemented according to the present document shall decode bitstreams where `bitstream_version` ≤ 2.

Table 51: bitstream_version

bitstream_version	Presentation information location	Description
0	`ac4_presentation_info()`	All presentations in the bitstream can be decoded by an AC-4 decoder conforming to [1] or to the present document.
1	`ac4_presentation_info()`	All presentations in the bitstream can be decoded by an AC-4 decoder conforming to the present document. Presentations with a presentation version value of 0, containing channel-based audio up to 7.1, can be decoded by an AC-4 decoder conforming to [1].
2	`ac4_presentation_v1_info()` and `ac4_substream_group_info()`	All presentations in the bitstream can be decoded by an AC-4 decoder conforming to the present document. The bitstream is not decodable by an AC-4 decoder conforming to [1].

6.3.2.1.2 br_code

The `br_code` value, in conjunction with the `wait_frames` value, supports accurate determination of the long-term bit rate for average bit-rate streams.

For average bit-rate streams, the `br_code` value of 0b11 is followed by a sequence of frames with `br_code` values different from 0b11. That sequence of values can be used to improve the bit-rate estimation compared to simple averaging of the size of the analysed frames. An algorithm for calculating the signalled bit rate is described in annex B.

Table 52: br_code

br_code	Description
0b00	value(br_code) = 0
0b01	value(br_code) = 1
0b10	value(br_code) = 2
0b11	Start-stop code for sequence of br_codes

6.3.2.1.3 b_iframe_global

This Boolean indicates whether the first substream in each presentation is encoded independently from preceding frames (i.e. there is no dependency over time).

The Booleans `b_audio_ndot`, `b_pres_ndot` and `b_oamd_ndot` indicate for each substream whether a substream is encoded independently from previous frames and there is no dependency over time for the corresponding substream.

The first transmission frame in a group of `frame_rate_fraction` frames has `b_iframe_global` set to true if the first codec frame of each presentation contains only substream without dependency on time. If `frame_rate_fraction` > 1, every other transmission frame in the group has `b_iframe_global` set to false.

See also clause 4.5.3, and clause 5.1.3.

6.3.2.1.4 b_program_id

This Boolean indicates whether program identification data is present.

6.3.2.1.5 short_program_id

The `short_program_id` element holds the program identification as a 16-bit value.

6.3.2.1.6 b_program_uuid_present

This Boolean indicates whether program identification as a universally unique identifier (UUID) value, `program_uuid`, is present.

6.3.2.1.7 program_uuid

The `program_uuid` element holds the program identification as a 16-byte UUID value.

6.3.2.1.8 total_n_substream_groups

The value of this helper variable is *total_n_substream_groups* = 1 + *max_group_index*, where *max_group_index* is the maximum of the `group_index` values contained in all occurrences of `ac4_sgi_specifier()`.

6.3.2.2 ac4_presentation_v1_info - AC-4 presentation version 1 information

6.3.2.2.1 b_single_substream_group

This Boolean indicates whether a single substream group is present. In this case, the substream type of the referenced substream group shall be considered as a Main substream. Otherwise, the number of substream groups, *n_substream_groups*, and the associated substream types are determined using the value of `presentation_config`.

6.3.2.2.2 presentation_config

This 3-bit code, which is extendable by `variable_bits()`, indicates the presentation configuration for `presentation_version` = 1 as shown in table 53. The presentation configuration for `presentation_version` = 0 is specified in ETSI TS 103 190-1 [1], clause 4.3.3.3.4.

Table 53: presentation_config to substream type mapping for presentation_version = 1

`presentation_config`	substream type		
	1st substream group	2nd substream group	3rd substream group
0	music and effects substream	dialogue substream	N/A
1	Main substream	dialogue enhancement substream	N/A
2	Main substream	associated audio substream	N/A
3	music and effects substream	dialogue substream	associated audio substream
4	Main substream	dialogue enhancement substream	associated audio substream
5	substream types are determined by `content_classifier` of each substream group according to table 54 - see note		
6	Extensible Metadata Delivery Format (EMDF)substream(s)		
≥ 7	Reserved		
NOTE: The number of substream groups is arbitrary			

For presentation_config = 5, substream types are determined by content_classifier of each substream group according to table 54.

Table 54: content_classifier to substream type mapping for presentation_config = 5

content_classifier	Content Classification (informative)	substream type
000	complete main	main or music and effects
001	music and effects	main or music and effects
010	visually impaired	associated audio
011	hearing impaired	associated audio
100	dialogue	dialogue
101	commentary	associated audio
110	emergency	main or music and effects
111	voice over	main or music and effects

6.3.2.2.3 md_compat

This field indicates the decoder compatibility as shown in table 55.

The md_compat element indicates decoder systems that are compatible with a presentation.

A decoder system with compatibility level *n* shall be able to decode all presentations with md_compat $\leq n$ and:

- presentation_version = 0 as defined in ETSI TS 103 190-1 [1], clause 4.3.3.3.8; and
- presentation_version = 1 as defined in table 55,

A system with compatibility level *n* shall not decode (i.e. select) presentations with md_compat $> n$.

NOTE: A presentation may consist of several substreams, which may be distributed over several elementary streams.

Table 55: md_compat for presentation_version = 1

md_compat	Maximum number of tracks (see note) if presentation includes		Maximum input channel configuration for channel-based immersive	Maximum reconstructed A-JOC objects
	no object audio	object audio		
0	2	N/A	N/A	N/A
1	6	6	N/A	15.1
2	9	8	7.1.4	15.1
3	11	10	7.1.4	17.1
4-6	Reserved			
7	Unrestricted			
NOTE: "Number of tracks" refers to the total number of audio objects and channels in all substreams contributing to the presentation, with the exception of LFE channels that have the b_lfe flag set in mono_data() structure.				

6.3.2.2.4 b_presentation_id

This Boolean indicates whether the presentation carries a presentation_id that uniquely identifies a presentation within this table of contents.

6.3.2.2.4a presentation_id

This unsigned integer uniquely identifies a presentation.

NOTE: A previous version of the present document used the term "presentation_group_index" instead of presentation_id.

6.3.2.2.5 b_presentation_filter

This Boolean indicates whether the b_enable_presentation Boolean is present.

6.3.2.2.6 b_enable_presentation

This Boolean indicates whether a presentation is enabled.

6.3.2.2.7 b_multi_pid

This Boolean indicates whether the presentation is contained in multiple packet identifier (PID) identified elementary streams.

6.3.2.2.8 n_substream_groups_minus2

The `n_substream_groups_minus2` element indicates the number of substream groups minus 2. To get the number of substream groups, *n_substream_groups*, a value of 2 needs to be added to `n_substream_groups_minus2`.

6.3.2.3 presentation_version - presentation version information

6.3.2.3.1 b_tmp

The `b_tmp` element, which might be present multiple times, is used to signal the version of the presentation. A presentation version value of 0 indicates a presentation that is decodable by an AC-4 decoder conforming to ETSI TS 103 190-1 [1] or to the present document. A decoder implemented in accordance with the present document shall decode a presentation if its presentation version is 1 or 2. A decoder implemented in accordance with the present document shall skip the `ac4_presentation_info()` or `ac4_presentation_v1_info()` if the version of the presentation is not 0, 1 or 2.

6.3.2.4 frame_rate_fractions_info - frame rate fraction information

6.3.2.4.1 b_frame_rate_fraction

This Boolean indicates whether the variable *frame_rate_fraction* is set to a value greater than 1.

6.3.2.4.2 b_frame_rate_fraction_is_4

This Boolean indicates whether the variable *frame_rate_fraction* is set to a value of 4.

6.3.2.5 ac4_substream_group_info - AC-4 substream group information

6.3.2.5.1 b_substreams_present

This Boolean indicates whether a substream group contains substreams.

6.3.2.5.2 n_lf_substreams_minus2

The `n_lf_substreams_minus2` element indicates the number of substreams referenced through either `ac4_substream_info_chan()`, `ac4_substream_info_ajoc()` or `ac4_substream_info_obj()` elements in the respective substream group.

The total number of these substreams is given by *n_lf_substreams*. To get this helper variable, a value of 2 needs to be added to `n_lf_substreams_minus2`. Additional `variable_bits()` are used to derive values of *n_lf_substreams* exceeding 4.

6.3.2.5.3 b_channel_coded

This Boolean indicates whether the substreams contain channel-based audio.

6.3.2.5.4 sus_ver

This bit indicates the substream version for bitstreams with bitstream_version = 1. A value of 0 identifies a channel-based audio substream using a bitstream syntax for ac4_substream(), which is compatible to the syntax defined in ETSI TS 103 190-1 [1]. A value of 1 indicates the usage of an extended syntax for the AC-4 substream. If sus_ver is not set, the default value is 0.

6.3.2.5.5 b_oamd_substream

This Boolean indicates whether a substream containing object audio metadata is present.

6.3.2.5.6 b_ajoc

This Boolean indicates whether advanced joint object coding is used.

6.3.2.6 ac4_sgi_specifier - AC-4 substream group information specifier

6.3.2.6.1 group_index

The group_index element, together with potential variable_bits(), indicates the substream group index of the related ac4_substream_group_info() element. The order of the referenced substream groups via ac4_sgi_specifier() elements within ac4_presentation_v1_info() is given by table 53.

EXAMPLE: For presentation_config = 4, the main substream group is indicated first, followed by the dialogue enhancement substream group and the associated audio substream group.

6.3.2.7 ac4_substream_info_chan - AC-4 substream information for channel based substreams

6.3.2.7.1 Introduction

The three fields b_4_channel_present, b_centre_present, and top_channels_present signal whether some of the channels as signalled by channel_mode are actually present in the original content (i.e. the audio signals fed to the AC-4 encoder during content creation) or not. This enables signalling and representing original content with channel configurations that do not utilize all channels as signalled by channel_mode. Channels that are not present in the original content contain encoded silence, and the decoder may choose not to decode them.

6.3.2.7.2 channel_mode

This variable length field indicates the channel mode and the variable *ch_mode*. The corresponding values for presentation_version = 1 are shown in table 56.

NOTE: The channel mode and the value *ch_mode* for presentation_version = 0 is shown in ETSI TS 103 190-1 [1], table 88.

Table 56: channel_mode

channel_mode	Channel mode	*ch_mode*
0b0	Mono	0
0b10	Stereo	1
0b1100	3.0	2
0b1101	5.0	3
0b1110	5.1	4
0b1111000	7.0: 3/4/0 (L, C, R, Ls, Rs, Lb, Rb)	5
0b1111001	7.1: 3/4/0.1 (L, C, R, Ls, Rs, Lb, Rb, LFE)	6
0b1111010	7.0: 5/2/0 (L, C, R, Lw, Rw, Ls, Rs)	7
0b1111011	7.1: 5/2/0.1 (L, C, R, Lw, Rw, Ls, Rs, LFE)	8
0b1111100	7.0: 3/2/2 (L, C, R, Ls, Rs, Tfl, Tfr)	9
0b1111101	7.1: 3/2/2.1 (L, C, R, Ls, Rs, Tfl, Tfr, LFE)	10
0b11111100	7.0.4	11
0b11111101	7.1.4	12
0b111111100	9.0.4	13
0b111111101	9.1.4	14
0b111111110	22.2	15
0b111111111…	reserved	≥ 16

6.3.2.7.3 b_4_back_channels_present

When the back channels (Lb, Rb) are present in the bitstream as signalled by table 56, this fixed-length field signals whether those channels are present in the original content or only contain encoded silence as shown in table 57.

Table 57: b_4_back_channels_present

b_4_back_channels_present	Original content	Encoded AC-4 bitstream
False	Original content contains two surround channels: Ls, Rs	Surround channels of the original content are carried in Ls, Rs; Lb, Rb contain silence
True	Original content contains four surround channels: Ls, Rs, Lb, Rb	Ls, Rs, Lb, Rb contain original content

6.3.2.7.4 b_centre_present

When the Centre channel is present in the bitstream as signalled by table 56, this fixed-length field signals whether it actually is present in the original content or contains encoded silence as shown in table 58.

Table 58: b_centre_present

b_centre_present	Original content	Encoded AC-4 bitstream
False	Original content does not contain a Centre channel	C contains silence
True	Original content contains a Centre channel	C contains original content

6.3.2.7.5 top_channels_present

When the top channels (Tfl, Tbl, Tfr, Tbr) are present in the bitstream as signalled by table 56, this fixed-length field signals the following information, as shown in table 59:

- whether all of those channels are present in the original content;
- whether the original content has only two top channels (Tsl, Tsr) and how they are carried; or
- or whether these channels contain encoded silence.

Table 59: top_channels_present

`top_channels_present`	Original content	Encoded AC-4 bitstream
0	Original content does not contain any of the channels Tfl, Tfr, Tbl, Tbr	Tfl, Tfr, Tbl, Tbr contain silence
1	Original content contains two top channels: Tsl and Tsr	Original content of Tsl, Tsr is carried in Tfl, Tfr; Tbl, Tbr contain silence
2	Original content contains two top channels: Tsl and Tsr	Original content of Tsl, Tsr is carried in Tbr, Tbl; Tfl, Tfr contain silence
3	Original content contains four top channels: Tfl, Tfr, Tbl, Tbr	Tfl, Tfr, Tbl, Tbr contain original content

6.3.2.7.6 b_audio_ndot

This Boolean indicates whether an audio substream can be decoded independently of preceding frames.

6.3.2.8 ac4_substream_info_ajoc - object type information for A-JOC coded substreams

6.3.2.8.1 Introduction

The `ac4_substream_info_ajoc` element is used when objects in the substream are A-JOC coded. It contains information about the number, types and positions of objects contained in a substream, both for core decoding and for full decoding.

An A-JOC coded substream can contain a mixture of dynamic objects, and static objects. The latter are either intermediate spatial format objects, or bed objects. Whenever a substream contains a mixture, the static objects precede the dynamic objects. The number of static objects can be derived from the object position properties (`ac4_substream_info_obj` or `bed_dyn_obj_assignment`); the number of dynamic objects is the difference to `n_objects`, the total number of objects in the substream. If the derived number of static objects is larger than `n_objects`, behaviour is undefined.

Clause 6.3.2.10.1 provides further details about the make-up of objects in substream groups.

6.3.2.8.2 b_lfe

This Boolean indicates whether an LFE channel is present in the set of dynamic objects.

NOTE: Contrary to direct coded objects, A-JOC coded bed objects cannot contain an LFE channel (neither LFE nor LFE2).

6.3.2.8.3 b_static_dmx

This Boolean indicates whether the A-JOC core decode signal is a static 5.0/5.1 bed.

6.3.2.8.4 n_fullband_dmx_signals_minus1

The `n_fullband_dmx_signals_minus1` unsigned integer indicates the value *n_fullband_dmx_signals* of the number of fullband signals in the core decode downmix. The value of *n_fullband_dmx_signals* is determined by the following equation:

$$\text{n\_fullband\_dmx\_signals} = \text{n\_fullband\_dmx\_signals\_minus1} + 1$$

6.3.2.8.5 b_oamd_common_data_present

This Boolean indicates whether OAMD common data is present.

6.3.2.8.6 n_fullband_upmix_signals_minus1

The `n_fullband_upmix_signals_minus1` unsigned integer indicates the value *n_fullband_upmix_signals* of the number of fullband signals in the full decode mode. The value of *n_fullband_upmix_signals* is determined by the following equation:

$$\text{n\_fullband\_upmix\_signals} = \text{n\_fullband\_upmix\_signals\_minus1} + 1$$

6.3.2.8.7 bed_dyn_obj_assignment - bed and dynamic object assignment

The `bed_dyn_obj_assignment` element is used when the objects in the substream are A-JOC coded. It contains information about the types and positions of objects contained in a substream.

Clause 6.3.2.10.8 provides information about the interpretation of elements in `bed_dyn_obj_assignment`.

6.3.2.9 AC-4 substream information for object based substreams using A-JOC

Please see clause 6.3.2.10.8.

6.3.2.10 ac4_substream_info_obj - object type information for direct-coded substreams

6.3.2.10.1 Introduction

The `ac4_substream_info_obj` element is used when objects in the substream are direct-coded. It contains information about the number, types and positions of objects contained in a substream.

6.3.2.10.2 n_objects_code

The `n_objects_code` codeword indicates the total number of objects (both dynamic and static) in the substream as per table 60.

Table 60: n_objects_code

n_objects_code	n_objects (Number of objects)
0	`b_lfe`
1	1+`b_lfe`
2	2+`b_lfe`
3	3+`b_lfe`
4	5+`b_lfe`
5…7	reserved

6.3.2.10.3 b_dynamic_objects and b_dyn_objects_only

These Booleans indicate whether the substream contains only dynamic objects.

6.3.2.10.4 b_lfe

This Boolean indicates whether an LFE channel is present. In this case, an LFE channel is part of the set of dynamic objects. Otherwise, the presence of an LFE in the set of static objects can be signalled by `bed_chan_assign_code`, `nonstd_bed_channel_assignment_flag[]`, or `std_bed_channel_assignment_flag[]`.

6.3.2.10.5 b_bed_objects

This Boolean indicates whether the substream contains bed objects.

6.3.2.10.6 b_bed_start

Beds and bed information can be split across several direct coded substreams.

This Boolean indicates whether the bed in this substream is independent. In this case, the bed and bed information contained in this substream are not an extension of previously transmitted information. Otherwise, they extend the bed transmitted in a previous substream.

NOTE: If the bed contains only one LFE channel, it will always be part of the first substream. If two LFE channels exist in the same bed, they are always split across two substreams, and the first substream contains LFE while the second substream contains LFE2.

6.3.2.10.7 b_isf_start

intermediate spatial format objects and ISF information can be distributed across several substreams.

This Boolean indicates whether the intermediate spatial format objects are independent. In this case, the intermediate spatial format objects and information contained in this substream are not an extension of the ISF objects and information transmitted in a previous substream. Otherwise, they extend previously transmitted information.

6.3.2.10.8 Interpreting object position properties

6.3.2.10.8.1 ISF object position signalling

The substream consists of intermediate spatial format objects when the `b_isf` flag is set to true, and the `isf_config` field indicates the intermediate spatial format of the objects in the substream group.

The `isf_config` field shall be interpreted according to table 61. The intermediate spatial format objects are ordered as presented in the table.

Table 61: isf_config

isf_config	Objects present in the ISF (in M.U.L.Z layer order)	Object configuration description	Number of objects
0b000	M1 M2 M3 U1	SR3.1.0.0	4
0b001	M1 M2 M3 M4 M5 U1 U2 U3	SR5.3.0.0	8
0b010	M1 M2 M3 M4 M5 M6 M7 U1 U2 U3	SR7.3.0.0	10
0b011	M1 M2 M3 M4 M5 M6 M7 M8 M9 U1 U2 U3 U4 U5	SR9.5.0.0	14
0b100	M1 M2 M3 M4 M5 M6 M7 U1 U2 U3 U4 U5 L1 L2 L3	SR7.5.3.0	15
0b101	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	SR15.9.5.1	30

6.3.2.10.8.2 Speaker anchored (bed) object position signalling

If neither the Boolean `b_dyn_objects_only` nor the Boolean `b_isf` is true, the substream consists of bed objects (that is, it constitutes a bed).

NOTE 1: In an A-JOC coded substream, the object type and object position are signalled independently for the core decoding mode and the full decoding mode in `bed_dyn_obj_assignment`; in a substream that is not advanced joint object coding (A-JOC) coded, they are signalled in `ac4_substream_info_obj`.

NOTE 2: Contrary to direct coded objects, A-JOC coded bed objects cannot contain an LFE. Any occurrence of LFE in the bed signalling is ignored. Instead, specify an LFE as part of the dynamic objects by setting the `b_lfe` flag in `ac4_substream_info_ajoc` to true.

Consecutive objects in the substream are assigned to speakers in the order that the speaker labels appear in the tables of the present clause.

EXAMPLE 1: If the substream contains a bed object destined for the Left channel and one object destined for the Right channel, then they appear in this order in the substream always.

NOTE 3: The number of objects in the substream and the number of bed assignments can be different. If the numbers are different, the behaviour is undefined.

There are several ways in which the location of the objects (that is, the configuration of the bed) can be signalled.

Signalling by channel assignment code

bed_chan_assign_code determines the speaker configuration that the objects are assigned to, as specified in table 62 and table 63.

Table 62: bed_chan_assign_code for direct coded objects

bed_chan_assign_code	Speakers	Channel configuration description	Number of channels
0	L, R	2.0.0	2
1	L, R, C	3.0.0	3
2	L, R, C, LFE, Ls, Rs	5.1.0	6
3	L, R, C, LFE, Ls, Rs, Tsl, Tsr	5.1.2	8
4	L, R, C, LFE, Ls, Rs, Tfl, Tfr, Tbl, Tbr	5.1.4	10
5	L, R, C, LFE, Ls, Rs, Lb, Rb	7.1.0	8
6	L, R, C, LFE, Ls, Rs, Lb, Rb, Tsl, Tsr	7.1.2	10
7	L, R, C, LFE, Ls, Rs, Lb, Rb, Tfl, Tfr, Tbl, Tbr	7.1.4	12

Table 63: bed_chan_assign_code for A-JOC coded substream

bed_chan_assign_code	Speakers	Channel configuration description	Number of channels
0	L, R	2.0.0	2
1	L, R, C	3.0.0	3
2	L, R, C, Ls, Rs	5.0.0	5
3	L, R, C, Ls, Rs, Tsl, Tsr	5.0.2	7
4	L, R, C, Ls, Rs, Tfl, Tfr, Tbl, Tbr	5.0.4	9
5	L, R, C, Ls, Rs, Lb, Rb	7.0.0	7
6	L, R, C, Ls, Rs, Lb, Rb, Tsl, Tsr	7.0.2	9
7	L, R, C, Ls, Rs, Lb, Rb, Tfl, Tfr, Tbl, Tbr	7.0.4	11

Signalling by channel assignment flags

The Boolean b_nonstd_bed_channel_assignment_flags_present indicates whether a non-standard or standard bed channel assignment is present.

In this case, a nonstd_bed_channel_assignment_flag[] array is present. The Boolean elements of nonstd_bed_channel_assignment_flag[] indicate an assignment of a speaker configuration to an object as per table 64.

Otherwise, a std_bed_channel_assignment_flag[] array is present. The Boolean elements of std_bed_channel_assignment_flag[] indicate an assignment of a speaker configuration to one or two objects as per table 65.

Array positions, starting at zero, are used as index into the table. The assignment proceeds by channel order, as indicated in the table, to the bed objects from first to last object.

Table 64: nonstd_bed_channel_assignment_flag[]

Array position	Channel order	Channel
16	0	L
15	1	R
14	2	C
13	3	LFE (see note)
12	4	Ls
11	5	Rs
10	6	Lb
9	7	Rb
8	8	Tfl
7	9	Tfr
6	10	Tsl
5	11	Tsr
4	12	Tbl
3	13	Tbr
2	14	Lw
1	15	Rw
0	16	LFE2 (see note)
NOTE: not applicable for A-JOC coded bed objects.		

NOTE 4: Table 65 can assign two objects at the same time. In that case, the Channel entry denotes *first*/*second* channel, separated with a slash (/).

Table 65: std_bed_channel_assignment_flag[]

Array position	Channel order	Channel	Number of channels
9	0	L/R	2
8	1	C	1
7	2	LFE (see note)	1
6	3	Ls/Rs	2
5	4	Lb/Rb	2
4	5	Tfl/Tfr	2
3	6	Tsl/Tsr	2
2	7	Tbl/Tbr	2
1	8	Lw/Rw	2
0	9	LFE2 (see note)	1
NOTE: not applicable for A-JOC coded bed objects.			

Signalling by individual channel assignment

In an A-JOC coded substream, the channels can be indicated by a list of integer values.

The Boolean `b_channel_assignment_flags_present` indicates whether channels are indicated by flags or a list of integers.

If this case, channels are indicated by flags as per **Signalling by channel assignment flags** above (see table 64 and table 65).

Otherwise, a list of `nonstd_bed_channel_assignment` integers indicate individual assignments, as per table 66.

Table 66: nonstd_bed_channel_assignment

nonstd_bed_channel_assignment	Channel
0	L
1	R
2	C
3	reserved (see note)
4	Ls
5	Rs
6	Lb
7	Rb
8	Tfl
9	Tfr
10	Tsl
11	Tsr
12	Tbl
13	Tbr
14	Lw
15	Rw
NOTE: LFE not applicable for A-JOC coded bed objects.	

EXAMPLE 2: In a substream that is not A-JOC coded, A 5.1 channel configuration could be signalled as either `bed_chan_assign_code`=2, or as `std_bed_channel_assignment_flag[]`=(0,0,0,0,0,0,1,1,1,1), or as `nonstd_bed_channel_assignment_flag[]`=(0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1).

EXAMPLE 3: In an A-JOC coded substream, A 5.1 channel configuration could be signalled as either `bed_chan_assign_code`=2, or by sending `nonstd_bed_channel_assignment` values of 0,1,2,4,5, or as `std_bed_channel_assignment_flag[]`=(0,0,0,0,0,0,1,0,1,1), or as `nonstd_bed_channel_assignment_flag[]`=(0,0,0,0,0,0,0,0,0,0,0,1,1,0,1,1,1); in addition to setting the `b_lfe` flag to true.

6.3.2.10.9 res_bytes

This element specifies the size of the `reserved_data` element in bytes.

6.3.2.10.10 reserved_data

The `reserved_data` element holds additional data and is reserved for future use.

6.3.2.11 ac4_presentation_substream_info - presentation substream information

6.3.2.11.1 b_alternative

This Boolean indicates whether an alternative presentation is present.

6.3.2.11.2 b_pres_ndot

This Boolean indicates whether a presentation substream can be decoded independently from preceding frames.

6.3.2.12 oamd_substream_info - object audio metadata substream information

6.3.2.12.1 b_oamd_ndot

This Boolean indicates whether an OAMD substream can be decoded independently from preceding frames.

6.3.3 AC-4 substreams

6.3.3.1 ac4_presentation_substream - AC-4 presentation substream

6.3.3.1.1 b_name_present

This Boolean indicates whether a presentation name is present.

6.3.3.1.2 b_length

This Boolean indicates whether the length of the presentation name is transmitted as `name_len`. Otherwise, the length of the `presentation_name` field is 32 bytes.

6.3.3.1.3 name_len

The `name_len` element indicates the length of the presentation name element in bytes.

6.3.3.1.4 presentation_name

The `presentation_name` element indicates the name of the presentation as a string using UTF-8 coding (ISO/IEC 10646 [2]).

The decoder shall read an array *byte* with `name_len` elements out of `presentation_name`, where each element has the length of one byte. If *byte*[`name_len`-1] = 0, the name of the presentation is given by *byte*[0] to *byte*[`name_len`-2]; otherwise, the name of the presentation is serialized into multiple chunks, each transmitted in one codec frame. If *byte*[`name_len`-2] = 0, the currently received chunk is the last chunk of the serialized name of the presentation. Out of this last chunk the decoder shall read the total number of chunks from *byte*[`name_len`-1] (as unsigned integer). The decoder shall store the received chunks until the total number of chunks is received and the full name of the presentation can be accumulated.

6.3.3.1.5 n_targets_minus1

The `n_targets_minus1` element indicates the number of presentation targets minus 1. To get the number of presentation targets, *n_targets*, a value of 1 needs to be added to `n_targets_minus1`. The result of additional `variable_bits` is added to *n_targets*, if *n_targets* = 4.

6.3.3.1.6 target_level

The `target_level` element indicates the decoder compatibility for a target.

NOTE: It is similar to the `md_compat` element that indicates the decoder compatibility for a presentation. See table 55 for more information.

6.3.3.1.7 target_device_category[]

The `target_device_category[]` is an array of Booleans as shown in table 67.

Table 67: target_device_category[]

target_device_category array index	Semantics if the Boolean is true
0	Target device category contains stereo speakers (1D)
1	Target device category contains 5.1 speakers (2D)
2	Target device category contains height speakers (3D)
3	Target device category contains portable speakers

6.3.3.1.8 tdc_extension

The `tdc_extension` element is used as an extension of the `target_device_category` element. See table 67.

6.3.3.1.9 b_ducking_depth_present

This Boolean indicates whether a `max_ducking_depth` element is present.

6.3.3.1.10 max_ducking_depth

The `max_ducking_depth` element indicates the maximum ducking depth according to table 68.

Table 68: max_ducking_depth

max_ducking_depth	Maximum ducking depth [dB]
0…62	-1 × max_ducking_depth
63	-∞

6.3.3.1.11 b_loud_corr_target

This Boolean indicates whether a `loud_corr_target` element is present.

6.3.3.1.12 loud_corr_target

The `loud_corr_target` element indicates a loudness correction factor for a presentation target. For values of `loud_corr_target` in [0; 30], this element determines a *target_corr_gain* value as:

$$\text{target\_corr\_gain} = (15 - \text{loud\_corr\_target})/2\,[\text{dB}_2]$$

If the value of `loud_corr_target` is 31, this element indicates a *loud_corr_gain* value of 0 dB_2.

6.3.3.1.13 n_substreams_in_presentation

This helper variable indicates the number of audio substreams (including dialogue enhancement substreams) in a presentation. The order of the substreams is defined by the order of appearance in `ac4_substream_group_info()` (inner loop) and appearance of `ac4_sgi_specifier()` in `ac4_presentation_v1_info()` (outer loop).

6.3.3.1.14 b_active

This Boolean indicates whether the substream *sus* is active for target *t*.

6.3.3.1.15 alt_data_set_index

The `alt_data_set_index` element, together with a possible extension via `variable_bits()`, indicates which of the alternative object audio metadata sets in the loop over `n_alt_data_sets` in `oamd_dyndata_single()` is used for the objects in the substream. A value of 0 indicates that no alternative data set is used and a value greater than 0 indicates that the alternative data set with index *s* = `alt_data_set_index` - 1 is used.

6.3.3.1.16 b_additional_data

This Boolean indicates whether additional data is present.

6.3.3.1.17 add_data_bytes_minus1

The `add_data_bytes_minus1` element indicates the number of additional data bytes minus 1. To get the number of additional data bytes, *add_data_bytes*, a value of 1 needs to be added to `add_data_bytes_minus1`. The result of additional `variable_bits()` is added to *add_data_bytes* if *add_data_bytes* = 16.

6.3.3.1.17a immersive_audio_indicator

This Boolean indicates whether this presentation contains immersive audio. This indicator has no effect on decoding and is purely informative.

6.3.3.1.17b b_oamd_common_timing

This Boolean indicates whether all OAMD substreams over all substream groups of this presentation are based on a common OAMD timing. This allows for simplified merging of substreams.

6.3.3.1.17c b_advanced_de_data_present

This Boolean indicates whether advanced dialogue enhancement metadata is present in an `advanced_de_data` element.

6.3.3.1.17d advanced_de_compr_tc_attack

This unsigned integer indicates the attack time constant of the compressor. The attack time, in milliseconds, is determined by the following equation:

$$\text{attack time constant} = \text{advanced\_de\_compr\_tc\_attack} * 5\text{ms}$$

The attack time ranges from 0 to 315 ms, with a 5 ms quantization step.

6.3.3.1.17e advanced_de_compr_tc_release

This unsigned integer indicates the release time constant of the compressor. The release time, in milliseconds, is determined by the following equation:

$$\text{release time constant} = \text{advanced\_de\_compr\_tc\_release} * 20\text{ms}$$

The release time ranges from 0 to 1 260 ms, with a 20 ms quantization step.

6.3.3.1.17f advanced_de_compr_ratio

This unsigned integer indicates the compressor ratio. The ratio is determined by the following equation:

$$\text{compressor ratio} = (\text{advanced\_de\_compr\_ratio}+1)/16$$

6.3.3.1.17g advanced_de_compr_thresh

This integer indicates a compressor threshold relatively to the dialogue level. The absolute threshold *T*, in dB, is determined by the following equation:

$$\text{T} = \text{D} + \text{advanced\_de\_compr\_thresh}$$

The dialog level *D* is determined by the following equation:

$$\text{D} = \text{dialnorm} - \text{rtll\_comp}$$

`advanced_de_compr_thresh` ranges from -32 to 31.

6.3.3.1.17h advanced_de_compr_gain

This unsigned integer indicates the compressor gain value, in dB. The gain ranges from 0 dB to 31 dB.

6.3.3.1.18 add_data

The `add_data` element holds additional data and is reserved for future use.

6.3.3.1.19 drc_metadata_size_value

This value indicates the DRC metadata size, i.e. the size of the `drc_frame()` element, in bits.

NOTE: If `b_more_bits` is true, the DRC metadata size is increased by `variable_bits`

6.3.3.1.20 b_more_bits

This Boolean indicates whether additional `variable_bits` are used to determine the DRC metadata size.

6.3.3.1.21 drc_frame

The `drc_frame` element is present as specified in ETSI TS 103 190-1 [1], clause 4.2.14.5. The possible values for *nr_drc_channels* specified in ETSI TS 103 190-1 [1], clause 4.3.13.7.1 shall be extended by the values shown in table 69 for the immersive and 22.2 channel configurations.

Table 69: nr_drc_channels for higher channel modes

Channel configuration	*nr_drc_channels*	Group 1	Group 2	Group 3	Group 4
7.X.4 (3/4/4)	4	L, R, [LFE]	C	Ls, Rs, Lb, Rb	Tfl, Tfr, Tbl, Tbr
9.X.4 (5/4/4)	4	L, R, [LFE], Lscr, Rscr	C	Ls, Rs, Lb, Rb	Tfl, Tfr, Tbl, Tbr
22.2	4	L, R, [LFE, LFE2], Lw, Rw	C	Ls, Rs, Lb, Rb, Bfl, Bfr, Bfc, Cb	Tfl, Tfr, Tbl, Tbr, Tsl, Tsr, Tfc, Tbc, Tc

The square brackets in table 69 indicate that the LFE channel(s) are part of the group, in case they are present in the used channel configuration.

6.3.3.1.22 b_substream_group_gains_present

This Boolean indicates whether gain values for the substream groups are present.

6.3.3.1.23 b_keep

This Boolean indicates whether substream group gains used in the previous AC-4 frame should be used. Otherwise, new substream group gains are present in the bitstream. Until the first reception of an AC-4 frame where `b_keep` is false, a value of 0 dB shall be used as substream group gain.

6.3.3.1.24 sg_gain

The `sg_gain` element indicates the substream group gain for the substream group *sg* according to table 70. The substream group gain is applied to all substreams in the substream group *sg*.

Table 70: sg_gain

sg_gain	Substream group gain [dB]
0…62	-0,25 × sg_gain
63	-∞

6.3.3.1.25 b_associated

This flag indicates whether associated audio mixing metadata is present.

6.3.3.1.26 b_associate_is_mono

This Boolean indicates whether the associated audio substream is a mono stream.

6.3.3.1.27 pres_ch_mode

This helper variable indicates the (virtual) channel mode of the presentation. *pres_ch_mode* can be derived from `ac4_toc()` information as indicated by the pseudocode in pseudocode 25.

Pseudocode 25: Determining pres_ch_mode

```
pres_ch_mode = -1;
b_obj_or_ajoc = 0;
```

```
for (sg = 0; sg < n_substream_groups; sg++) {
  for (s = 0; s < n_substreams[sg]; s++) {
    if ("substream s is of type ac4_substream()") {
      if (b_channel_coded) {
        pres_ch_mode = superset(pres_ch_mode, ch_mode);
      }
      else {
        b_obj_or_ajoc = 1;
      }
    }
  }
}
if (b_obj_or_ajoc) {
  pres_ch_mode = -1;
}
```

The *superset()* function takes two *ch_mode* values and returns one *ch_mode* value. The returned *ch_mode* value indicates the lowest possible *ch_mode* which includes all channels present in the two provided *ch_mode* values. If one input *ch_mode* value is -1, the other input *ch_mode* value is returned. The result of *superset(0,1)* shall be 1.

EXAMPLE: If the two input *ch_mode* values are 4 and 11, indicating the channel modes 5.1 and 7.0.4 (see table 56), *superset()* will return a value of 12, indicating the channel mode 7.1.4.

6.3.3.1.28 pres_ch_mode_core

This helper variable indicates the (virtual) core channel mode of the presentation.

The core channel mode of a substream is derived from substream properties as indicated in table 71.

Table 71: ch_mode_core

Substream properties	Core channel mode	*ch_mode_core*
In ac4_substream_group_info: b_channel_coded is false and b_ajoc is true, in ac4_substream_info_ajoc: b_static_dmx is true and b_lfe is false	5.0	3
In ac4_substream_group_info: b_channel_coded is false and b_ajoc is true, in ac4_substream_info_ajoc: b_static_dmx is true and b_lfe is true	5.1	4
In ac4_substream_group_info: b_channel_coded is true, in ac4_substream_info_chan: ch_mode in [11,13]	5.0.2 core	5
In ac4_substream_group_info: b_channel_coded is true, in ac4_substream_info_chan: ch_mode in [12,14]	5.1.2 core	6
All other cases	N/A	-1

To get the core channel mode of a presentation, the substream core channel modes from all substreams belonging to the presentation are evaluated. The pseudocode in pseudocode 26 describes the determination of *pres_ch_mode_core*, which shall be done after the determination of *pres_ch_mode*.

Pseudocode 26: Determining pres_ch_mode_core

```
pres_ch_mode_core = -1;
b_obj_or_ajoc_adaptive = 0;
for (sg = 0; sg < n_substream_groups; sg++) {
  for (s = 0; s < n_substreams[sg]; s++) {
    if ("substream s is of type ac4_substream()") {
      if (b_channel_coded) {
        pres_ch_mode_core = superset(pres_ch_mode_core, ch_mode_core);
      }
      else {
        if (b_ajoc) {
          if (b_static_dmx) {
            pres_ch_mode_core = superset(pres_ch_mode_core, ch_mode_core);
          }
          else {
            b_obj_or_ajoc_adaptive = 1;
          }
        }
        else {
          b_obj_or_ajoc_adaptive = 1;
        }
      }
    }
  }
}
if (b_obj_or_ajoc_adaptive) {
  pres_ch_mode_core = -1;
}
if (pres_ch_mode_core == pres_ch_mode) {
  pres_ch_mode_core = -1;
}
```

The *superset()* function takes two *ch_mode_core* values and returns one *ch_mode_core* value. The returned *ch_mode_core* value indicates the lowest possible *ch_mode_core* which includes all channels present in the two provided *ch_mode_core* values. If one input *ch_mode_core* value is -1, the other input *ch_mode_core* value is returned. The result of *superset(0,1)* shall be 1.

6.3.3.1.29 b_pres_4_back_channels_present

This helper flag is a logical disjunction of all `b_4_back_channels_present` flags of all substreams in the presentation that carry that flag.

6.3.3.1.29a b_pres_centre_present

This helper flag is a logical disjunction of all `b_centre_present` flags of all substreams in the presentation that carry that flag.

6.3.3.1.30 pres_top_channel_pairs

This helper variable shall be initialized according to table 72.

Table 72: pres_top_channel_pairs

pres_top_channel_pairs	Condition
0	`top_channels_present` is 0 for all substreams in the presentation that carry that flag
1	At least one substream in the presentation that carries `top_channels_present` has this one set to 1 or 2
2	At least one substream in the presentation that carries `top_channels_present` has this one set to 3

6.3.3.1.31 b_pres_has_lfe

This Boolean indicates whether an LFE is present in the presentation. In this case, at least one substream in the presentation contains an LFE. Otherwise, no substream contains an LFE.

If `pres_ch_mode` ≥ 0, the presence of an LFE is determined by the function `channel_mode_contains_Lfe()`. If `pres_ch_mode` = -1, the presence of an LFE is determined by the condition that `pres_ch_mode_core` = 4 or `pres_ch_mode_core` = 6.

6.3.3.2 oamd_substream

6.3.3.2.1 Introduction

The `oamd_substream` element is referenced in the substream group. The same `oamd_substream` element can be referenced by several substream groups.

6.3.3.2.2 b_oamd_common_data_present

This Boolean indicates whether `oamd_common_data` is present in the corresponding `oamd_substream`.

6.3.3.2.3 b_oamd_timing_present

This Boolean indicates whether `oamd_timing_data` is present in the corresponding `oamd_substream`.

6.3.4 Audio data

6.3.4.1 b_some_signals_inactive

This Boolean indicates whether some of the signals present in `var_channel_element` contain coded silence as indicated by `dmx_active_signals_mask`.

6.3.4.2 dmx_active_signals_mask[]

The array of flags `dmx_active_signals_mask` indicates which of the signals in `var_channel_element` contain coded silence. Each flag corresponds to a signal in the `var_channel_element`, according to the order in which the signals are transmitted. If the flag is set to false, the signal contains coded silence.

6.3.4.3 b_dmx_timing

This Boolean indicates whether an individual OAMD timing data instance is present for core decoding mode.

6.3.4.4 b_oamd_extension_present

This Boolean indicates whether OAMD extension data is present.

NOTE: OAMD extension data is reserved for future use.

6.3.4.5 skip_data

The `skip_data` element holds additional data and is reserved for future use.

6.3.4.6 b_umx_timing

This Boolean indicates whether an individual OAMD timing data instance is present for full decoding mode.

6.3.4.7 b_derive_timing_from_dmx

This Boolean indicates whether the OAMD timing data instance that is present for core decoding mode is also valid for full decoding mode.

6.3.5 Channel elements

6.3.5.1 immersive_codec_mode_code

For parsing `immersive_codec_mode_code`, one bit shall be read first. If this bit is 0, another two bits shall be read. The bit code indicates the *immersive_codec_mode* for the `immersive_channel_element` as shown in table 73.

Table 73: Immersive codec mode

immersive_codec_mode_code	immersive_codec_mode		Description
	value	descriptor	
0b000	0	SCPL	SMP + S-CPL
0b001	1	ASPX_SCPL	SMP + A-SPX + S-CPL
0b010	2	ASPX_ACPL_1	SMP + A-SPX + A-CPL mode 1
0b011	3	ASPX_ACPL_2	SMP + A-SPX + A-CPL mode 2
0b1	4	ASPX_AJCC	SMP + A-SPX + A-JCC

6.3.5.2 core_channel_config

This helper variable indicates the core channel configuration that is used in the bitstream syntax of the `immersive_channel_element` and can be derived from the *immersive_codec_mode* as shown in table 74.

Table 74: Core channel configuration

immersive_codec_mode	core_channel_config	Core channel configuration
SCPL	7CH_STATIC	5.X.2
ASPX_SCPL	7CH_STATIC	5.X.2
ASPX_ACPL_1	7CH_STATIC	5.X.2
ASPX_ACPL_2	7CH_STATIC	5.X.2
ASPX_AJCC	5CH_DYNAMIC	5.X.0

6.3.5.3 core_5ch_grouping

The `core_5ch_grouping` indicates the core channel data element grouping as shown in table 75.

Table 75: Core channel data element grouping

core_5ch_grouping	Core channel data element grouping
0	1+2+2
1	3+2
2	1+4
3	5

6.3.5.4 22_2_codec_mode

This bit indicates the 22.2 codec mode as shown in table 76.

Table 76: 22.2 codec mode

22_2_codec_mode	Meaning
0	Simple
1	A-SPX

6.3.5.5 var_codec_mode

This bit indicates the codec mode of the `var_channel_element()` as shown in table 77.

Table 77: var codec mode

var_codec_mode	Meaning
0	Simple
1	A-SPX

6.3.5.6 var_coding_config

This bit indicates the coding configuration of the `var_channel_element()`.

6.3.6 Advanced Joint Object Coding (A-JOC)

6.3.6.1 ajoc

6.3.6.1.1 ajoc_num_decorr

This element indicates the number of decorrelators to be used for the A-JOC reconstruction.

6.3.6.2 ajoc_config

6.3.6.2.1 ajoc_decorr_enable[d]

This flag indicates whether the decorrelator with index *d* is enabled.

6.3.6.2.2 ajoc_object_present[o]

This flag indicates whether the reconstructed object with index *o* is present.

6.3.6.2.3 ajoc_num_bands_code[o]

This element indicates the number of parameter bands, *ajoc_num_bands*, for the reconstructed object with index *o* according to table 78.

Table 78: ajoc_num_bands_code

ajoc_num_bands_code	ajoc_num_bands
0	23
1	15
2	12
3	9
4	7
5	5
6	3
7	1

6.3.6.2.4 ajoc_quant_select

This element indicates the quantization mode for the reconstructed object with index *o* according to table 79.

Table 79: ajoc_quant_select

ajoc_quant_select	Meaning
0	Fine
1	Coarse

6.3.6.2.5 ajoc_sparse_select

This element indicates the reconstruction mode for the reconstructed object with index *o* according to table 80.

Table 80: ajoc_sparse_select

ajoc_sparse_select	Meaning
0	Upmix from all inputs; all upmix matrices are transmitted
1	Upmix from some inputs; not all upmix matrices are transmitted

6.3.6.2.6 ajoc_mix_mtx_dry_present[o][ch]

This Boolean indicates whether the dry matrix element is present. In this case, `mix_mtx_dry` is transmitted in the bitstream. Otherwise, the matrix element shall be set to a default value of 0,0.

NOTE: If the element is not transmitted, then input *ch* is not mixed into the reconstruction of output *o*.

6.3.6.2.7 ajoc_mix_mtx_wet_present[o][d]

This Boolean indicates whether the wet matrix element is present. In this case, `mix_mtx_wet` is transmitted in the bitstream. Otherwise, the matrix element shall be set to a default value of 0,0.

NOTE: If the element is not transmitted, then decorrelator output *de* is not mixed into the reconstruction of output *o*.

6.3.6.3 ajoc_data

6.3.6.3.1 ajoc_b_nodt

This element indicates whether time-differential coding is allowed in the current A-JOC frame. Time-differential coding is only allowed if `ajoc_b_nodt` = 0.

6.3.6.4 ajoc_data_point_info

6.3.6.4.1 ajoc_num_dpoints

This element indicates the number of A-JOC data points.

6.3.6.4.2 ajoc_start_pos

This element indicates the first QMF time slot that is included in the interpolation of a reconstruction matrix element.

6.3.6.4.3 ajoc_ramp_len_minus1

This element indicates the length of the interpolation ramp *ajoc_ramp_len* = `ajoc_ramp_len_minus1` + 1 in QMF time slots.

6.3.6.5 ajoc_huff_data

6.3.6.5.1 diff_type

This value indicates the type of differential coding according to table 81.

Table 81: diff_type

diff_type	Description
0	Frequency differential coding
1	Time differential coding

6.3.6.5.2 ajoc_hcw

The `ajoc_hcw` element is a Huffman coded code word for the A-JOC data. The pseudocode in pseudocode 27 shows how to determine the correct Huffman table for decoding of the Huffman code words.

Pseudocode 27: get_ajoc_hcb()

```
get_ajoc_hcb(data_type, quant_mode, hcb_type)
{
  // data_type = {DRY, WET}
  // quant_mode = {COARSE, FINE}
  // hcb_type = {F0, DF, DT}

  ajoc_hcb = AJOC_HCB_<data_type>_<quant_mode>_<hcb_type>;
  // the line above expands using the inputs data_type, quant_mode and hcb_type

  return ajoc_hcb;
}
```

NOTE: These 12 Huffman code books are given in clause A.1.1 and are named according to the schema outlined above.

6.3.6.6 ajoc_dmx_de_data

6.3.6.6.1 b_dmx_de_cfg

This Boolean indicates whether a dialogue enhancement configuration is present. `b_keep_dmx_de_coeffs` shall be ignored by the decoder if `b_dmx_de_cfg` is true.

6.3.6.6.2 b_keep_dmx_de_coeffs

This Boolean indicates whether the dialogue downmix coefficients are independent. In this case, the coefficients used in the previous AC-4 frame are to be reused in the current frame. Otherwise, new coefficients are present in the bitstream.

`b_keep_dmx_de_coeffs` shall be ignored by the decoder if the current codec frame is an I-frame or `b_dmx_de_cfg` is true.

6.3.6.6.3 de_main_dlg_flag

The `de_main_dlg_flag[]` element is an array of flags indicating which objects represent the main dialogue that should be boosted when dialogue enhancement is enabled. A flag set to 1 indicates an object where the dialogue enhancement is enabled. The function *dlg_obj()*, as specified in pseudocode 28, shows how to derive the helper elements *num_dlg_obj* and *dlg_idx[]* from the `de_main_dlg_flag[]` element. The variable *num_dlg_obj* indicates the number of main dialogue objects, and the array *dlg_idx[]* gives a mapping from dialogue object indices to upmix object indices.

Pseudocode 28: dlg_obj()

```
dlg_obj()
{
  num_dlg_obj = 0;
```

```
  for (obj = 0; obj < num_umx_signals; obj++)
  {
    if (de_main_dlg_flag[obj])
    {
      dlg_idx[num_dlg_obj] = obj;
      num_dlg_obj++;
    }
  }
  return (num_dlg_obj, dlg_idx);
}
```

6.3.6.6.4 de_dlg_dmx_coeff_idx

The `de_dlg_dmx_coeff_idx` elements contain quantized downmix coefficients for the main dialogue objects. The conversion into *de_dlg_dmx_coeff* values shall be done according to table 82.

Table 82: de_dlg_dmx_coeff

de_dlg_dmx_coeff_idx	*de_dlg_dmx_coeff*
0b0	0
0b10000	1/15
0b10001	2/15
0b10010	3/15
0b10011	4/15
0b10100	5/15
0b10101	6/15
0b10110	7/15
0b10111	8/15
0b11000	9/15
0b11001	10/15
0b11010	11/15
0b11011	12/15
0b11100	13/15
0b11101	14/15
0b1111	1

6.3.6.7 ajoc_bed_info

6.3.6.7.1 b_obj_without_bed_info_present

The `b_obj_without_bed_info_present` flag indicates whether objects without corresponding bed render information are present in the AJOC substream.

6.3.6.7.2 num_obj_with_bed_render_info

The `num_obj_with_bed_render_info` unsigned integer determines the number of objects with corresponding bed render information in the AJOC substream.

NOTE: The bed objects are the first transmitted objects.

6.3.7 Advanced Joint Channel Coding (A-JCC)

6.3.7.1 ajcc_data

6.3.7.1.1 b_no_dt

This Boolean indicates whether the present parameters are delta-frequency or delta-time coded. In this case, all parameters are delta-frequency coded. Otherwise, present parameters are delta-frequency or delta-time coded.

6.3.7.1.2 ajcc_num_param_bands_id

The ajcc_num_param_bands_id element is an index to table 83 that indicates the number of parameter bands.

Table 83: ajcc_num_bands_table

ajcc_num_param_bands_id	ajcc_num_bands_table[ajcc_num_param_bands_id]
0	15
1	12
2	9
3	7

6.3.7.1.3 ajcc_core_mode

The ajcc_core_mode element indicates the channel configuration of the advanced joint channel coded core, as shown in table 84.

Table 84: ajcc_core_mode

ajcc_core_mode	Present channels
0	L, R, C, Ls, Rs
1	L, R, C, Tfl, Tfr

NOTE: ajcc_core_mode is only present if *b_5fronts* is false.

6.3.7.1.4 ajcc_qm_f

The ajcc_qm_f element indicates the quantization mode for the parameters that are related to the front channels, as shown in table 85.

Table 85: ajcc_qm_f

ajcc_qm_f	Description
0	Fine quantization
1	Coarse quantization

6.3.7.1.5 ajcc_qm_b

The ajcc_qm_b indicates the quantization mode for the parameters that are related to the back channels, as shown in table 86.

Table 86: ajcc_qm_b

ajcc_qm_b	Description
0	Fine quantization
1	Coarse quantization

6.3.7.1.6 ajcc_qm_ab

The ajcc_qm_ab indicates the quantization mode for the alpha and beta parameter, as shown in table 87.

Table 87: ajcc_qm_ab

ajcc_qm_ab	Description
0	Fine quantization
1	Coarse quantization

6.3.7.1.7 ajcc_qm_dw

The `ajcc_qm_dw` indicates the quantization mode for the dry and wet parameter, as shown in table 88.

Table 88: ajcc_qm_dw

ajcc_qm_dw	Description
0	Fine quantization
1	Coarse quantization

6.3.7.2 ajcc_framing_data

6.3.7.2.1 ajcc_interpolation_type

This value indicates the type of interpolation used as shown in table 89.

Table 89: ajcc_interpolation_type

ajcc_interpolation_type	Meaning
0	Smooth A-JCC interpolation
1	Steep A-JCC interpolation

6.3.7.2.2 ajcc_num_param_sets_code

The `ajcc_num_param_sets_code` element indicates the value of *ajcc_num_param_sets* as shown in table 90. The *ajcc_num_param_sets* variable indicates how many A-JCC parameter sets per frame are transmitted in the bitstream.

Table 90: ajcc_num_param_sets

ajcc_num_param_sets_code	*ajcc_num_param_sets*
0	1
1	2

6.3.7.2.3 ajcc_param_timeslot

When steep interpolation is used, this value indicates the QMF time slot (0 - 31) at which the A-JCC parameter set values change.

6.3.7.3 ajcc_huff_data

6.3.7.3.1 diff_type

This bitstream element has been described in clause 6.3.6.5.1.

6.3.7.3.2 ajcc_hcw

The `ajcc_hcw` element is a Huffman coded code word for the A-JCC data. The pseudocode in pseudocode 29 describes how to determine the correct Huffman code book for decoding the Huffman code words.

Pseudocode 29: get_ajcc_hcb()

```
get_ajcc_hcb(data_type, quant_mode, hcb_type)
{
  // data_type = {ALPHA, BETA, DRY, WET}
  // quant_mode = {COARSE, FINE}
  // hcb_type = {F0, DF, DT}
  if (data_type == ALPHA || data_type == BETA) {
    ajcc_hcb = ACPL_HCB_<data_type>_<quant_mode>_<hcb_type>; // (Note 1)
    // the line above expands using the inputs data_type, quant_mode and hcb_type
```

```
  }
  else {
    ajcc_hcb = AJCC_HCB_<data_type>_<quant_mode>_<hcb_type>; // (Note 2)
  }
  return ajcc_hcb;
}
```

NOTE 1: These 12 Huffman code books are given in ETSI TS 103 190-1 [1], Annex A.3 and are named according to the schema outlined above.

NOTE 2: These 12 Huffman code books are given in clause A.1.2.

6.3.8 Metadata

6.3.8.1 basic_metadata - basic metadata

6.3.8.1.1 b_substream_loudness_info

This Boolean indicates whether substream loudness information is present.

6.3.8.1.2 substream_loudness_bits

The `substream_loudness_bits` element specifies the substream loudness in dB_{FS}, in steps of 1/4 dB_{FS}:

$$\text{substream\_loudness} = -\text{substream\_loudness\_bits}/4[\text{dB}_{\text{FS}}]$$

6.3.8.1.3 b_further_substream_loudness_info

This Boolean indicates whether additional substream loudness information is present in the `further_loudness_info()` element.

6.3.8.1.4 dialnorm_bits

This codeword indicates the `dialnorm` value for the substream.

If `presentation_version` is 0, the `dialnorm` value of each dialogue substream shall be the `dialnorm` value for the mix of the music and effects substream and the dialogue substream. When mixing a music and effects, dialogue and associated audio substream, the `dialnorm` of either the dialogue or the associated audio substream may be used for the mixed signal.

6.3.8.1.5 loro_dmx_loud_corr

The `loro_dmx_loud_corr` element signals the loudness correction that shall be applied when downmixing the substream to Left only/Right only (Lo/Ro).

If `presentation_version` is 0, the `loro_dmx_loud_corr` value of each dialogue substream shall be the `loro_dmx_loud_corr` value for the mix of music and effects substream and the dialogue substream.

6.3.8.1.6 ltrt_dmx_loud_corr

If `presentation_version` is 0, the `ltrt_dmx_loud_corr` value of each dialogue substream shall be the `ltrt_dmx_loud_corr` value for the mix of music and effects substream and the dialogue substream.

6.3.8.2 further_loudness_info - additional loudness information

6.3.8.2.1 b_rtllcomp

This Boolean indicates whether Real-Time Loudness Leveller (RTLL) data is present.

6.3.8.2.2 rtll_comp

This field indicates the real-time loudness correction factor `rtll_comp_gain`, which can be calculated as:

$$\text{rtll\_comp\_gain} = (\text{rtll\_comp} - 128)/4 [\text{dB}]$$

6.3.8.3 dialog_enhancement - dialogue enhancement

6.3.8.3.1 b_de_simulcast

This Boolean indicates whether separate dialogue enhancement data for core decoding is present.

6.3.8.4 Channel mode query functions

6.3.8.4.1 channel_mode_contains_Lfe()

This function returns true if the channel mode contains an LFE channel.

Pseudocode 30: channel_mode_contains_lfe

```
channel_mode_contains_Lfe()
{
  if (ch_mode in [4,6,8,10,12,14,15]) {
    return TRUE;
  }
  return FALSE;
}
```

6.3.8.4.2 channel_mode_contains_c()

This function returns true if the channel mode contains a Centre channel.

Pseudocode 31: channel_mode_contains_c

```
channel_mode_contains_c()
{
  if (ch_mode == 0 || (ch_mode >= 2 && ch_mode <= 15)) {
    return TRUE;
  }
  return FALSE;
}
```

6.3.8.4.3 channel_mode_contains_lr()

This function returns true if the channel mode contains a Left and a Right channel.

Pseudocode 32: channel_mode_contains_lr

```
channel_mode_contains_lr()
{
  if (ch_mode >= 1 && ch_mode <= 15) {
    return TRUE;
  }
  return FALSE;
}
```

6.3.8.4.4 channel_mode_contains_LsRs()

This function returns true if the channel mode contains a Left Side/Surround and a Right Side/Surround channel.

Pseudocode 33: channel_mode_contains_LsRs

```
channel_mode_contains_LsRs()
{
```

```
  if (ch_mode >= 3 && ch_mode <= 15) {
    return TRUE;
  }
  return FALSE;
}
```

6.3.8.4.5 channel_mode_contains_LbRb()

This function returns true if the channel mode contains a Left Back and a Right Back channel.

Pseudocode 34: channel_mode_contains_LrsRrs

```
channel_mode_contains_LbRb()
{
  if (ch_mode == 5 || ch_mode == 6 || (ch_mode >= 11 && ch_mode <= 15)) {
    return TRUE;
  }
  return FALSE;
}
```

6.3.8.4.6 channel_mode_contains_LwRw()

This function returns true if the channel mode contains a Left Wide and a Right Wide channel.

Pseudocode 35: channel_mode_contains_LwRw

```
channel_mode_contains_LwRw()
{
  if (ch_mode == 7 || ch_mode == 8 || ch_mode == 15) {
    return TRUE;
  }
  return FALSE;
}
```

6.3.8.4.7 channel_mode_contains_TflTfr()

This function returns true if the channel mode contains a Top Front Left and a Top Front Right channel.

Pseudocode 36: channel_mode_contains_TflTfr

```
channel_mode_contains_TflTfr()
{
  if (ch_mode == 9 || ch_mode == 10) {
    return TRUE;
  }
  return FALSE;
}
```

6.3.8.5 pan_signal_selector

This field indicates the possibility to perform a simplified decoding of the `3_0_channel_element`.

A simplified decoding and processing of `3_0_channel_element` may be carried out if all of the following conditions are met:

- `ch_mode` is 2 (indicating 3 channels);
- `3_0_channel_element` is used;
- `3_0_codec_mode` is set to SIMPLE;
- `3_0_coding_config` is set to 1; and
- `pan_signal_selector` is greater than 0.

To carry out the simplified decoding, the following steps shall be executed:

1) partially decode the three_channel_data as indicated in table 91;

2) pan the two decoded signals using pan_dialog[0] and pan_dialog[1] as described in clause 5.10.2.3.

NOTE: The simplified processing obviates the full processing as described in ETSI TS 103 190-1 [1], clause 5.3.3.3

Table 91: pan_signal_selector

pan_signal_selector	Meaning
0	decode sf_data of all 3 tracks
1	decode sf_data of track 1 and track 2
2	decode sf_data of track 0 and track 2
3	decode sf_data of track 0 and track 1

6.3.9 Object audio metadata (OAMD)

6.3.9.1 Introduction

OAMD comprises parameters that indicate properties of audio objects.

6.3.9.2 oamd_common_data - OAMD common data

6.3.9.2.1 Introduction

The oamd_common_data contains properties that are common to all objects in the corresponding substream group.

6.3.9.2.2 b_default_screen_size_ratio

This Boolean indicates whether the decoder assumes a default value of 1,0 for *master_screen_size_ratio*.

6.3.9.2.3 master_screen_size_ratio_code

The master_screen_size_ratio_code bitstream element indicates the *master_screen_size_ratio*, which defines the ratio of the width of the screen to the distance between the L and R speakers in the mastering room. If master_screen_size_ratio = 1, the L and R speakers in the mastering room were located on the edge of the screen. The value of *master_screen_size_ratio* is given by:

$$\text{master\_screen\_size\_ratio} = (\text{master\_screen\_size\_ratio\_code}+1)/33$$

6.3.9.2.4 b_bed_object_chan_distribute

This Boolean indicates whether the channel distribution of bed objects is activated. In this case, one bed object can be distributed over multiple channels. Otherwise, bed objects cannot be distributed.

6.3.9.3 oamd_timing_data

6.3.9.3.1 Introduction

The oamd_timing_data element provides timing information to be utilized for synchronization of object properties to the object essence audio samples. One oamd_timing_data element applies to all substreams in a substream group.

6.3.9.3.2 sample_offset

The *sample_offset* helper variable indicates a timing offset value in audio samples.

6.3.9.3.3 oa_sample_offset_type

The oa_sample_offset_type element uses a prefix code and indicates the configuration type for the *sample_offset* as shown in table 92.

Table 92: oa_sample_offset_type

oa_sample_offset_type	Description
0b0	*sample_offset* = 0
0b10	*sample_offset* is indicated by oa_sample_offset_code
0b11	*sample_offset* is indicated by oa_sample_offset

6.3.9.3.4 oa_sample_offset_code

The oa_sample_offset_code element indicates the *sample_offset* value by a prefix codeword as shown in table 93.

Table 93: oa_sample_offset_code

oa_sample_offset_code	*sample_offset*
0b0	16
0b10	8
0b11	24

6.3.9.3.5 oa_sample_offset

The oa_sample_offset element indicates the *sample_offset* value.

6.3.9.3.6 num_obj_info_blocks

The num_obj_info_blocks element indicates the number of present object_info_block elements for each object. The value of num_obj_info_blocks can be 0 unless the current codec frame is an I-frame.

6.3.9.3.7 block_offset_factor

The block_offset_factor element is available for each object_info_block and indicates a timing offset in audio samples for each corresponding object_info_block.

6.3.9.3.8 ramp_duration

The ramp_duration bitstream element indicates the *ramp_duration* value, which is a helper variable indicating a transition time value in audio samples. The *ramp_duration* value has a range of [0; 2 047].

6.3.9.3.9 ramp_duration_code

The ramp_duration_code element indicates the *ramp_duration* value. This bitstream element codes the most common *ramp_duration* values and enables an option for alternative *ramp_duration* signalling. The possible values for ramp_duration_code are shown in table 94.

Table 94: ramp_duration_code

ramp_duration_code	Description
0b00	*ramp_duration* is 0 audio samples
0b01	*ramp_duration* is 512 audio samples
0b10	*ramp_duration* is 1 536 audio samples
0b11	*ramp_duration* is signalled via additional elements

6.3.9.3.10 b_use_ramp_table

This Boolean indicates how the *ramp_duration* in audio samples is indicated. In this case, the *ramp_duration* value is indicated by the `ramp_duration_table` element. Otherwise, the *ramp_duration* value is indicated by the by the `ramp_duration` element.

6.3.9.3.11 ramp_duration_table

The `ramp_duration_table` element indicates the *ramp_duration* value as shown in table 95.

Table 95: ramp_duration_table

`ramp_duration_table`	***ramp_duration* in audio 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

6.3.9.4 oamd_dyndata_single

6.3.9.4.1 Introduction

The `oamd_dyndata_single` element contains object properties for objects of a single substream. The order of the objects corresponds to the presence of object essences in the substream.

6.3.9.4.2 b_ducking_disabled

This Boolean indicates whether automatic ducking is disabled for the corresponding object.

6.3.9.4.3 object_sound_category

The `object_sound_category` element defines the object sound category according to table 96.

Table 96: object_sound_category

object_sound_category	**Sound category**
0	Not indicated
1	Dialogue
Other	Reserved

6.3.9.4.4 n_alt_data_sets

The `n_alt_data_sets` element indicates the number of alternative properties sets.

6.3.9.4.5 b_keep

This Boolean indicates whether the values of the last object property update are still valid.

6.3.9.4.6 b_common_data

This Boolean indicates whether the alternative properties set has common parameters for all present objects.

6.3.9.4.7 b_alt_gain

This Boolean indicates whether alternative gain data is present in the bitstream. In this case, the gain data contained in the `alt_gain` element shall be used. Otherwise, the gain data in the *object_gain* element of each block of the `object_basic_info` element shall be used.

6.3.9.4.8 alt_gain

The `alt_gain` element indicates the alternative gain value, as shown in table 97, to be used in place of *object_gain*.

Table 97: alt_gain

alt_gain	object_gain [dB]
0…62	14 – alt_gain
63	$-\infty$

6.3.9.4.9 b_alt_position

This Boolean indicates whether the set of alternative properties `alt_pos3D_X`, `alt_pos3D_Z`, `alt_pos3D_Z_sign`, and `alt_pos3D_Z` contains position data. If the alternative properties set contains position data, that position data shall be used in place of the position data contained in each block of the `object_render_info` element.

6.3.9.4.10 alt_pos3D_X

The `alt_pos3D_X` element indicates the X-axis position data in the alternative properties set. The value of the X-axis position is given by:

$$\text{object\_position\_X} = \text{clip3}\left(0.0, 1.0, \frac{\text{clip3}(0, 62, \text{alt\_pos3D\_X})}{62} + \frac{\text{clip3}(-2, 2, \text{ext\_prec\_pos3D\_X})}{62 \times 5}\right)$$

NOTE 1: Clause 6.3.9.8.4.1 describes room-anchored position data.

NOTE 2: The value of `ext_prec_pos3D_X` is specified in clause 6.3.9.12.2.

6.3.9.4.11 alt_pos3D_Y

The `alt_pos3D_Y` element indicates the Y-axis position data in the alternative properties set. The value of the Y-axis position is given by:

$$\text{object\_position\_Y} = \text{clip3}\left(0.0, 1.0, \frac{\text{clip3}(0, 62, \text{alt\_pos3D\_Y})}{62} + \frac{\text{clip3}(-2, 2, \text{ext\_prec\_pos3D\_Y})}{62 \times 5}\right)$$

NOTE 1: Clause 6.3.9.8.4.1 describes room-anchored position data.

NOTE 2: The value of `ext_prec_pos3D_Y` is specified in clause 6.3.9.12.3.

6.3.9.4.12 alt_pos3D_Z_sign

The `alt_pos3D_z_sign` element indicates the sign of the Z-axis position data in the alternative properties set. If the `alt_pos3D_z_sign` is set to 1, the *sign* of coordinate Z is +1; otherwise, the *sign* is -1.

NOTE: Clause 6.3.9.8.4.1 describes room-anchored position data.

6.3.9.4.13 alt_pos3D_Z

The `alt_pos3D_Z` element indicates the Z-axis position data in the alternative properties set. The value of the Z-axis position is given by:

$$\text{object\_position\_Z} = \text{clip3}\left(-1.0, 1.0, \text{sign} \times \frac{\text{clip3}(0, 15, \text{alt\_pos3D\_Z})}{15} + \frac{\text{clip3}(-2, 2, \text{ext\_prec\_pos3D\_Z})}{15 \times 5}\right)$$

NOTE 1: Clause 6.3.9.8.4.1 describes room-anchored position data.

NOTE 2: The value of *sign* is specified in clause 6.3.9.4.12.

NOTE 3: The value of `ext_prec_pos3D_Z` is specified in clause 6.3.9.12.4.

6.3.9.5 oamd_dyndata_multi

The `oamd_dyndata_multi` element contains object properties for the objects in the corresponding substream group. The order of the objects in `oamd_dyndata_multi` corresponds to the order of all object essences present over all audio substreams of the according substream group in the order of bitstream presence.

6.3.9.6 obj_info_block

6.3.9.6.1 Introduction

An `obj_info_block` element contains one update of properties for an object. Two tables of properties may be present in `obj_info_block`:

- basic information (present in `obj_basic_info`);
- rendering information (present in `obj_render_info`).

NOTE: Handling of the updates in terms of timing is indicated via `oamd_timing_data` as described in clause 6.3.9.3.

6.3.9.6.2 b_object_not_active

This Boolean indicates whether the object essence of the corresponding object contains silence.

6.3.9.6.3 object_basic_info_status

The *object_basic_info_status* helper variable indicates whether `object_basic_info` is present, whether default values shall be used, or whether the value from the previous `obj_info_block` shall be reused. The possible values for *object_basic_info_status* are shown in table 98.

Table 98: object_basic_info_status

object_basic_info_status	Description
DEFAULT	• *object_gain* = -∞ dB • *object_priority* = 0
ALL_NEW	Updates for all properties in `object_basic_info` are present
REUSE	Reuse metadata from previous `obj_info_block`

6.3.9.6.4 b_basic_info_reuse

This Boolean indicates whether properties shall be reused from the `obj_basic_info` of the previous `obj_info_block`.

6.3.9.6.5 object_render_info_status

The *object_render_info_status* helper variable indicates whether `object_render_info` is present, whether default values shall be used for the according properties, whether the values shall be determined from the previous `obj_info_block`, or whether only some fields shall be reused from the previous `obj_info_block`. The possible values for *object_render_info_status* are shown in table 99.

Table 99: object_render_info_status

object_render_info_status	Description
DEFAULT	• *object_position_X* = 0,5 • *object_position_Y* = 0,5 • *object_position_Z* = 0 • `zone_mask` = 0b000 • `b_enable_elevation` = false • *object_width* = 0 • *object_screen_factor* = 0 • `b_object_snap` = false
ALL_NEW	Updates for all properties in `object_render_info` are present
REUSE	Reuse metadata from previous `obj_info_block`
PART_REUSE	Partially reuse metadata from previous `obj_info_block`

6.3.9.6.6 b_render_info_reuse

This Boolean indicates whether object properties shall be reused. In this case, object properties from the `obj_render_info` of the previous `obj_info_block` shall be reused. Otherwise, object properties shall not be reused.

6.3.9.6.7 b_render_info_partial_reuse

This Boolean indicates whether some object properties shall be reused from the `obj_render_info` of the previous `obj_info_block`.

6.3.9.6.8 b_add_table_data

This Boolean indicates whether additional reserved data is present.

6.3.9.6.9 add_table_data_size_minus1

The `add_table_data_size_minus1` element indicates the size of `add_table_data` field minus 1.

6.3.9.6.10 add_table_data

The `add_table_data` is a field reserved for future extensions of `obj_info_block`.

6.3.9.7 obj_basic_info

6.3.9.7.1 Introduction

The `obj_basic_info` field contains high-level information about each object, such as its gain value.

6.3.9.7.2 b_default_basic_info_md

This Boolean indicates default object basic info metadata. In this case, the metadata shall be set to default values. Otherwise, the metadata are explicitly transmitted.

object_gain has a default value of 0 dB. *object_priority* has a default value of 1.

NOTE: These default values differ from the default values for *object_gain* and *object_priority* defined in table 98, where `object_basic_info_status` = DEFAULT.

6.3.9.7.3 basic_info_md

The `basic_info_md` element is coded using a variable length prefix code as defined in table 100, and indicates the basic info metadata coding.

Table 100: basic_info_md

basic_info_md	Description
0b0	Non-default gain, default priority
0b10	Non-default gain, non-default priority
0b11	Default gain, non-default priority

6.3.9.7.4 object_gain_code

The `object_gain_code` element is coded using a variable length prefix code as defined in table 101 and indicates the value for *object_gain*.

Table 101: object_gain_code

object_gain_code	***object_gain*** **[dB]**
0b0	Specified by `object_gain_value`
0b10	$-\infty$
0b11	Set to *object_gain* of previous object

6.3.9.7.5 object_gain_value

The `object_gain_value` indicates the value of *object_gain* as shown in table 102.

Table 102: object_gain_value

object_gain_value	***object_gain*** **[dB]**
0…14	15 – object_gain_value
15…63	14 – object_gain_value

6.3.9.7.6 object_priority_code

The `object_priority_code` element indicates the *object_priority* of an object which is in the range [0, 1]. The higher the *object_priority* value, the more important that object is. The value of *object_priority* is given by:

$$\text{object\_priority} = \text{object\_priority\_code}/31$$

6.3.9.8 obj_render_info

6.3.9.8.1 Introduction

The `obj_render_info` element contains multiple object properties, e.g. position properties.

6.3.9.8.2 obj_render_info_mask

These flags indicate whether property updates are transmitted. If `object_render_info_status` = `ALL_NEW`, the flags are implicitly set to true, and all properties are transmitted.

b_obj_render_position_present
This flag indicates that a position update is transmitted in this section (if true); otherwise, the position of the previous `obj_info_block` shall be reused.

b_obj_render_zone_present
This flag indicates that a zone mask update is transmitted in this section (if true); otherwise, the zone mask information of the previous `obj_info_block` shall be reused.

b_obj_render_otherprops_present
This flag indicates that updates on a several other properties are transmitted in this section (if true); otherwise, the properties of the previous `obj_info_block` shall be reused.

6.3.9.8.3 b_diff_pos_coding

This Boolean indicates whether the position properties are differential coded by `diff_pos3D_X`, `diff_pos3D_Y`, and `diff_pos3D_Z`. The difference is relative to the position transmitted in the previous `obj_info_block`. Values are coded as signed integers (2's complement).

6.3.9.8.4 Room-anchored position

6.3.9.8.4.1 Introduction

The object position is given by 3 dimensional cartesian coordinates and determined as *object_position_X*, *object_position_Y*, and *object_position_Z* and derived from:

- `pos3D_X`, `pos3D_Y`, `pos3D_Z_sign`, and `pos3D_Z`; or
- `diff_pos3D_X`, `diff_pos3D_Y`, and `diff_pos3D_Z` if the position is coded differentially; and
- `ext_prec_pos3D_X`, `ext_prec_pos3D_Y`, and `ext_prec_pos3D_Z` if the position coordinate is coded using extend precision.

The coordinates are defined in relation to a normalized room. The room consists of two adjacent normalized unit cubes to describe the playback room boundaries as shown in figure 13. The origin is defined to be the front left corner of the room at the height of the main screen. Location (0,5; 0; 0) corresponds to the middle of the screen.

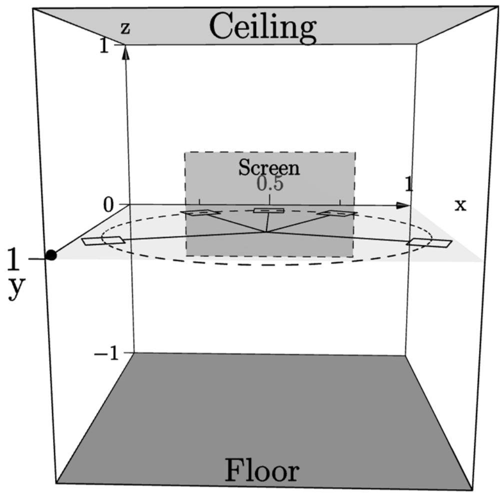


Figure 13: Object position coordinate system

- x-axis: describes latitude, or left/right position:
  - x = 0 corresponds to left wall;
  - x = 1 corresponds to right wall.
- y-axis: describes longitude, or front/back position:
  - y = 0 corresponds to front wall;
  - y = 1 corresponds to back wall.
- z-axis: describes elevation, or up/down position:
  - z = 0 corresponds to a horizontal plane at the height of the main screen, surround, and rear surround loudspeakers;
  - z = 1 corresponds to the ceiling;
  - z = -1 corresponds to the floor.

6.3.9.8.4.2 diff_pos3D_X

The `diff_pos3D_X` element indicates the X-axis coordinate of the object position *object_position_X*, coded as a difference between current and previous object position in standard precision. The current value is given by:

$$\text{object\_position\_X} = \text{clip3}\left(0.0, 1.0, \frac{\text{clip3}(0, 62, \text{pos3D\_X\_prev})}{62} + \frac{\text{clip3}(-4, 3, \text{diff\_pos3D\_X})}{62} + \frac{\text{clip3}(-2, 2, \text{ext\_prec\_pos3D\_X})}{62 \times 5}\right)$$

NOTE 1: The *pos3D_X_prev* value is the standard precision position value coded in the previous metadata update block.

NOTE 2: The value of `ext_prec_pos3D_X` is specified in clause 6.3.9.12.2.

6.3.9.8.4.3 diff_pos3D_Y

The `diff_pos3D_Y` element indicates the Y-axis coordinate of the object position *object_position_Y*, coded as a difference between current and previous object position *object_position_Y_prev*. The current value is given by:

$$\text{object\_position\_Y} = \text{clip3}\left(0.0, 1.0, \frac{\text{clip3}(0, 62, \text{pos3D\_Y\_prev})}{62} + \frac{\text{clip3}(-4, 3, \text{diff\_pos3D\_Y})}{62} + \frac{\text{clip3}(-2, 2, \text{ext\_prec\_pos3D\_Y})}{62 \times 5}\right)$$

NOTE 1: The *pos3D_Y_prev* value is the standard precision position value coded in the previous metadata update block.

NOTE 2: The value of `ext_prec_pos3D_Y` is specified in clause 6.3.9.12.3.

6.3.9.8.4.4 diff_pos3D_Z

The `diff_pos3D_Z` element indicates the Z-axis coordinate of the object position *object_position_Z*, coded as a difference between current and previous object position *object_position_Z_prev*. The current value is given by:

$$\text{object\_position\_Z} = \text{clip3}\left(-1.0, 1.0, \frac{\text{clip3}(-15, 15, \text{pos3D\_Z\_prev})}{15} + \frac{\text{clip3}(-4, 3, \text{diff\_pos3D\_Z})}{15} + \frac{\text{clip3}(-2, 2, \text{ext\_prec\_pos3D\_Z})}{15 \times 5}\right)$$

NOTE 1: The *pos3D_Z_prev* value is the standard precision position value coded in the previous metadata update block.

NOTE 2: The value of `ext_prec_pos3D_Z` is specified in clause 6.3.9.12.4.

6.3.9.8.4.5 pos3D_X

The `pos3D_X` element indicates the X-axis coordinate of the object position *object_position_X*. This field is coded as unsigned integer. The linear value of X-axis coordinate is given by:

$$\text{object\_position\_X} = \text{clip3}\left(0.0, 1.0, \frac{\text{clip3}(0, 62, \text{pos3D\_X})}{62} + \frac{\text{clip3}(-2, 2, \text{ext\_prec\_pos3D\_X})}{62 \times 5}\right)$$

NOTE: The value of `ext_prec_pos3D_X` is specified in clause 6.3.9.12.2.

6.3.9.8.4.6 pos3D_Y

The `pos3D_Y` element indicates the Y-axis coordinate of the object position *object_position_Y*. This field is coded as unsigned integer. The linear value of Y-axis coordinate is given by:

$$\text{object\_position\_Y} = \text{clip3}\left(0.0, 1.0, \frac{\text{clip3}(0, 62, \text{pos3D\_Y})}{62} + \frac{\text{clip3}(-2, 2, \text{ext\_prec\_pos3D\_Y})}{62 \times 5}\right)$$

NOTE: The value of `ext_prec_pos3D_Y` is specified in clause 6.3.9.12.3.

6.3.9.8.4.7 pos3D_Z_sign

The `pos3D_Z_sign` element indicates the sign of the Z-axis coordinate of the object position. If the `pos3D_Z_sign` bit is set to 1, the sign of *object_position_Z* is +1, otherwise the sign is -1.

6.3.9.8.4.8 pos3D_Z

The `pos3D_Z` element indicates the Z-axis coordinate of the object position *object_position_Z*. The value is coded as unsigned integer. The linear value of Z-axis coordinate is given by:

$$\text{object\_position\_Z} = \text{clip3}\left(-1.0, 1.0, \text{sign} \times \frac{\text{clip3}(0, 15, \text{pos3D\_Z})}{15} + \frac{\text{clip3}(-2, 2, \text{ext\_prec\_pos3D\_Z})}{15 \times 5}\right)$$

NOTE 1: The value of *sign* is specified in clause 6.3.9.8.4.7.

NOTE 2: The value of `ext_prec_pos3D_Z` is specified in clause 6.3.9.12.4.

6.3.9.8.5 b_grouped_zone_defaults

This Boolean indicates whether the properties of the zone group shall be set to default values. Otherwise, `group_zone_flag[]` and `zone_mask` are present in the bitstream.

6.3.9.8.6 group_zone_flag

The `group_zone_flag[]` is an array of flags used for assignment of properties of the zone group as shown in table 103.

Table 103: group_zone_flag

group_zone_flag[]	Description
`group_zone_flag[2]`	`zone_mask` is present
`group_zone_flag[1]`	`b_enable_elevation` is false
`group_zone_flag[0]`	`b_object_snap` is true

6.3.9.8.7 zone_mask

Zones are subsets of speakers as specified in clause A.4. The `zone_mask` is a code that indicates constraints to include or exclude zones for the rendering process. The assignment of `zone_mask` values to zone constraints is shown in table 104.

Table 104: zone_mask

zone_mask	Zone constraints
0	No constraints
1	Back zone disabled
2	Side zone disabled
3	Only centre and back zone enabled
4	Only screen zone enabled
5	Only surround zone enabled
6	Only proscenium zone enabled
7	Reserved

6.3.9.8.8 b_enable_elevation

This Boolean indicates whether rendering to height (top) speakers is enabled. By default `b_enable_elevation` should be true. It is implicitly signalled via `group_zone_flag[1]`.

6.3.9.8.9 b_object_snap

This Boolean indicates whether the artistic intent is to avoid changing the object timbre by distributing its energy over multiple loudspeakers. By default `b_object_snap` should be false. It is implicitly signalled via `group_zone_flag[0]`.

6.3.9.8.10 b_grouped_other_defaults

This Boolean indicates whether properties of the other properties group shall be set to default values.

6.3.9.8.11 group_other_mask

The `group_other_mask` is a bitmask that indicates which properties of the other properties group are present in the bitstream as shown in table 105.

Table 105: group_other_mask

group_other_mask	Description
0b0001	Object width properties present
0b0010	`object_screen_factor` and `object_depth_factor` element present
0b0100	Object distance properties present
0b1000	Object divergence properties present

6.3.9.8.12 object_width_mode

The `object_width_mode` flag indicates the mode of the object spreading according to table 106. Increasing the object width may increase the number of speakers used to playback a particular object.

Table 106: object_width_mode

object_width_mode	Description
0	Only one value is transmitted for 3D object spreading, i.e. the spreading is identical for all three dimensions.
1	Each dimension of the 3D object spreading is transmitted individually.

6.3.9.8.13 object_width_code

The `object_width_code` element indicates the radial *object_width*. The value is given by:

$$\text{object\_width} = \text{object\_width\_code}/31$$

6.3.9.8.14 object_width_X_code

The `object_width_X_code` element indicates *object_width_X*, the X-axis value of the 3D object width. The value is given by:

$$\text{object\_width\_X} = \text{object\_width\_X\_code}/31$$

6.3.9.8.15 object_width_Y_code

The `object_width_Y_code` element indicates *object_width_Y*, the Y-axis value of the 3D object width. The value is given by:

$$\text{object\_width\_Y} = \text{object\_width\_Y\_code}/31$$

6.3.9.8.16 object_width_Z_code

The `object_width_Z_code` element indicates *object_width_Z*, the Z-axis value of the 3D object width. The value is given by:

$$\text{object\_width\_Z} = \text{object\_width\_Z\_code}/31$$

6.3.9.8.17 object_screen_factor_code

The object_screen_factor_code element indicates the scaling factor *object_screen_factor*, which is applied to the X and Z dimensions for objects that pan across the screen. The value is given by:

$$\text{object\_screen\_factor} = (\text{object\_screen\_factor\_code}+1)/8$$

If the object_screen_factor_code element is not present, object_screen_factor shall be 0.

6.3.9.8.18 object_depth_factor

The amount of X- and Z-position scaling varies linearly with the Y-position so that if sounds are panned off-screen they can use the full width and height of the room. The object_depth_factor indicates the rate at which the scaling converges when the object approaches the screen. The value specifies the exponent to be applied to the Y-position value as specified in table 107.

Table 107: object_depth_factor

object_depth_factor	Y-position exponent
0	0,25
1	0,5
2	1
3	2

6.3.9.8.19 b_obj_at_infinity

This Boolean indicates whether the object_distance_factor is infinity.

6.3.9.8.20 object_distance_factor_code

The object_distance_factor_code bitstream element indicates the *object_distance_factor*, which defines how far outside of the room the corresponding object is. The values of *object_distance_factor* are specified in table 108.

Table 108: object_distance_factor

object_distance_factor_code	*object_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

6.3.9.8.21 object_div_mode

The object_div_mode element indicates how the value of *object_divergence* is transmitted, as shown in table 109.

Table 109: object_div_mode

object_div_mode	Description
0b00	indicated by object_div_table.
0b01	reuse *object_divergence* as transmitted in the previous obj_info_block.
0b10	indicated by object_div_code.
0b11	Reserved

6.3.9.8.22 object_div_table

The object_div_table element indicates the value of *object_divergence* as shown in table 110.

Table 110: object_div_table

object_div_table	*object_divergence*
0b00	0,500755
0b01	0,608529
0b10	0,704833
0b11	1,0

NOTE: An emulation of the divergence control found in a Live Broadcast mixing console can be achieved when an object is positioned at x = 0,5, y = 0, z = 0 (centre front speaker position). With the object positioned at this location, the object divergence signals the separation in the x axis of the diverged object from the Centre speaker (div = 0) to the Left and Right speakers (div = 1) and the effective maximum azimuth angle is ±45 degrees.

6.3.9.8.23 object_div_code

The object_div_code indicates the value of *object_divergence* as shown in table 111.

Table 111: object_div_code

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

NOTE: An emulation of the divergence control found in a Live Broadcast mixing console can be achieved when an object is positioned at x = 0,5, y = 0, z = 0 (centre front speaker position). With the object positioned at this location, the object divergence signals the separation in the x axis of the diverged object from the Centre speaker (div = 0) to the Left and Right speakers (div = 1) and the effective maximum azimuth angle is ±45 degrees.

6.3.9.9 bed_render_info

6.3.9.9.1 Introduction

The bed rendering information in the bitstream contains elements that indicate parameters for rendering of bed objects. Metadata for rendering bed objects to output loudspeaker layouts align with metadata for custom downmix of channel-based immersive content. Bitstream elements that are not described in this clause are described in clause 6.3.10.3, because the semantics are identical.

NOTE: Normative operation of a renderer is not in scope of this specification.

6.3.9.9.2 b_bed_render_info

The `b_bed_render_info` flag indicates whether information for rendering of bed objects is present in the bitstream.

6.3.9.9.3 b_stereo_dmx_coeff

The `b_stereo_dmx_coeff` flag indicates whether information for rendering of bed objects to stereo output is present in the bitstream.

6.3.9.9.4 b_cdmx_data_present

The `b_cdmx_data_present` flag indicates whether information for rendering of immersive bed objects is present in the bitstream.

6.3.9.9.5 Bed render information gain presence flags

The following flags indicate whether corresponding custom gain values are present in the bitstream:

- b_cdmx_w_to_f
- b_cdmx_b4_to_b2
- b_cdmx_t2_to_f_s_b
- b_cdmx_t2_to_f_s
- b_cdmx_tb_to_f_s_b
- b_cdmx_tb_to_f_s
- b_cdmx_tf_to_f_s_b
- b_cdmx_tf_to_f_s
- b_cdmx_tfb_to_tm

NOTE: See clause 6.2.8.8 for syntactical information on the listed presence flags.

6.3.9.9.6 Bed render information channel presence flags

Table 112 shows flags that indicate whether corresponding channels are present as bed objects.

Table 112: Bed render information channel presence flags

Bed render information channel presence flag	Channels present as bed objects
`b_tf_ch_present`	Top Front Left and/or Top Front Right
`b_tb_ch_present`	Top Back Left and/or Top Back Right
`b_tm_ch_present`	Top left and/or top right
NOTE: Column one lists presence flags and column two the channels that are present as bed objects, if the corresponding flag is true.	

6.3.9.9.7 gain_w_to_f_code

The `gain_w_to_f_code` codeword indicates the gain value *gain_w_to_f* as specified in table 112a.

Table 112a: gain_w_to_f_code

`gain_w_to_f_code`	***gain_w_to_f* in dB**
0	0
1	-1,5
2	-3
3	-4,5
4	-6
5	-9
6	-12
7	-inf

If `gain_w_to_f_code` is not present in the bitstream, *gain_w_to_f* shall be -3 dB.

The gain value *gain_w_to_f* may be used by a renderer to mix the Left Wide / Right Wide to Left / Right channels.

6.3.9.9.8 gain_b4_to_b2_code

The `gain_b4_to_b2_code` codeword indicates the gain value *gain_b4_to_b2* as specified in table 112b.

Table 112b: gain_b4_to_b2_code

`gain_b4_to_b2_code`	***gain_b4_to_b2* in dB**
0	0
1	-1,5
2	-3
3	-4,5
4	-6
5	-9
6	-12
7	-inf

If `gain_b4_to_b2` is not present in the bitstream, *gain_b4_to_b2* shall be -3 dB.

The gain value *gain_b4_to_b2* may be used by a renderer to mix the Left Back / Right Back to Left Side/Surround / Right Side/Surround channels.

NOTE: See Note 2 in table A.27.

6.3.9.9.9 gain_tfb_to_tm_code

The `gain_tfb_to_tm_code` codeword indicates the gain value *gain_tfb_to_tm* as specified in table 112c.

Table 112c: gain_tfb_to_tm_code

`gain_tfb_to_tm_code`	***gain_tfb_to_tm* in dB**
0	0
1	-1,5
2	-3
3	-4,5
4	-6
5	-9
6	-12
7	-inf

If `gain_tfb_to_tm_code` is not present in the bitstream, *gain_tfb_to_tm* shall be -3 dB.

The gain value *gain_tfb_to_tm* may be used by a renderer to mix the Top Front Left / Top Front Right and Top Back Left / Top Back Right channels to Top Side Left / Top Side Right channels respectively.

6.3.9.10 Trim

6.3.9.10.1 b_trim_present

The `b_trim_present` flag indicates whether trim metadata is present in the bitstream.

6.3.9.10.2 warp_mode

The `warp_mode` element is a codeword that indicates an object position adjustment before rendering as specified in table 113.

Table 113: 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.

6.3.9.10.3 global_trim_mode

The `global_trim_mode` element is a codeword that indicates the trim mode as specified in table 114. The global trim mode is applicable to all trim configurations.

Table 114: 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: • `bal3D_Y_sign_tb_code` • `bal3D_Y_amount_tb` • `bal3D_Y_sign_lis_code` • `bal3D_Y_amount_lis`
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 6.3.9.10.5 and `b_disable_trim` is described in clause 6.3.9.10.6		

6.3.9.10.4 NUM_TRIM_CONFIGS

The *NUM_TRIM_CONFIGS* is a helper variable that indicates the number of trim configurations. The number of trim configurations is nine, i.e. *NUM_TRIM_CONFIGS* = 9.

6.3.9.10.5 b_default_trim

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

6.3.9.10.6 b_disable_trim

The `b_disable_trim` flag indicates whether the *trim_mode* is set to *disable_trim* for the corresponding trim configuration.

6.3.9.10.7 trim_balance_presence[]

The Boolean elements of this array indicate whether corresponding bitstream elements for trim and/or balance processing are transmitted.

Table 115 lists the correspondence between elements of `trim_balance_presence[]` and bitstream elements of the `oamd_common_data/trim` element.

Table 115: trim_balance_presence[] elements

`trim_balance_presence` element index	Transmitted bitstream elements
4	`trim_centre`
3	`trim_surround`
2	`trim_height`
1	`bal3D_Y_sign_tb_code`, `bal3D_Y_amount_tb`
0	`bal3D_Y_sign_lis_code`, `bal3D_Y_amount_lis`

6.3.9.10.8 trim_centre

The `trim_centre` codeword indicates the *trim_centre* value as specified in table 116.

Table 116: 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	-15
15	-36

6.3.9.10.9 trim_surround

The `trim_surround` codeword indicates the *trim_surround* value as specified in table 117.

Table 117: 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	-15
15	-36

6.3.9.10.10 trim_height

The `trim_height` codeword indicates the *trim_height* value as specified in table 118.

Table 118: 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	-15
15	-36

6.3.9.10.11 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 119.

Table 119: 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.

6.3.9.10.12 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 6.3.9.10.11.

6.3.9.10.13 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 120.

Table 120: 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.

6.3.9.10.14 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 6.3.9.10.13.

6.3.9.10a headphone

6.3.9.10a.1 b_headphone

This Boolean indicates whether common headphone metadata is present.

6.3.9.10a.2 hp_operation_mode

The `hp_operation_mode` codeword indicates the expected headphone operation mode as specified in table 120a.

Table 120a: hp_operation_mode

hp_operation_mode	Headphone operation mode
0	Stereo Mode
1	Default headphone NEAR mode
2	Default headphone FAR mode
3	Manual mode
4…7	reserved
NOTE: Headphone operation modes are further specified in table 120b.	

Table 120b: Headphone operation modes

Headphone operation mode	Description
Stereo mode	Headphone virtualization is disabled with bypass settings applied. Bypass settings are: • Render mode is *Bypass* for all objects; and • Head tracking is disabled for all objects.
Default headphone NEAR mode	Headphone virtualization is applied with default settings. Default settings are: • Render mode is *NEAR* for all objects; and • Head tracking application for all objects is indicated by `b_head_track_disable_all`.
Default headphone FAR mode	Headphone virtualization is applied with default settings. Default settings are: • Render mode is *FAR* for all objects; and • Head tracking application for all objects is indicated by `b_head_track_disable_all`.
Manual mode	Headphone virtualization is applied with control settings per object. The control settings are indicated by `hp_render_mode_obj` and `b_head_track_disable_obj`.
NOTE: Headphone rendering modes are described in clause 6.3.9.11.4.	

6.3.9.10a.3 b_head_track_disable_all

This Boolean indicates whether head tracking is disabled for all objects.

6.3.9.11 add_per_obj_md

6.3.9.11.1 b_obj_trim_disable

The `b_obj_trim_disable` flag indicates if the trim mode is *trim_disable* for the corresponding object.

6.3.9.11.2 b_ext_prec_pos

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

6.3.9.11.3 b_headphone

This Boolean indicates whether per-object headphone metadata is present.

6.3.9.11.4 hp_render_mode_obj

The `hp_render_mode_obj` element indicates the mode of rendering for individual objects, as shown in table 121.

Table 121: hp_render_mode_obj

hp_render_mode_obj	Description
0	Bypass mode (stereo): Headphone virtualization is bypassed and normal stereo rendering is applied. Directional cues and room simulation are disabled.
1	Near mode (neutral timbre): Headphone virtualization is applied with emphasis on directionality and neutral timbre. Little to no room simulation is applied.
2	Far mode (externalized): Headphone virtualization is applied with directionality and room simulation to match different acoustical environments. Emphasis on accurate localization.
3	Mid mode: headphone virtualization with intermediate amount of room simulation between "near" and "far".

6.3.9.11.5 b_head_track_disable_obj

This Boolean indicates whether head tracking is disabled for an individual object.

6.3.9.12 ext_prec_pos

6.3.9.12.1 ext_prec_pos_presence[]

The Boolean elements of this array indicate whether corresponding extended precision metadata elements are transmitted. If any extended precision metadata element is not transmitted, its value shall be 0.

Table 122 lists the correspondence between elements of `ext_prec_pos_presence[]` and bitstream elements of the `add_per_object_md/ext_prec_pos` element.

Table 122: 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`

6.3.9.12.2 ext_prec_pos3D_X

The `ext_prec_pos3D_X` indicates the extended precision value to be used for calculation of *object_position_X* as specified in clause 6.3.9.8.4.5 and clause 6.3.9.4.10. The extended precision value is shown in table 123.

Table 123: ext_prec_pos3D_X

`ext_prec_pos3D_X`	Semantics
0b00	1
0b01	2
0b10	-1
0b11	-2

6.3.9.12.3 ext_prec_pos3D_Y

The `ext_prec_pos3D_Y` indicates the extended precision value to be used for calculation of *object_position_Y* as specified in clause 6.3.9.8.4.6 and clause 6.3.9.4.11. The extended precision value is shown in table 124.

Table 124: ext_prec_pos3D_Y

`ext_prec_pos3D_Y`	Semantics
0b00	1
0b01	2
0b10	-1
0b11	-2

6.3.9.12.4 ext_prec_pos3D_Z

The `ext_prec_pos3D_Z` indicates the extended precision value to be used for calculation of *object_position_Z* as specified in clause 6.3.9.8.4.8 and clause 6.3.9.4.13. The extended precision value is shown in table 125.

Table 125: ext_prec_pos3D_Z

`ext_prec_pos3D_Z`	Semantics
0b00	1
0b01	2
0b10	-1
0b11	-2

6.3.10 Presentation data

6.3.10.1 loud_corr - loudness correction

6.3.10.1.1 b_obj_loud_corr

This Boolean indicates whether loudness correction data for objects is present.

6.3.10.1.2 b_corr_for_immersive_out

This Boolean indicates whether loudness correction data for immersive output channel configurations is present.

6.3.10.1.3 b_loro_loud_comp

This Boolean indicates whether Lo/Ro downmix loudness correction data is present.

6.3.10.1.4 b_ltrt_loud_comp

This Boolean indicates whether Lt/Rt (Dolby Pro Logic II compatible output) downmix loudness correction data is present.

6.3.10.1.5 b_loud_comp

This Boolean indicates whether a loudness correction value follows in the bitstream.

6.3.10.1.6 loud_corr_OUT_CH_CONF

This field indicates a loudness correction factor for a specific output channel configuration OUT_CH_CONF.
OUT_CH_CONF $\in$ {5_X, 5_X_2, 5_X_4, 7_X, 7_X_2, 7_X_4, 9_X_4, core_loro, core_ltrt, core_5_X, core_5_X_2}.

$$\text{loud\_corr\_gain\_OUT\_CH\_CONF} = (15 - \text{loud\_corr\_OUT\_CH\_CONF})/2\,[\text{dB}_2]$$

A value of 31 is reserved. If present, it indicates a gain of 0 dB.

6.3.10.2 custom_dmx_data - custom downmix data

6.3.10.2.1 bs_ch_config

The helper variable *bs_ch_config*, calculated as specified within clause 6.2.9.2, indicates the bitstream channel configuration relevant for custom downmix processing according to table 126.

Table 126: bs_ch_config

bs_ch_config	Description
0	9.X.4
1	7.X.4
2	5.X.4
3	9.X.2
4	7.X.2
5	5.X.2

6.3.10.2.2 b_cdmx_data_present

This flag indicates whether custom downmix data is present.

6.3.10.2.3 n_cdmx_configs_minus1

The `n_cdmx_configs_minus1` element indicates the number of custom downmix configurations present in the bitstream minus 1. To get the number of custom downmix configurations, a value of 1 needs to be added to `n_cdmx_configs_minus1`.

6.3.10.2.4 out_ch_config[dc]

The `out_ch_config` element indicates the output channel configuration for the downmix configuration *dc* according to table 127.

Table 127: out_ch_config

out_ch_config	Description
0	5.X.0
1	5.X.2
2	5.X.4
3	7.X.0
4	7.X.2
5, 6, 7	Unused

6.3.10.2.5 b_stereo_dmx_coeff

This Boolean indicates whether stereo downmix coefficients are present.

6.3.10.3 Custom downmix parameters

6.3.10.3.1 gain_f1_code

The `gain_f1_code` element indicates a *gain_f1* value according to table 128.

Table 128: gain_f1_code

gain_f1_code	gain_f1 [dB]
0	3,0
1	1,5
2	0
3	-1,5
4	-3,0
5	-4,5
6	-6,0
7	-∞

6.3.10.3.2 gain_f2_code, gain_b_code, gain_t1_code, and gain_t2[abcdef]_code

The following gain codes map to gain values as shown in table 129:

- `gain_f2_code` (*gain_f2*)
- `gain_b_code` (*gain_b*)
- `gain_t1_code` (*gain_t1*)
- `gain_t2a_code` (*gain_t2a*)
- `gain_t2b_code` (*gain_t2b*)
- `gain_t2c_code` (*gain_t2c*)
- `gain_t2d_code` (*gain_t2d*)
- `gain_t2e_code` (*gain_t2e*)
- `gain_t2f_code` (*gain_t2f*)

Table 129: gain_f2_code, gain_b_code, gain_t*_code

gain_f2_code, gain_b_code, gain_t*_code	gain_f2, gain_b, gain_t* [dB]
0	0
1	-1,5
2	-3,0
3	-4,5
4	-6,0
5	-9,0
6	-12,0
7	-∞

6.3.10.3.3 b_put_screen_to_c

This flag indicates whether the Lscr and Rscr signals are mixed into the Centre channel C.

6.3.10.3.4 b_top_front_to_front

This Boolean indicates whether the Tfl and Tfr signals are mixed into the front channels L and R.

6.3.10.3.5 b_top_front_to_side

This Boolean indicates whether the Tfl and Tfr signals are mixed into the side channels Ls and Rs.

6.3.10.3.6 b_top_back_to_front

This Boolean indicates whether the Tbl and Tbr signals are mixed into the front channels L and R.

6.3.10.3.7 b_top_back_to_side

This Boolean indicates whether the Tbl and Tbr signals are mixed into the side channels Ls and Rs.

6.3.10.3.8 b_top_to_front

This Boolean indicates whether the Tsl and Tsr signals are mixed into the front channels L and R.

6.3.10.3.9 b_top_to_side

This Boolean indicates whether Tsl and Tsr signals are mixed into the side channels Ls and Rs.

6.3.10.3.10 Default custom downmix parameter

If a custom downmix parameter is not transmitted for a certain *out_ch_config*, the default value for this custom downmix parameter shall be used as specified in table 130 with the following exception: For the downmix from `bs_ch_config` = 1 to `out_ch_config` = 4 the same `tool_t4_to_t2()` parameters as for the downmix to `out_ch_config` = 1 shall be used, if transmitted.

Table 130: Default custom downmix parameter

Custom downmix parameter	Default value
b_put_screen_to_c	False
b_top_front_to_front	False
b_top_front_to_side	True
b_top_back_to_front	False
b_top_back_to_side	True
b_top_to_front	False
b_top_to_side	True
gain_f1	-∞ dB
gain_f2	0 dB
gain_b	-3 dB
gain_t1	-3 dB
gain_t2a	-∞ dB
gain_t2b	-3 dB
gain_t2c	-∞ dB
gain_t2d	-∞ dB
gain_t2e	-3 dB
gain_t2f	-∞ dB

Annex A (normative): Tables

A.1 Huffman tables

A.1.1 A-JOC Huffman codebook tables

Table A.1: A-JOC Huffman codebook AJOC_HCB_DRY_COARSE_F0

Codebook name	AJOC_HCB_DRY_COARSE_F0
Codebook length table	AJOC_HCB_DRY_COARSE_F0_LEN
Codebook codeword table	AJOC_HCB_DRY_COARSE_F0_CW
codebook_length	51
cb_off	0

Table A.2: A-JOC Huffman codebook AJOC_HCB_DRY_FINE_F0

Codebook name	AJOC_HCB_DRY_FINE_F0
Codebook length table	AJOC_HCB_DRY_FINE_F0_LEN
Codebook codeword table	AJOC_HCB_DRY_FINE_F0_CW
codebook_length	101
cb_off	0

Table A.3: A-JOC Huffman codebook AJOC_HCB_DRY_COARSE_DF

Codebook name	AJOC_HCB_DRY_COARSE_DF
Codebook length table	AJOC_HCB_DRY_COARSE_DF_LEN
Codebook codeword table	AJOC_HCB_DRY_COARSE_DF_CW
codebook_length	51
cb_off	0

Table A.4: A-JOC Huffman codebook AJOC_HCB_DRY_FINE_DF

Codebook name	AJOC_HCB_DRY_FINE_DF
Codebook length table	AJOC_HCB_DRY_FINE_DF_LEN
Codebook codeword table	AJOC_HCB_DRY_FINE_DF_CW
codebook_length	101
cb_off	0

Table A.5: A-JOC Huffman codebook AJOC_HCB_DRY_COARSE_DT

Codebook name	AJOC_HCB_DRY_COARSE_DT
Codebook length table	AJOC_HCB_DRY_COARSE_DT_LEN
Codebook codeword table	AJOC_HCB_DRY_COARSE_DT_CW
codebook_length	101
cb_off	50

Table A.6: A-JOC Huffman codebook AJOC_HCB_DRY_FINE_DT

Codebook name	AJOC_HCB_DRY_FINE_DT
Codebook length table	AJOC_HCB_DRY_FINE_DT_LEN
Codebook codeword table	AJOC_HCB_DRY_FINE_DT_CW
codebook_length	201
cb_off	100

Table A.7: A-JOC Huffman codebook AJOC_HCB_WET_COARSE_F0

Codebook name	AJOC_HCB_WET_COARSE_F0
Codebook length table	AJOC_HCB_WET_COARSE_F0_LEN
Codebook codeword table	AJOC_HCB_WET_COARSE_F0_CW
codebook_length	21
cb_off	0

Table A.8: A-JOC Huffman codebook AJOC_HCB_WET_FINE_F0

Codebook name	AJOC_HCB_WET_FINE_F0
Codebook length table	AJOC_HCB_WET_FINE_F0_LEN
Codebook codeword table	AJOC_HCB_WET_FINE_F0_CW
codebook_length	41
cb_off	0

Table A.9: A-JOC Huffman codebook AJOC_HCB_WET_COARSE_DF

Codebook name	AJOC_HCB_WET_COARSE_DF
Codebook length table	AJOC_HCB_WET_COARSE_DF_LEN
Codebook codeword table	AJOC_HCB_WET_COARSE_DF_CW
codebook_length	21
cb_off	0

Table A.10: A-JOC Huffman codebook AJOC_HCB_WET_FINE_DF

Codebook name	AJOC_HCB_WET_FINE_DF
Codebook length table	AJOC_HCB_WET_FINE_DF_LEN
Codebook codeword table	AJOC_HCB_WET_FINE_DF_CW
codebook_length	41
cb_off	0

Table A.11: A-JOC Huffman codebook AJOC_HCB_WET_COARSE_DT

Codebook name	AJOC_HCB_WET_COARSE_DT
Codebook length table	AJOC_HCB_WET_COARSE_DT_LEN
Codebook codeword table	AJOC_HCB_WET_COARSE_DT_CW
codebook_length	41
cb_off	20

Table A.12: A-JOC Huffman codebook AJOC_HCB_WET_FINE_DT

Codebook name	AJOC_HCB_WET_FINE_DT
Codebook length table	AJOC_HCB_WET_FINE_DT_LEN
Codebook codeword table	AJOC_HCB_WET_FINE_DT_CW
codebook_length	81
cb_off	40

A.1.2 A-JCC Huffman codebook tables

Table A.13: A-JCC Huffman codebook AJCC_HCB_DRY_COARSE_F0

Codebook name	AJCC_HCB_DRY_COARSE_F0
Codebook length table	AJCC_HCB_DRY_COARSE_F0_LEN
Codebook codeword table	AJCC_HCB_DRY_COARSE_F0_CW
codebook_length	12
cb_off	0

Table A.14: A-JCC Huffman codebook AJCC_HCB_DRY_FINE_F0

Codebook name	AJCC_HCB_DRY_FINE_F0
Codebook length table	AJCC_HCB_DRY_FINE_F0_LEN
Codebook codeword table	AJCC_HCB_DRY_FINE_F0_CW
codebook_length	23
cb_off	0

Table A.15: A-JCC Huffman codebook AJCC_HCB_DRY_COARSE_DF

Codebook name	AJCC_HCB_DRY_COARSE_DF
Codebook length table	AJCC_HCB_DRY_COARSE_DF_LEN
Codebook codeword table	AJCC_HCB_DRY_COARSE_DF_CW
codebook_length	23
cb_off	11

Table A.16: A-JCC Huffman codebook AJCC_HCB_DRY_FINE_DF

Codebook name	AJCC_HCB_DRY_FINE_DF
Codebook length table	AJCC_HCB_DRY_FINE_DF_LEN
Codebook codeword table	AJCC_HCB_DRY_FINE_DF_CW
codebook_length	45
cb_off	22

Table A.17: A-JCC Huffman codebook AJCC_HCB_DRY_COARSE_DT

Codebook name	AJCC_HCB_DRY_COARSE_DT
Codebook length table	AJCC_HCB_DRY_COARSE_DT_LEN
Codebook codeword table	AJCC_HCB_DRY_COARSE_DT_CW
codebook_length	23
cb_off	11

Table A.18: A-JCC Huffman codebook AJCC_HCB_DRY_FINE_DT

Codebook name	AJCC_HCB_DRY_FINE_DT
Codebook length table	AJCC_HCB_DRY_FINE_DT_LEN
Codebook codeword table	AJCC_HCB_DRY_FINE_DT_CW
codebook_length	45
cb_off	22

Table A.19: A-JCC Huffman codebook AJCC_HCB_WET_COARSE_F0

Codebook name	AJCC_HCB_WET_COARSE_F0
Codebook length table	AJCC_HCB_WET_COARSE_F0_LEN
Codebook codeword table	AJCC_HCB_WET_COARSE_F0_CW
codebook_length	21
cb_off	0

Table A.20: A-JCC Huffman codebook AJCC_HCB_WET_FINE_F0

Codebook name	AJCC_HCB_WET_FINE_F0
Codebook length table	AJCC_HCB_WET_FINE_F0_LEN
Codebook codeword table	AJCC_HCB_WET_FINE_F0_CW
codebook_length	41
cb_off	0

Table A.21: A-JCC Huffman codebook AJCC_HCB_WET_COARSE_DF

Codebook name	AJCC_HCB_WET_COARSE_DF
Codebook length table	AJCC_HCB_WET_COARSE_DF_LEN
Codebook codeword table	AJCC_HCB_WET_COARSE_DF_CW
codebook_length	41
cb_off	20

Table A.22: A-JCC Huffman codebook AJCC_HCB_WET_FINE_DF

Codebook name	AJCC_HCB_WET_FINE_DF
Codebook length table	AJCC_HCB_WET_FINE_DF_LEN
Codebook codeword table	AJCC_HCB_WET_FINE_DF_CW
codebook_length	81
cb_off	40

Table A.23: A-JCC Huffman codebook AJCC_HCB_WET_COARSE_DT

Codebook name	AJCC_HCB_WET_COARSE_DT
Codebook length table	AJCC_HCB_WET_COARSE_DT_LEN
Codebook codeword table	AJCC_HCB_WET_COARSE_DT_CW
codebook_length	41
cb_off	20

Table A.24: A-JCC Huffman codebook AJCC_HCB_WET_FINE_DT

Codebook name	AJCC_HCB_WET_FINE_DT
Codebook length table	AJCC_HCB_WET_FINE_DT_LEN
Codebook codeword table	AJCC_HCB_WET_FINE_DT_CW
codebook_length	81
cb_off	40

A.2 Coefficient tables

A.2.1 ISF coefficients

This clause specifies matrix coefficients to render ISF object audio essences into speaker feeds.

Table A.25 and table A.26 list the coefficient matrix identifiers from the ts_103190_tables_part2.c file. The file is located in the ts_10319002v010101p0.zip archive, which accompanies the present document.

Table A.25: ISF coefficient matrix identifiers in ts_103190_tables_part2.c (Table 1 of 2)

M.U.L.Z (in)	Channel mode (out)				
	2.x	5.x	5.x.2	5.x.4	7.x
3.1.0.0	SR3100_to_2	SR3100_to_50	SR3100_to_502	SR3100_to_504	SR3100_to_70
5.3.0.0	SR5300_to_2	SR5300_to_50	SR5300_to_502	SR5300_to_504	SR5300_to_70
7.3.0.0	SR7300_to_2	SR7300_to_50	SR7300_to_502	SR7300_to_504	SR7300_to_70
9.5.0.0	SR9500_to_2	SR9500_to_50	SR9500_to_502	SR9500_to_504	SR9500_to_70
7.5.3.0	SR7530_to_2	SR7530_to_50	SR7530_to_502	SR7530_to_504	SR7530_to_70
15.9.5.1	SR15951_to_2	SR15951_to_50	SR15951_to_502	SR15951_to_504	SR15951_to_70

Table A.26: ISF coefficient matrix identifiers in ts_103190_tables_part2.c (Table 2 of 2)

M.U.L.Z (in)	Channel mode (out)				
	7.x.2	7.x.4	9.x	9.x.2	9.x.4
3.1.0.0	SR3100_to_702	SR3100_to_704	SR3100_to_90	SR3100_to_902	SR3100_to_904
5.3.0.0	SR5300_to_702	SR5300_to_704	SR5300_to_90	SR5300_to_902	SR5300_to_904
7.3.0.0	SR7300_to_702	SR7300_to_704	SR7300_to_90	SR7300_to_902	SR7300_to_904
9.5.0.0	SR9500_to_702	SR9500_to_704	SR9500_to_90	SR9500_to_902	SR9500_to_904
7.5.3.0	SR7530_to_702	SR7530_to_704	SR7530_to_90	SR7530_to_902	SR7530_to_904
15.9.5.1	SR15951_to_702	SR15951_to_704	SR15951_to_90	SR15951_to_902	SR15951_to_904

A.3 Channel configurations

Table A.27 shows a list of audio loudspeaker names with corresponding indices and a mapping to AC-4 audio channel layouts and channel group indices. The fields of this matrix indicate audio channels included in the corresponding channel layout as defined in see note 1 in table A.27 that are designated to the respective loudspeaker.

NOTE: See clause 3.4 for the format of configuration notation.

Table A.27: Speaker layouts and speaker indices

Speaker index	Speaker name	Channel layouts														Channel group index
		5.X	7.X			5.X.0	5.X.2	5.X.4	7.X.0	7.X.2	7.X.4	9.X.0	9.X.2	9.X.4	22.2	
			3/4/0	5/2/0	3/2/2											
0	L	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0
1	R	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
2	C	Y	Y	Y	Y	S	S	S	S	S	S	S	S	S	Y	1
3	Ls	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	2 (note 2)
4	Rs	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
5	Lb	-	Y	-	-	-	-	-	Y_B	Y_B	Y_B	S	S	S	Y	3
6	Rb	-	Y	-	-	-	-	-	Y_B	Y_B	Y_B	S	S	S	Y	
7	Tfl	-	-	-	Y	-	-	Y_{T3}	-	-	Y_{T3}	-	-	Y_{T3}	Y	4
8	Tfr	-	-	-	Y	-	-	Y_{T3}	-	-	Y_{T3}	-	-	Y_{T3}	Y	
9	Tbl	-	-	-	-	-	-	Y_{T3}	-	-	Y_{T3}	-	-	Y_{T3}	Y	5
10	Tbr	-	-	-	-	-	-	Y_{T3}	-	-	Y_{T3}	-	-	Y_{T3}	Y	
11	LFE	X	X	X	X	X	X	X	X	X	X	X	X	X	Y	6
12	Tsl	-	-	-	-	-	$Y_{T1,2}$	-	-	$Y_{T1,2}$	-	-	$Y_{T1,2}$	-	Y	7
13	Tsr	-	-	-	-	-	$Y_{T1,2}$	-	-	$Y_{T1,2}$	-	-	$Y_{T1,2}$	-	Y	
14	reserved	-	-	-	-	-	-	-	-	-	-	-	-	-	-	8 (note 3)
15	reserved	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
16	Tfc	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	9
17	Tbc	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	10
18	Tc	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	11
19	LFE2	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	12
20	Bfl	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	13
21	Bfr	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	
22	Bfc	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	14
23	Cb	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	15
24	Lscr	-	-	-	-	-	-	-	-	-	-	Y	Y	Y	-	16
25	Rscr	-	-	-	-	-	-	-	-	-	-	Y	Y	Y	-	
26	Lw	-	-	Y	-	-	-	-	-	-	-	-	-	-	Y	17
27	Rw	-	-	Y	-	-	-	-	-	-	-	-	-	-	Y	

NOTE 1: Y = Speaker present in layout; Y_B = Speaker present in layout (`b_4_back_channels_present` = true); $Y_{T1,2}$ = Speaker present in layout (`top_channels_present` = {1,2}); Y_{T3} = Speaker present in layout (`top_channels_present` = 3); S = Speaker present in layout but can be signalled as silent; X = Speaker present in layout only if the channel configuration includes LFE.

NOTE 2: If channel group 3 is silent or not present, the azimuth angle of group 2 is 110 degrees, otherwise it is 90 degrees.

NOTE 3: This group is deprecated. If speaker indices of group 8 are signalled, they should be interpreted as the corresponding indices of group 7.

Table A.28 maps ch_modes with speaker layouts from table A.27. Additionally, this table provides references to ITU channel names as defined in [i.1] the channel layouts are based on, including the respective speaker configuration notation in the form of (*upper+middle+bottom*) speakers.

Table A.28: Speaker layouts, Channel Modes and ITU configurations for Sound System

Channel Layout	ch_mode		ITU configuration for Sound System, [i.1]
	w/o LFE	with LFE	
2.0	1	-	A (0+2+0)
5.X	3	4	B (0+5+0)
7.X (3/4/0)	5	6	I (0+7+0)
7.X (5/2/0)	7	8	-
7.X (3/2/2)	9	10	C (2+5+0)
5.X.0	11	12	F (3+7+0)
5.X.2			
5.X.4			
7.X.0			
7.X.2			
7.X.4			
9.X.0	13	14	G (4+9+0)
9.X.2			
9.X.4			
22.2	-	15	H (9+10+3)

A.4 Speaker zones

Table A.29: Speaker zones

Speaker		ITU configuration for Sound System					
ITU SP Label	α (azimuth)	A (0+2+0)	B-E (*x*+5+0)	F,J (*x*+7+0)	G (4+9+0)	H (9+10+3)	I (0+7+0)
M+000	0°		Sc, P	Sc, P	Sc, P	Sc, P	Sc, P
M+SC, M-SC	See note 1				Sc, P		
M+030, M-030	±30°	Sc, Su, P	Sc, P	Sc, P	Sc, P	Sc, P	Sc, P
M+060, M-060	±60°					P	
M+090, M-090	±90°			Si, Su	Si, Su	Si, Su	Si, Su
M+110, M-110	±110°		B, Su				
M+135, M-135	±135°			B, Su	B, Su	B, Su	B, Su
M+180	180°					B, Su	
U+045, U-045	±45°			P	P	P	
NOTE 1: "SC" is located at the left and right edges of the display. NOTE 2: B = back zone, Si = side zone, Su = surround zone, Sc = screen zone, P = proscenium zone. The centre-back zone is a union of the back zone with the Centre speaker.							

Table A.29 specifies an assignment for the horizontal and height speakers listed in [i.1] to zones (height speakers belong to the height zone and do not influence assignment of speakers in the plane). For speaker pairs, the azimuth angle is positive for the left speaker, and negative for the right speaker.

From the listed configuration, further configurations can be derived. ITU configurations for Sound System *C* through *E* are equivalent to the one for Sound System *B* since only height speakers are added, *F* and *J* are equivalent because they only differ in their height speakers. Layouts without the Centre speaker can be derived by substituting the Centre speaker by Left and Right.

The stereo configuration effectively defeats zone constraints; the back and side zones are empty, while all other zones contain only the two speakers.

A.5 Speaker Locations

Table A.30 lists approximate speaker positions for each channel, based on table A.27 and table A.29. Polar vector angles are with regards to [i.1], while the cartesian coordinates can be used with systems such as CTA 3D Audio [i.8] Speaker Location Descriptors. The azimuth angles in conjunction with the cartesian Z coordinates determine cylindrical coordinate vectors.

Value definitions:

- Azimuth: The azimuth angle expressed in degrees; positive values rotate to the left when facing the front.
- X: Coordinate value normalized to the maximum distance along the X axis from the centre of the room. The X axis is positive to the right of the centre.
- Y: Coordinate value normalized to the maximum distance along the Y axis from the centre of the room. The Y axis is positive from the centre going forward.
- Z: Coordinate value normalized to the maximum distance along the Z axis from the centre of the room. The Z axis is positive from the centre going up.

Table A.30: Speaker Locations (informative)

Speaker		Polar	Cartesian			Note
Name	**Label**	**Azimuth**	**X**	**Y**	**Z**	
Left	L	+45°	-1	+1	0	
Right	R	-45°	+1	+1	0	
Centre	C	0°	0	+1	0	
Left Side/Surround	Ls	+90°	-1	0	0	7+.X.X layouts
		+110°	-1	-0,333	0	5.X.X layouts
Right Side/Surround	Rs	-90°	+1	0	0	7+.X.X layouts
		-110°	+1	-0,333	0	5.X.X layouts
Left Back	Lb	+135°	-1	-1	0	
Right Back	Rb	-135°	+1	-1	0	
Top Front Left	Tfl	+45°	-0,5	+0,5	+1	
Top Front Right	Tfr	-45°	+0,5	+0,5	+1	
Top Back Left	Tbl	+135°	-0,5	-0,5	+1	
Top Back Right	Tbr	-135°	+0,5	-0,5	+1	
Low-Frequency Effects	LFE	+45°	-1	+1	-1	
Top Side Left	Tsl	+90°	-0,5	0	+1	
Top Side Right	Tsr	-90°	+0,5	0	+1	
Top Front Centre	Tfc	0°	0	+0,5	+1	
Top back centre	Tbc	+180°	0	-0,5	+1	
Top centre	Tc	0°	0	0	+1	
Low-Frequency Effects 2	LFE2	-45°	+1	+1	-1	
Bottom Front Left	Bfl	+45°	-1	+1	-1	
Bottom Front Right	Bfr	-45°	+1	+1	-1	
Bottom Front Centre	Bfc	0°	0	+1	-1	
Back Centre	Cb	+180°	0	-1	0	
Left Screen	Lscr	+SC°	-SC	+1	0	Left edge of screen/display
Right Screen	Rscr	-SC°	+SC	+1	0	Right edge of screen/display
Left Wide	Lw	+60°	-1	+0,667	0	
Right Wide	Rw	-60°	+1	+0,667	0	

Annex B (informative): AC-4 bit-rate calculation

For some systems, knowledge of the approximate audio bit rate is required.

For Constant Bit Rate (CBR) streams (signalled by a `wait_frames` value of 0) the bit rate can be calculated directly from the frame sizes. (Because frame sizes may still vary by 1 byte, a time average of frame sizes will give a better estimate of bit rate.)

For Average Bit Rate (ABR) streams, the following steps outline an algorithm that provides a high precision estimate of the bit rate.

1) Define an admissible sequence of length L as a sequence of l consecutive frames, where all of the following conditions are fulfilled for $1 \leq i < L$:

   - $\texttt{frame\_rate\_index}_i \equiv$ constant (the frame rate remains constant);
   - $\texttt{sequence\_counter}_i$ signals no source change (see ETSI TS 103 190-1 [1], clause 4.3.3.2.2);
   - $1 \leq \texttt{wait\_frames}_i \leq 6$ (the stream is an ABR stream).

NOTE: Here and in the following steps, $\texttt{p}_i$ denotes parameter `p` of frame i, where $0 \leq i < L$.

2) Find a length $n + 2$ admissible sequence where the $\texttt{br\_code}_0 = \texttt{br\_code}_{n+1} = 0b11$ and $\texttt{br\_code}_i \neq 0b11$ for $0 < i < n+1$.

3) Determine the frame rate framerate_{ac4} from the constant `frame_rate_index` parameter of the sequence.

4) Calculate correction factors:

$$F^- = \prod_{i=1}^{n} 2^{\frac{\text{br\_code}_i}{3^i}}$$

and

$$F^+ = F^- \times 2^{\frac{1}{3^n}}$$

5) Find a length m+1 admissible sequence with:

$$N' = \begin{cases} m+\text{wait\_frames}_0 - \text{wait\_frames}_m \geq 3 & \text{if frame\_rate\_idx} \in [0;9] \text{ or frame\_rate\_idx} = 13 \\ m+2\times\text{wait\_frames}_0 - 2\times\text{wait\_frames}_m \geq 6 & \text{if frame\_rate\_idx} \in [10;12] \end{cases}$$

6) Find raw frame sizes S_i in bytes by either reading them directly from the sync frame header (if available), or otherwise deriving them by subtracting the transport overhead to arrive at the raw frame size as described in table B.1.

7) Calculate the overall size of frames 1…m:

$$S_{Tot} = 8 \times \sum_{i=1}^{m} S_i \text{ [bits]}$$

8) Calculate br_{min} and br_{max}:

$$br_{min} = \begin{cases} \frac{S_{Tot}}{N'+1} \times \text{framerate}_{ac4}[\text{bps}] & \text{if frame\_rate\_idx} \in [0;9] \text{ or frame\_rate\_idx} = 13 \\ \frac{S_{Tot}}{N'+2} \times \text{framerate}_{ac4}[\text{bps}] & \text{if frame\_rate\_idx} \in [10;12] \end{cases}$$

$$br_{max} = \begin{cases} \frac{S_{Tot}}{N'-1} \times \text{framerate}_{ac4}[\text{bps}] & \text{if frame\_rate\_idx} \in [0;9] \text{ or frame\_rate\_idx} = 13 \\ \frac{S_{Tot}}{N'-2} \times \text{framerate}_{ac4}[\text{bps}] & \text{if frame\_rate\_idx} \in [10;12] \end{cases}$$

9) Calculate:

$$K_1 = \left\lfloor log_2\left(\frac{br_{min}}{1000[bps]}\right)\right\rfloor$$

$$K_2 = K_1 + 1$$

10) Calculate four bit-rate estimates:

$$br_i^- = F^- \times 2^{K_i}[kbps], i \in \{1; 2\}$$

and

$$br_i^+ = F^+ \times 2^{K_i}[kbps], i \in \{1; 2\}$$

11) Find the index *opt* for which

$$br_{opt}^- < br_{max} \wedge br_{opt}^+ \geq br_{min}$$

12) Calculate R_{min} and R_{max}:

$$R_{min} = max\left(br_{opt}^-, br_{min}\right)[kbps]$$

and

$$R_{max} = min\left(br_{opt}^+, br_{max}\right)[kbps]$$

.

13) Determine the framing overhead bit rate *B* in kbps. If the `ac4_syncframe` format is used, then $B = OH_{syncF} \times framerate_{ac4} \times \frac{8}{1000}[kbps]$, where OH_{syncF} is determined from table B.1.

Table B.1: Overhead for different flavours of the sync frame transport format

`sync_word`	`frame_size` [bits]	Overhead	OH_{syncF} [bytes]
AC40	16	Sync word, length word	4
AC40	24	Sync word, length word	7
AC41	16	Sync word, length word, CRC	6
AC41	24	Sync word, length word, CRC	9

The stream bit rate lies in the interval:

$$[R_{min} + B, R_{max} + B][kbps]$$

Annex C (normative): AC-4 Sync Frame

ETSI TS 103 190-1 [1], Annex G defines a sync frame format and matching media type for AC-4. The AC-4 sync frame is an optional bitstream layer for encapsulating AC-4 raw frames.

If a bitstream complying with the present document is encapsulated in the AC-4 sync frame format, the provisions of [1], Annex G apply.

Frameworks that require a MIME type to signal specific stream properties may further qualify the payload.

EXAMPLE: `audio/ac4; version=02.01.03` can signal an AC-4 sync frame stream with `presentation_version`=1, `bitstream_version`=1 and `md_compat`=3.

Annex D (normative): AC-4 in MPEG-2 transport streams

D.1 Introduction

This appendix contains the elementary information for the integration of AC-4 coded bitstreams in MPEG-2 transport streams. The AC-4 elementary bitstream is included in an MPEG-2 transport stream using packetized elementary stream (PES) packetization and therefore carried in much the same way an MPEG-1 audio stream would be included. AC-4 specific signalling is used to distinguish AC-4 from an MPEG audio stream so that streams can be routed to the correct decoder.

The next clause specifies how AC-4 is carried in an MPEG-2 transport stream and outlines the constraints that need to be taken into account when creating MPEG-2 transport streams that include AC-4. It specifically includes:

- properties of the elementary stream (clause D.2.1);
- properties of the packetized elementary stream (clause D.2.2);
- AC-4 signalling on service information level (clause D.2.3); and
- input buffer size for decoders (clause D.2.4).

D.2 Constraints on carrying AC-4 in MPEG2 transport streams

D.2.1 Audio elementary stream

When AC-4 is multiplexed into an MPEG-2 transport stream, the AC-4 stream shall be formatted using the AC-4 sync frame format as specified in ETSI TS 103 190-1 [1], Annex G.

D.2.2 PES packaging

- The value of `stream_id` in the PES header shall be 0xBD (indicating `private_stream_1`). Multiple AC-4 streams may share the same value of `stream_id` since each stream is carried using a unique PID value.
- The AC-4 elementary stream shall be byte-aligned within the packetized elementary stream (PES) packet payload. The first byte of an AC-4 frame shall reside in the first byte of the PES packet payload.
- One or more AC-4 frames can be packaged into one PES packet.

D.2.3 Service information signalling

- AC-4 content is identified by an AC-4 specific descriptor in the Program Map Table (PMT) descriptor loop following the `ES_info_length` field. The signalling should use an AC-4 specific descriptor defined by application standards.
- Stream_type: the value of `stream_type` for an AC-4 elementary stream shall be set to 0x06 (indicating PES packets containing private data).

D.2.4 T-STD audio buffer size

The main audio buffer size BS_n shall have a fixed value of 131 072 bytes.

Annex E (normative): AC-4 bitstream storage in the ISO base media file format

E.1 Introduction

This annex defines the necessary structures for the integration of AC-4 coded bitstreams in a file format that is compliant with the ISO Base Media File Format [3]. Examples of file formats that are derived from the ISO Base Media File Format include the MP4 file format ISO/IEC 14496-14 [i.9] and the 3rd Generation Partnership Project (3GPP) file format ETSI TS 126 244 [i.10].

This annex additionally covers:

- the steps required to properly packetize an AC-4 bitstream for multiplexing and storage in a DASH compliant ISO base media file format file; and
- the steps required to demultiplex an AC-4 bitstream from a DASH compliant ISO base media file format format file; and
- definition of the Multipurpose Internet Mail Extensions (MIME) *codecs* parameter.

The Box syntax definitions in the subsequent sections shall inherit the ISO base media file format provisions as specified in [3]. This includes the object definition for `Box` and `FullBox` from section 4.2.2, the `AudioSampleEntry` in section 12.2.3 as well as the object syntax conventions in section 4.2.1 for the data types used in the semantics descriptions.

E.2 AC-4 track definition

In the terminology of the ISO base media file format specification [3], AC-4 tracks are audio tracks. It therefore follows that these rules apply to the media box in the AC-4 tracks.

- In the Handler Reference Box, the `handler_type` field shall be set to `soun`.
- The Media Information Header Box shall contain a Sound Media Header Box.
- The Sample Description Box shall contain a box derived from AudioSampleEntry. This box is called `AC4SampleEntry` and is defined in clause E.4.

The value of the `timescale` parameter in the Media Header Box (`mdhd`), referred to as media time scale, depends on *frame_rate* and *base_samp_freq*. The media time scale shall be set according to table E.1.

NOTE: For the definition of samples, see clause E.3.

The value of the *language* parameter in the Media Header Box (`mdhd`) declares the language of the AC-4 audio track. For single-language AC-4 tracks, the value of this field should contain the ISO 639-2/T [i.7] equivalent of the language tag in `language_tag_bytes` in the `AC4SampleEntry`. If the AC-4 track contains multiple presentations in different languages, the *language* field should contain '`mul`'.

The Sample Table Box (`stbl`) of an AC-4 audio track shall contain a Sync Sample Box (`stss`), unless all samples are sync samples. The Sync Sample Box shall reference all sync samples part of that track. For Movie Fragments this corresponds to *sample_is_non_sync_sample* = false. The `sequence_counter` of the first sample should be set to zero.

Table E.1: Timescale for Media Header Box

base_samp_freq **[kHz]**	`frame_rate_index`	***frame_rate*** **[fps]**	**Media time scale [1/sec]**	`sample_delta` **[units of media time scale]**
48	0	23,976	48 000	2 002
	1	24	48 000	2 000
	2	25	48 000	1 920
	3	29,97 (see note 1)	240 000	8 008
			48 000	1 601, 1 602 (see note 2)
	4	30	48 000	1 600
	5	47,95	48 000	1 001
	6	48	48 000	1 000
	7	50	48 000	960
	8	59,94	240 000	4 004
	9	60	48 000	800
	10	100	48 000	480
	11	119,88	240 000	2 002
	12	120	48 000	400
	13	(23,44)	48 000	2 048
	14	Reserved		
	15	Reserved		
44,1	0…12	Reserved		
	13	(21,53)	44 100	2 048
	14, 15	Reserved		
NOTE 1: There are two possible choices for the media time scale. NOTE 2: The `sample_delta` value is non-constant and it changes between the two specified values.				

E.3 AC-4 sample definition

For the purpose of carrying AC-4 in the ISO base media file format, an AC-4 sample corresponds to one `raw_ac4_frame()`, as defined in ETSI TS 103 190-1 [1], clause 4.2.1.

Sync samples are defined as samples that have the `b_iframe_global` flag set in the `ac4_toc()`. AC-4 sync samples are marked as specified in clause E.2. Each AC-4 sync sample is a random access point and stream access point.

The first sample of an AC-4 segment or fragment shall be an AC-4 sync sample.

NOTE: A segment or fragment may contain multiple sync samples.

E.4 AC4SampleEntry Box

Definition

Table E.2: Definition of AC4SampleEntry

Box Type:	`'ac-4'`
Container:	Sample Description Box (`'stsd'`)
Mandatory:	Yes
Quantity:	Exactly one

The `AC4SampleEntry` shall contain an `AC4SpecificBox`, as defined in clause E.5.

The following optional boxes inherited from AudioSampleEntry from ISO/IEC 14496-12 [3] should not be present:

- DownMixInstructions()
- DRCCoefficientsBasic()
- DRCInstructionsBasic()
- DRCCoefficientsUniDRC()
- DRCInstructionsUniDRC()

Syntax

Pseudocode E.1: Syntax of AC4SampleEntry()

```
aligned(8) class AC4SampleEntry extends AudioSampleEntry('ac-4') {
   AC4SpecificBox();
   // we permit any number of AC4PresentationLabel boxes:
   AC4PresentationLabelBox() [];
   Box () [];         // further boxes as needed
}
```

Semantics

The layout of the `AC4SampleEntry` box is identical to that of `AudioSampleEntry` defined in ISO/IEC 14496-12 [3], clause 12.2.3 (including the reserved fields and their values), except that `AC4SampleEntry` additionally contains AC-4 specific boxes at the end, i.e. AC-4 bitstream information called `AC4SpecificBox` and, optionally, one or more `AC4PresentationLabelBox`. The `AC4SpecificBox` field structure for AC-4 is defined in clause E.5 and the `AC4PresentationLabelBox` field structure is defined in clause E.5a.

Additional AC-4 specific requirements on the elements in the `AudioSampleEntry` are provided in table E.3.

Table E.3: AC-4 specific requirements on the elements in AC4SampleEntry box

Element	Data Type	On decoding, the value indicates	On encoding, the value
`Box.size`	unsigned int(32)	the size of the `sampleEntry` box	See note
`Box.type`	unsigned int(32)	N/A	shall be set to 0x61632D34 ('`ac-4`')
`SampleEntry.data_reference_index`	unsigned int(16)	N/A	See note
`AudioSampleEntry.channelcount`	unsigned int(16)	shall be ignored	should be set to the total number of audio output channels of the first presentation of that track; see NOTE
`AudioSampleEntry.samplesize`	unsigned int(16)	N/A	shall be set to 16
`AudioSampleEntry.samplingrate`	unsigned int(32)	shall be ignored	should be set to the base sampling frequency of the track, as indicated by `ac4_toc/fs_index`; see note
NOTE: set according to the Sample Entry definition in ISO/IEC 14496-12 [3], clause 12.2.3.			

E.5 AC4SpecificBox

Definition

Table E.4: Definition of AC4SpecificBox()

Box Type:	'dac4'
Container:	AC4SampleEntry Box ('ac-4')
Mandatory:	Yes
Quantity:	Exactly one

The AC4SpecificBox shall contain an ac4_dsi_v1() data structure as specified in clause E.6.1.

Syntax

Pseudocode E.2: Syntax of AC4SpecificBox()

```
aligned(8) class AC4SpecificBox extends Box('dac4') {
   bit(8)[] ac4_dsi_v1();  // to end of the box
}
```

Semantics

Table E.5: Element description for AC4SpecificBox

Element	Data Type	On decoding, the value indicates	On encoding, the value
Box.size	unsigned int(32)	the size of the box	shall be specified by [3]
Box.type	unsigned int(32)	N/A	shall contain the value 0x64616334 ('dac4').
ac4_dsi_v1()	bit(8)[]	see clause E.6.1	see clause E.6.1

E.5a AC4 Presentation Label Box

Definition

Table E.5a: Definition of AC4PresentationLabelBox()

Box Type:	'lac4'
Container:	AC4SampleEntry Box ('ac-4')
Mandatory:	No
Quantity:	Zero or more

The AC-4 Presentation Label Box provides labels that can be used in a user interface to guide user-driven presentation selection.

The AC4PresentationLabelBox may occur zero or more times after AC4SampleEntryBox/AC4SpecificBox. If there are multiple occurrences of the AC-4 Presentation Label Box, then they shall each have a different language_tag.

Syntax

Pseudocode E.2a: Syntax of AC4PresentationLabelBox()

```
aligned(8) class AC4PresentationLabelBox extends FullBox('lac4', version = 0, 0) {
   unsigned int(16) num_presentation_labels;
   utf8string language_tag;
   for (i=0; i < num_presentation_labels; i++) {
      unsigned int(16) presentation_id;
      utf8string presentation_label;
   }
}
```

Semantics

Table E.5b: Element Description for AC4PresentationLabelBox

Element	Data Type	On decoding, the value indicates	On encoding, the value
Box.size	unsigned int(32)	the size of the `AC4PresentationLabel` box	See note 1
Box.type	unsigned int(32)	N/A	shall be set to 0x6C616334 ('lac4')
FullBox.version	unsigned int(8)	N/A	shall be set to 0
FullBox.flags	bit(24)	N/A	shall be set to 0
`num_presentation_labels`	unsigned int(16)	the number of presentation labels contained in the `AC4PresentationLabelBox`	shall be set to the number of presentation labels that are included in this box.
`language_tag`	null-terminated string using UTF-8 characters	the language of all labels in the containing `AC4PresentationLabelBox`. For different display languages, multiple AC-4 Presentation Label Boxes shall be used.	shall be specified as in IETF BCP 47 [7], encoded as a null-terminated UTF-8 string. If no language tag is provided, the terminating null-byte shall be written.
`presentation_id`	unsigned int(16)	indicates the matching presentation in `ac4_dsi_v1()` for a label. See note 2	shall be set to the `presentation_id` or `extended_presentation_id` of the presentation in the `ac4_toc()` that the following `presentation_label` corresponds to.
`presentation_label`	null-terminated string using UTF-8 characters	a textual label for a presentation	shall be stored as a null-terminated UTF-8-encoded string containing a textual label for the matching presentation. On writing, the string shall be terminated with a null-byte, even when the label is empty.
NOTE 1: set according to the Sample Entry definition in ISO/IEC 14496-12 [3], clause 12.2.3. NOTE 2: The value does not exceed 511 as it is limited by the maximum possible number in `ac4_presentation_v1_dsi/extended_presentation_id`.			

See also clause E.10.2.

E.6 ac4_dsi_v1

E.6.1 Syntax and Semantics

The `ac4_dsi_v1()` structure summarizes the content of all samples referenced by `Ac4SampleEntry` containing the DSI, with elements aligned and sized such that parsing the information involves less bit operations. This information may be used to populate manifest files.

On decoding, if the `ac4_dsi_v1()` structure is available to the decoder, it shall be used for presentation selection. In this case, selection criteria shall be only applied to the information in the `ac4_dsi_v1()` and its substructures.

On encoding, there are certain constraints placed on the values in the `ac4_dsi_v1()` and its substructures, as further detailed in the rest of the present Annex.

Inside the `ac4_dsi_v1()` structure, presentations are represented in an array of `ac4_presentation_v1_dsi()` elements. The number and order of presentations in the `ac4_dsi_v1()` structure need not be the same as in the `ac4_toc()`; in fact, both structures may contain a different number of presentations. Therefore, to identify a presentation in the `ac4_toc()`, a decoder shall match a presentation selected through its `ac4_presentation_v1_dsi()` element to a presentation in the `ac4_toc()` as specified in table E.11; the decoder shall decode the matching presentation.

NOTE 1: In the following, the "matching presentation" is the presentation contained in the `ac4_toc()` that matches as specified in `presentation_id`. See table E.11.

On encoding, each `ac4_presentation_v1_dsi()` should match one presentation.

NOTE 2: Therefore, entries in the `ac4_dsi_v1()` element apply to all samples that reference the `Ac4SampleEntry` containing the DSI. No configuration change can occur inside an `Ac4SampleEntry`.

Syntax	No of bits
ac4_dsi_v1()	
{	
ac4_dsi_version;	3
bitstream_version;	7
fs_index;	1
frame_rate_index;	4
n_presentations;	9
if (bitstream_version > 1) {	
b_program_id;	1
if (b_program_id) {	
short_program_id;	16
b_uuid;	1
if (b_uuid) {	
program_uuid;	16*8
}	
}	
}	
ac4_bitrate_dsi();	
byte_align;	0…7
for (i = 0; i < n_presentations; i++) {	
presentation_version;	8
pres_bytes;	8
if (pres_bytes == 255) {	
add_pres_bytes;	16
pres_bytes += add_pres_bytes;	
}	
if (presentation_version == 0) {	
presentation_bytes = ac4_presentation_v0_dsi();	
}	
else {	
if (presentation_version == 1) {	
presentation_bytes = ac4_presentation_v1_dsi(pres_bytes);	
}	
else {	
presentation_bytes = 0;	
}	
}	
skip_bytes = pres_bytes - presentation_bytes;	
skip_area;	skip_bytes*8
}	
}	
NOTE: The number of bits in `byte_align` pads the number of bits, counted from the start of `ac4_dsi_v1` to a multiple of 8.	

Table E.6: Element description for ac4_dsi_v1() Elements

Element	Data Type	On decoding, the value indicates	On encoding, the value
ac4_dsi_version	uimsbf	the version of the decoder-specific information (DSI). If the ac4_dsi_version field is set to a value greater than 1, decoders conforming to the present document shall discontinue parsing and skip the rest of the box.	shall be written equal to 1 see note
bitstream_version	uimsbf	the version of the bitstream	should be copied from ac4_toc/bitstream_version
frame_rate_index	uimsbf	the frame rate index as specified in ETSI TS 103 190-1 [1], clause 4.3.3.2.6	should be copied from ac4_toc/frame_rate_index
n_presentations	uimsbf	the number of presentations available for selection	should be copied from ac4_toc/n_presentations
short_program_id	uimsbf	a locally unique program ID	should be copied from ac4_toc/short_program_id
program_uuid	uimsbf	a globally unique audio program ID	should be copied from ac4_toc/program_uuid
presentation_version	uimsbf	the presentation version as specified in ETSI TS 103 190-1 [1], clause 4.3.3.4.	should be copied from ac4_presentation_info/presentation_version or ac4_presentation_v1_info/presentation_version, whichever is present in the ac4_toc()
pres_bytes	uimsbf	the length, in bytes, of the adjacent per-audio presentation information section	is set to the length, in bytes, of the following presentation information section, including the skip_area.
ac4_presentation_v0_dsi()		structure representing system-level information of a presentation with presentation_version set to 0 as specified in ETSI TS 103 190-1 [1], clause E.4a	
NOTE: For ac4_dsi_version = 0, see ETSI TS 103 190-1 [1], clause E.4.			

E.7 ac4_bitrate_dsi

E.7.1 Syntax and Semantics

This structure provides information on the bit rate of an AC-4 stream or of individual presentations of an AC-4 stream.

Syntax	No of bits

ac4_bitrate_dsi()
{
  bit_rate_mode;
  bit_rate;
  bit_rate_precision;
}
	
	2
	32
	32

Table E.7: Detailed Description for ac4_bitrate_dsi()

Element	Data Type	On decoding, the value indicates	On encoding, the value
bit_rate_mode	uimsbf	the bit rate control algorithm. The control mode is specified by table E.8.	depends on wait_frames in the ac4_toc(). If wait_frames=0 in the ac4_toc() of all samples referring to this sample entry, set the bit_rate_mode to 1. If in the ac4_toc() of all samples referring to this sample entry, 1≤wait_frames≤6, set the bit_rate_mode to 2. For all other cases set the bit_rate_mode to 3.
bit_rate	uimsbf	the bit rate in bits/second. A value of 0 means that the bit rate is unknown.	should be chosen according to $R_{min} \leq$ bit_rate $\leq R_{max}$ as specified in annex B.
bit_rate_precision	uimsbf	the precision of bit_rate in bits/second. This parameter means that the true bit rate is in the range [bit_rate - bit_rate_precision, bit_rate + bit_rate_precision]. The special value 0xFFFFFFFF indicates that the precision is unknown.	shall be written such that bit_rate – bit_rate_precision $\leq R_{min}$ and $R_{max} \leq$ bit_rate + bit_rate_precision.

Table E.8: bit_rate_mode values

bit_rate_mode	Meaning
0	Bit-rate mode not specified
1	Constant bit rate
2	Average bit rate
3	Variable bit rate

E.8 ac4_presentation_v0_dsi

E.8.1 Introduction

The ac4_presentation_v0_dsi() structure represents system-level information of a presentation contained in the sample entry.

The syntax is defined in ETSI TS 103 190-1 [1].

The current specification defines syntactic elements originally reserved in ETSI TS 103 190-1 [1] with a different naming as indicated in table E.9:

Table E.9: Element mapping for ac4_presentation_v0_dsi()

Element name in this specification	Element name in ETSI TS 103 190-1 [1]
`presentation_v0_channel_groups[]`	`reserved_zero_channel_mask`

E.8.2 Semantics

Table E.10: Element description for ac4_presentation_v0_dsi()

Element	Data Type	On decoding, the value indicates	On encoding, the value
`presentation_v0_channel_groups[]`	bool[]	indicates which audio channel groups are active in the presentation corresponding to table A.27	conforming to the pseudo-code in clause E.10.3 each Boolean in this array shall indicate whether the corresponding audio channel group is present in the original content. The audio channel groups are shown in table A.27 and the right column shows the index that maps to the `presentation_v0_channel_groups[]` array, where the first sent element of the array indicates the channel group with index 17 (Lw and Right Wide (Rw))

E.9 Void

E.10 ac4_presentation_v1_dsi

E.10.0 Introduction

The `ac4_presentation_v1_dsi()` structure represents system-level information of a presentation contained in the sample entry.

E.10.1 Syntax

Syntax	No of bits
ac4_presentation_v1_dsi(pres_bytes)	
{	
presentation_config_v1;	5
if (presentation_config_v1 == 0x06) {	
b_add_emdf_substreams = 1;	
}	
else {	
md_compat;	3
b_presentation_id;	1
if (b_presentation_id) {	
presentation_id;	5
}	
dsi_frame_rate_multiply_info;	2
dsi_frame_rate_fraction_info;	2
presentation_emdf_version;	5
presentation_key_id;	10
b_presentation_channel_coded;	1
if (b_presentation_channel_coded) {	
dsi_presentation_ch_mode;	5
if (dsi_presentation_channel_mode in [11, 12, 13, 14]) {	
pres_b_4_back_channels_present;	1
pres_top_channel_pairs;	2
}	
reserved_zero;	6
presentation_v1_channel_groups[];	18

```
Syntax                                                                  No of bits
    }
    b_presentation_core_differs; ........................................ 1
    if (b_presentation_core_differs) {
      b_presentation_core_channel_coded; ................................ 1
      if (b_presentation_core_channel_coded) {
        dsi_presentation_channel_mode_core; ............................. 2
      }
    }
    b_presentation_filter; .............................................. 1
    if (b_presentation_filter) {
      b_enable_presentation; ............................................ 1
      n_filter_bytes; ................................................... 8
      for (i = 0; i < n_filter_bytes; i++) {
        filter_data; .................................................... 8
      }
    }
    if (presentation_config_v1 == 0x1f) {
      ac4_substream_group_dsi();
    }
    else {
      b_multi_pid; ...................................................... 1
      if (presentation_config_v1 in [0, 1, 2]) {
        ac4_substream_group_dsi();
        ac4_substream_group_dsi();
      }
      if (presentation_config_v1 in [3, 4]) {
        ac4_substream_group_dsi();
        ac4_substream_group_dsi();
        ac4_substream_group_dsi();
      }
      if (presentation_config_v1 == 5) {
        n_substream_groups_minus2; ...................................... 3
        n_substream_groups = n_substream_groups_minus2 + 2;
        for (sg = 0; sg < n_substream_groups; sg++) {
          ac4_substream_group_dsi();
        }
      }
      if (presentation_config_v1 > 5) {
        n_skip_bytes; ................................................... 7
        for (i = 0; i < n_skip_bytes; i++) {
          skip_data; .................................................... 8
        }
      }
    }
    b_pre_virtualized; .................................................. 1
    b_add_emdf_substreams; .............................................. 1
  }
  if (b_add_emdf_substreams) {
    n_add_emdf_substreams; .............................................. 7
    for (j = 0; j < n_add_emdf_substreams; j++) {
      substream_emdf_version; ........................................... 5
      substream_key_id; ................................................. 10
    }
  }
  b_presentation_bitrate_info; .......................................... 1
  if (b_presentation_bitrate_info) {
    ac4_bitrate_dsi();
  }
  b_alternative; ........................................................ 1
  if (b_alternative) {
    byte_align; ......................................................... 0…7
    alternative_info();
  }
  byte_align; ........................................................... 0…7
  if (bits_read() <= (pres_bytes - 1) * 8) {
    de_indicator; ....................................................... 1
    immersive_audio_indicator; .......................................... 1
    reserved; ........................................................... 4
    b_extended_presentation_id; ......................................... 1
    if (b_extended_presentation_id) {
      extended_presentation_id; ......................................... 9
    }
    else {
      reserved; ......................................................... 1
    }
  }
}
```

Syntax	No of bits
NOTE: The number of bits in byte_align pads the number of bits, counted from the start of ac4_presentation_v1_dsi() to a multiple of 8.	

E.10.2 Semantics

Table E.11: Element description for ac4_presentation_v1_dsi()

Element	Data Type	On decoding, the value indicates	On encoding, the value
presentation_config_v1	uimsbf	the number and types of substream groups inside the presentation	shall be set to • the value of presentation_config read from the respective ac4_presentation_v1_info() element, if b_single_substream_group = 0, or • 0x1F, otherwise.
md_compat	uimsbf	the decoder compatibility as described in ETSI TS 103 190-1 [1], clause 4.3.3.3.8.	shall be written equal to ac4_presentation_v1_dsi/md_compat
b_presentation_id	bool	whether the presentation carries a presentation_id that uniquely identifies a presentation in a table of contents	shall be written equal to ac4_toc/ac4_presentation_v1_info/b_presentation_id
presentation_id	uimsbf	a presentation carried in ac4_toc/ac4_presentation_v1_info(). The presentation_id shall be overridden by the value of extended_presentation_id, if ac4_dsi_v1/extended_presentation_id is present in the bitstream.	shall be identical, to match a presentation in ac4_dsi_v1/ac4_presentation_v1_dsi() to a presentation in ac4_toc/ac4_presentation_v1_info()
dsi_frame_rate_multiply_info	uimsbf	frame_rate_multiply_info as described in ETSI TS 103 190-1 [1], clause 4.3.3.5	shall correspond to the respective value read from the ac4_presentation_v1_info() element as shown in table E.12
dsi_frame_rate_fractions_info	uimsbf	the frame_rate_fractions_info as described in clause 6.3.2.4	shall correspond to the respective value read from the ac4_presentation_v1_info() element as shown in table E.13
presentation_emdf_version	uimsbf	the Extensible Metadata Delivery Format (EMDF) syntax version as described in ETSI TS 103 190-1 [1], clause 4.3.3.6.1	shall be the same as the respective emdf_version value read from the emdf_info() field in the ac4_presentation_v1_info()
presentation_key_id	uimsbf	the authentication ID as described in ETSI TS 103 190-1 [1], clause 4.3.3.6.2	shall be the same as the respective key_id value read from the emdf_info() field in the ac4_presentation_v1_info()
b_presentation_channel_coded	bool	that the presentation is channel coded	shall be false if the *pres_ch_mode* is determined as -1, according to pseudocode 25
dsi_presentation_ch_mode	uimsbf	the presentation channel mode	shall equal the value of the *pres_ch_mode* determined according to pseudocode 25, for the case when it is different from -1
pres_b_4_back_channels_present	bool	the presence of non-silent signals in four versus two back channels	shall equal the value of *b_pres_4_back_channels_present* as described in clause 6.3.3.1.29
pres_top_channel_pairs	uimsbf	the presence of a non-silent signal pair in the four top channels	shall equal the value of pres_top_channel_pairs as described in clause 6.3.3.1.30

Element	Data Type	On decoding, the value indicates	On encoding, the value
presentation_v1_channel_groups[]	bool[]	which audio channel groups are active in the presentation corresponding to table A.27	conforming to the pseudo-code in clause E.10.3 each Boolean in this array shall indicate whether the corresponding audio channel group is present in the original content. The audio channel groups are shown in table A.27 and the right column shows the index that maps to the presentation_v1_channel_groups[] array, where the first sent element of the array indicates the channel group with index 17 (Lw and Rw).
b_presentation_core_differs	bool	the presentation allows core decode or full decode	shall be set to true if the *pres_ch_mode_core* according to pseudocode 26 has a value of -1; or in any ac4_substream_group_info() of the presentation: b_channel_coded is false and b_ajoc is true. Otherwise, this value shall be set to false.
b_presentation_core_channel_coded	bool	whether the core decoding mode of an presentation is channel coded	shall be set to false if the pres_ch_mode_core is determined as -1, according to pseudocode 26
dsi_presentation_channel_mode_core	uimsbf	the channel mode of the core decoding mode of an presentation	shall be written as specified in table E.14, where *pres_ch_mode_core* is determined according to pseudocode 26
b_presentation_filter	bool	whether presentation filter data is available for the presentation	shall be the same as the value of b_presentation_filter in the respective ac4_presentation_v1_info() element
b_enable_presentation	bool	whether an presentation is enabled for playback	shall be the same as the value of b_enable_presentation in the respective ac4_presentation_v1_info() element
n_filter_bytes	uimsbf	the length of the following data field for filter data	should be written as 0
filter_data	uimsbf	filter data. Decoders implemented according to the present document shall parse the data but ignore its contents.	this data is reserved and should be written as a sequence of 0x00
b_multi_pid	bool	whether the presentation is split over more than one AC-4 elementary stream, i.e. that the substreams of some substream groups might not be stored in the bitstream.	shall be the same as the respective value read from the b_multi_pid field in the ac4_presentation_v1_info() element
n_substream_groups_minus2	uimsbf	the number of substream groups minus 2 for presentation_config_v1 = 5	shall be the same as the respective *n_substream_groups* value for presentation_config = 5 in the ac4_presentation_v1_info() element
b_pre_virtualized	bool	pre-rendering as described in ETSI TS 103 190-1 [1], clause 4.3.3.3.5	shall be the same as the respective value read from the ac4_presentation_v1_info() element
b_add_emdf_substreams	bool	whether additional extensible metadata delivery format (EMDF) containers are present, as described in ETSI TS 103 190-1 [1], clause 4.3.3.3.6	shall be the same as the respective value read from the ac4_presentation_v1_info() element
substream_emdf_version	uimsbf	the EMDF syntax version as described in ETSI TS 103 190-1 [1], clause 4.3.3.6.1	shall be the same as the respective emdf_version value read from the emdf_info() field in the n_add_emdf_substreams() loop in the ac4_presentation_v1_info() element
substream_key_id	uimsbf	the authentication ID as described in ETSI TS 103 190-1 [1], clause 4.3.3.6.2	shall be the same as the respective key_id value read from the emdf_info() field in the n_add_emdf_substreams() loop in the ac4_presentation_v1_info() element

Element	Data Type	On decoding, the value indicates	On encoding, the value
b_presentation_bitrate_info	bool	the presence of an ac4_bitrate_dsi() element	shall be set to true if b_bitrate_info is set to true in all ac4_substream_info_chan(), ac_substream_info_ajoc() and ac4_substream_info_obj() contributing to the presentation. The values in the following ac4_bitrate_dsi() element shall be calculated as described in clause E.7.1, considering only those substreams contributing to the presentation.
de_indicator	bool	whether the audio presentation provides dialogue enhancement.	shall be set to true if the presentation enables dialogue enhancement, i.e. at least one substream contributing to the presentation has at least one of the flags b_de_data_present or b_dialog_max_gain set to true.
immersive_audio_indicator	bool	whether this presentation contains immersive audio. This indicator has no effect on decoding and is purely informative.	shall be the same as the respective immersive_audio_indicator value read from the ac4_presentation_substream() element.
b_extended_presentation_id	bool	whether an extended_presentation_id element is present	shall be set to true if the value of the presentation_id is 32 or greater, to enable coding it as extended_presentation_id.
extended_presentation_id	uimsbf	an extended_presentation_id that overrides the presentation_id of this presentation. see note	shall be set to the presentation_id value of the corresponding presentation_info() in the ac4_toc() if enabled by b_extended_presentation_id.
NOTE: extended_presentation_id offers the possibility of transmitting presentation_id values larger than 31.			

Table E.12: Determining dsi_frame_rate_multiply_info for v1

frame_rate_index	b_multiplier	multiplier_bit	dsi_frame_rate_multiply_info
2, 3, 4	0	X	00
	1	0	01
	1	1	10
0, 1, 7, 8, 9	0	X	00
	1	X	01
5, 6, 10, 11, 12, 13	X	X	00

Table E.13: AC-4 substream decoder-specific information v1

frame_rate_index	b_frame_rate_fraction	b_frame_rate_fraction_is_4	dsi_frame_rate_fractions_info
10,11,12	False	X	00
	True	False	01
	True	True	10
5, 6, 7, 8, 9	False	X	00
	True	X	01
0,1,2,3,4,13	X	X	00

Table E.14: Determining dsi_presentation_channel_mode_core

pres_ch_mode_core	dsi_presentation_channel_mode_core	Presentation core channel mode
3	0	5.0
4	1	5.1
5	2	5.0.2 core
6	3	5.1.2 core

E.10.3 Presentation channel groups

The value for the elements of the `presentation_v1_channel_groups[]` array, and `presentation_v0_channel_groups[]` respectively, shall be derived from `pres_ch_mode`, `pres_top_channel_pairs`, `b_pres_centre_present` and `b_pres_4_back_channels_present` as shown in pseudocode E.3.

Pseudocode E.3: presentation_channel_groups

```
{
  presentation_v1_channel_groups[0...17] = 0;

  if (pres_ch_mod == 15)
  {
    presentation_v1_channel_groups[17] = 1; // Lw Rw
    presentation_v1_channel_groups[15] = 1; // Cb
    presentation_v1_channel_groups[14] = 1; // Bfc
    presentation_v1_channel_groups[13] = 1; // Bfl Bfr
    presentation_v1_channel_groups[12] = 1; // LFE2
    presentation_v1_channel_groups[11] = 1; // Tc
    presentation_v1_channel_groups[10] = 1; // Tbc
    presentation_v1_channel_groups[9] = 1; // Tfc
  }

  if (pres_ch_mod ≥ 11)
  {

    if (pres_top_channel_pairs == 1)
      presentation_v1_channel_groups[7] = 1; // Tsl Tsr

    if (pres_top_channel_pairs == 2)
    {
      presentation_v1_channel_groups[4] = 1; // Tfl Tfr
      presentation_v1_channel_groups[5] = 1; // Tbl Tbr
    }

    if (b_pres_4_back_channels_present)
      presentation_v1_channel_groups[3] = 1; // Lb Rb
```

```
    if (b_pres_centre_present)
      presentation_v1_channel_groups[2] = 1; // C
  }

  if (2 ≤ pres_ch_mode ≤ 10)
  {

    if (pres_ch_mode ≥ 3)
      presentation_v1_channel_groups[2] = 1; // Ls Rs

    if (pres_ch_mode in [5; 6])
      presentation_v1_channel_groups[3] = 1; // Lb Rb

    if (pres_ch_mode in [7; 8])
      presentation_v1_channel_groups[17] = 1; // Lw Rw

    if (pres_ch_mode in [9; 10])
      presentation_v1_channel_groups[4] = 1; // Tfl Tfr

    presentation_v1_channel_groups[0] = 1; // L R
    presentation_v1_channel_groups[1] = 1; // C
  }

  if (pres_ch_mode == 1)
    presentation_v1_channel_groups[0] = 1; // L R

  if (pres_ch_mode == 0)
    presentation_v1_channel_groups[1] = 1; // C

  if (0)   // not present in any supported channel configuration
    presentation_v1_channel_groups[16] = 1; // Lscr Rscr
  if (0)   // reserved_zero
    presentation_v1_channel_groups[8] = 1;

}
```

E.11 ac4_substream_group_dsi

E.11.1 Syntax

Syntax	No of bits
ac4_substream_group_dsi()	
{	
b_substreams_present;	1
b_hsf_ext;	1
b_channel_coded;	1
n_substreams;	8
for (i = 0; i < n_substreams; i++) {	
dsi_sf_multiplier;	2
b_substream_bitrate_indicator;	1
if (b_substream_bitrate_indicator) {	
substream_bitrate_indicator;	5
}	
if (b_channel_coded) {	
reserved_zero;	6
dsi_substream_channel_groups[];	18
}	
else {	
b_ajoc;	1
if (b_ajoc) {	
b_static_dmx;	1
if (b_static_dmx == 0) {	
n_dmx_objects_minus1;	4
}	
n_umx_objects_minus1;	6
}	
/* Substream composition information */	
b_substream_contains_bed_objects;	1
b_substream_contains_dynamic_objects;	1
b_substream_contains_ISF_objects;	1
reserved;	1

Syntax	No of bits

```
    }
  }
  b_content_type; ............................................................... 1
  if (b_content_type) {
    content_classifier; ......................................................... 3
    b_language_indicator; ....................................................... 1
    if (b_language_indicator) {
      n_language_tag_bytes; ..................................................... 6
      for (i = 0; i < n_language_tag_bytes; i++) {
        language_tag_bytes; ..................................................... 8
      }
    }
  }
}
```

E.11.2 Semantics

Table E.15: Element description for ac4_substream_group_dsi()

Element description for	Data Type	On decoding, the value indicates	On encoding, the value(s)
b_substreams_present	bool	whether the substreams referenced in the substream group are stored inside the track	shall be equal to the value of b_substreams_present in the respective ac4_substream_group_info() element
b_hsf_ext	bool	the availability of spectral data for high sampling frequencies as described in ETSI TS 103 190-1 [1], clause 4.3.3.3.3	shall be the same as the value read from the respective ac4_substream_group_info() element
b_channel_coded	bool	whether the substreams referenced in the substream group are channel coded	shall be equal to the value of b_channel_coded in the respective ac4_substream_group_info() element
n_substreams	uimsbf	the number of audio substreams contained in the substream group	shall be equal to the value of n_lf_substreams in the respective ac4_substream_group_info()
dsi_sf_multiplier	uimsbf	the sampling frequency multiplier as described in ETSI TS 103 190-1 [1], clause 4.3.3.7.3	shall correspond to the value as shown in ETSI TS 103 190-1 [1], table E.5d with the parameters read from the respective ac4_substream_info_obj(), ac4_substream_info_ajoc(), or ac4_substream_info_chan() element in the ac4_substream_group_info() element, referenced by the respective ac4_presentation_v1_info() element.
b_substream_bitrate_indicator	bool	whether substream bitrate information is provided	shall be equal to the value of b_bitrate_info in the respective ac4_substream_info_obj(), ac4_substream_info_ajoc(), or ac4_substream_info_chan() element in the ac4_substream_group_info() element, referenced by the respective ac4_presentation_v1_info() element.

Element description for	Data Type	On decoding, the value indicates	On encoding, the value(s)
substream_bitrate_indicator	uimsbf	the upper limit of the average bit-rate per channel in the substream as described in ETSI TS 103 190-1 [1], clause 4.3.3.7.5.	shall correspond to the bitrate_indicator value read from the respective ac4_substream_info_obj(), ac4_substream_info_ajoc(), or ac4_substream_info_chan() element in the ac4_substream_group_info() element, referenced by the respective ac4_presentation_v1_info() element, expressed through the brate_ind parameter from ETSI TS 103 190-1 [1], table 90.
dsi_substream_channel_groups[]	bool[]	whether the corresponding audio channel group is present in the original content of this substream. The audio channel groups are shown in table A.27; the right column shows the index that maps to the bool[] array. See also the pseudocode in clause E.10.3 and see NOTE 1.	shall be equal to 1 if the corresponding audio channel group is present in the original content of this substream (see NOTE 2)
b_ajoc	bool	whether the substream is coded using the A-JOC coding tool	shall be equal to the value of b_ajoc in the respective ac4_substream_group_info() element
b_static_dmx	bool	whether the A-JOC coded substream has a static downmix	shall be equal to the value of b_static_dmx in the respective ac4_substream_info_ajoc() element in ac4_substream_group_info()
n_dmx_objects_minus1	uimsbf	the number of downmix objects of an A-JOC coded substream	shall be equal to the value of n_fullband_dmx_signals_minus1 in the respective ac4_substream_info_ajoc() element in ac4_substream_group_info()
n_umx_objects_minus1	uimsbf	the number of upmix objects of an A-JOC coded substream	shall be equal to the calculated value of *n_fullband_upmix_signals* - 1 from the respective ac4_substream_info_ajoc() element in ac4_substream_group_info()
b_substream_contains_bed_objects	bool	whether one or more bed objects are contained in the substream. If the substream is A-JOC coded, this element indicates the type of objects contained in the A-JOC upmix. Otherwise, the element indicates the type of objects contained in the substream.	is transmitted only for a substream, where b_channel_coded = **false**. If the substream is: • A-JOC coded: the values shall match the values transmitted in ac4_substream_info_ajoc. • **Not** A-JOC coded: the values shall match the values transmitted in ac4_substream_info_obj().
b_substream_contains_dynamic_objects	bool	whether one or more dynamic objects are contained in the substream	

Element description for	Data Type	On decoding, the value indicates	On encoding, the value(s)
b_substream_contains_ISF_objects	bool	whether one or more intermediate spatial format objects are contained in the substream	
b_content_type	bool	whether content_type information is present as described in ETSI TS 103 190-1 [1], clause 4.3.3.7.7	shall be equal to the value of b_content_type in the respective ac4_substream_group_info()
content_classifier	uimsbf	the content classifier as described in ETSI TS 103 190-1 [1], clause 4.3.3.8.1	shall correspond to the value read from the respective content_type field in the ac4_substream_group_info() element in ac4_presentation_v1_info()
b_language_indicator	bool	whether programme language indication is present as described in ETSI TS 103 190-1 [1], clause 4.3.3.8.2	shall be equal to the value of b_language_indicator in the respective ac4_substream_group_info()
n_language_tag_bytes	uimsbf	the number of subsequent language tag bytes as described in ETSI TS 103 190-1 [1], clause 4.3.3.8.6	shall be set to the number of the language tag bytes written as the next element in this ac4_substream_group_dsi()
language_tag_bytes	uimsbf	a language as described in ETSI TS 103 190-1 [1], clause 4.3.3.8.7	shall be equal to the language tag in langange_tag_bytes or, depending on the b_serialized_language_tag flag, the concatenated version of language_tag_chunk in the respective ac4_substream_group_info() in the respective ac4_presentation_v1_info().
NOTE 1: The first element of the array in bitstream order indicates the channel group with index 17 (Lw/Rw). NOTE 2: The original content may be derived from b_centre_present, b_4_back_channels_present and top_channels_present. See clause 6.3.2.7.1.			

E.12 alternative_info

E.12.1 Syntax

Syntax	No of bits

```
alternative_info()
{
  name_len; ............................................................ 16
  presentation_name; ............................................. 8 * name_len
  n_targets; ............................................................ 5
  for (t = 0; t < n_targets; t++) {
    target_md_compat; ................................................... 3
    target_device_category; ............................................. 8
  }
}
```

E.12.2 Semantics

Table E.16: Element description for alternative_info()

Element	Data Type	On decoding, the value indicates	On encoding, the value
`name_len`	uimsbf	the length of the following `presentation_name[]` string	
`presentation_name[]`	uimsbf[]	the name of the presentation as a string using UTF-8 coding (ISO/IEC 10646 [2]).	
`n_targets`	uimsbf	the value of `n_targets_minus1` + 1 with `n_targets_minus1` as defined in clause 6.3.3.1.5	
`target_md_compat`	uimsbf	the value of `target_level` as defined in clause 6.3.3.1.6	
`target_device_category`	uimsbf	the respective field defined in clause 6.3.3.1.7	

E.13 The MIME codecs parameter

The MIME *codecs* parameter shall conform to the syntax described in [4]. The value of the parameter shall be set to the dot-separated list of the four following parts of which the latter three are represented by two-digit hexadecimal numbers:

1) the fourCC *ac-4*.
2) `bitstream_version` as indicated in the `ac4_dsi_v1()`.
3) `presentation_version` as indicated for the presentation in the `ac4_dsi_v1()`.
4) `md_compat` as indicated for the presentation in the `ac4_dsi_v1()`.

EXAMPLE: A valid `codecs` value for a presentation is *ac-4.02.01.03*, signalling AC-4 audio with `bitstream_version`=2, `presentation_version`=1 and `md_compat`=3.

Annex F (informative): Decoder Interface for Object Audio

F.1 Introduction

An AC-4 decoder may output an object essence and corresponding object metadata that should be used by an object audio renderer.

ETSI TS 103 448 [i.2] specifies an AC-4 object audio renderer for consumer use. As specified in clause 5.9, the object essence and object metadata are time synchronized after decoding. The object audio output metadata corresponding to one object essence can be considered in different groups, according to the function of the metadata in the rendering process.

F.2 Room-anchored position metadata

Room-anchored position metadata is specified by room-anchored coordinates. Room-anchored objects are typically used for off-screen effects. Dynamic objects have room-anchored position metadata.

Coordinates are specified in the normalized room coordinate system shown in figure 13. Table F.1 shows the position metadata output by the decoder.

Table F.1: Room-anchored position metadata output

Metadata	Description
`object_position_X`	Determines the position of the object on the X axis.
`object_position_Y`	Determines the position of the object on the Y axis.
`object_position_Z`	Determines the position of the object on the Z axis.

The decoder derives the room-anchored position metadata from

- `pos3D_{X;Y;Z}`,
- `diff_pos3D_{X;Y;Z}`,
- `ext_prec_pos3D_{X;Y;Z}`,
- `alt_pos3D_{X;Y;Z}`.

See also clause 6.3.9.8.4.

F.3 Speaker-anchored position metadata

Bed objects have speaker-anchored position metadata, i.e. bed objects are anchored to speaker positions. Typically, beds and bed objects are used to present channel-based audio content like complex ambiances, reverb, or music in combination with more dynamic objects.

Bed objects convey an object audio metadata parameter indicating the loudspeaker the bed object is expected to be rendered to. Table F.2 shows the bed metadata output by the decoder.

Table F.2: Speaker-anchored position metadata output

Metadata	Description
`channel`	Indicates the loudspeaker that the bed object is expected to be rendered to.

The decoder derives the speaker-anchored position metadata from:

- `bed_chan_assign_code`,
- `std_bed_channel_assignment`,
- `nonstd_bed_channel_assignment`.

See also clause 6.3.2.10.8.2.

F.4 Screen-anchored position metadata

Screen-anchored position metadata provides the means to adapt the room-anchored position of objects to take account of different screen sizes, preserving the collocation of audio and visual events in the playback environment.

The position of the L and R speakers can vary greatly in consumer playback environments (from being adjacent to the screen to being wider, sometimes much wider, than the screen). Without screen anchoring, audio and visual events that would be collocated in the mastering environment might become unaligned in the playback environment. Table F.3 shows the screen-anchored metadata output by the decoder.

Table F.3: Screen-anchored position metadata output

Metadata	Description
`object_screen_factor`	The scaling factor applied to the X and Z dimensions for objects that pan across the screen.
`y_position_exponent`	An exponent to be applied to the Y position metadata value. This value is derived from the object_depth_factor, which indicates the rate at which X and Z position scaling converges as the object approaches the screen.

The decoder derives the screen-anchored position metadata from:

- `object_screen_factor_code`,
- `object_depth_factor`.

See also clause 6.3.9.8.17, and clause 6.3.9.8.18.

F.5 Gain metadata

The decoder provides a gain value to apply an explicit gain modification in the object audio renderer processing. This is a convenience functionality that might enable more efficient implementations.

Table F.4 shows the gain metadata output by the decoder.

Table F.4: Gain metadata output

Metadata	Description
`object_gain`	The value of object gain in dB.

The decoder derives the gain metadata from:

- `b_default_basic_info_md`,
- `object_gain_value`,
- `object_gain_code`.

See also clause 6.3.9.7.2, clause 6.3.9.7.5, and clause 6.3.9.7.4.

F.6 Size metadata

The decoder provides data to modify the apparent spatial extent of the object. Using this data, large objects can be efficiently represented.

NOTE: This control does not change the total object loudness.

Table F.5 shows the size metadata output by the decoder.

Table F.5: Size metadata output

Metadata	Description
object_width	The value of the 3D object width with identical extent into all three dimensions (see note).
object_width_X	The X-axis value of the 3D object width.
object_width_Y	The Y-axis value of the 3D object width.
object_width_Z	The Z-axis value of the 3D object width.
NOTE: Either *object_width* or the triplet *object_width_{X;Y;Z}* is output of the decoder as indicated by `object_width_mode`.	

The decoder derives the size metadata from:

- `object_width`,
- `object_width_X`,
- `object_width_Y`,
- `object_width_Z`.

See also clause 6.3.9.8.12, clause 6.3.9.8.13, clause 6.3.9.8.14, clause 6.3.9.8.15, and clause 6.3.9.8.16.

F.7 Priority metadata

The decoder provides an indication of object priority. Object priorities can support device and playback environment adaptable rendering.

EXAMPLE: Two objects that were spatially separated in a 5.1.2 speaker configuration might end up collocated when rendered in 5.1 or in stereo; a renderer could decide to slightly move or attenuate the lower priority object in order to make sure that the higher priority object is clearly perceived.

Table F.6 shows the priority metadata output by the decoder.

Table F.6: Priority metadata output

Metadata	Description
object_priority	Indicates the priority of the object. The higher the priority, the greater the importance of the object.

The decoder derives the priority metadata from `object_priority_code`.

See also clause 6.3.9.7.6.

F.8 Zone constraints metadata

The zone constraints metadata can be used to include or exclude specific groups of speakers (referred to as zones) in the rendering process.

Table F.7 shows the zone metadata output by the decoder.

Table F.7: Zone constraints metadata output

Metadata	Description
zone_mask	Indicates the speaker zones to be included or excluded.
b_enable_elevation	Indicates whether rendering to height speakers is enabled.

The decoder derives the zone metadata from:

- `zone_mask`,
- `b_enable_elevation`.

See also clause 6.3.9.8.7, and clause 6.3.9.8.8.

F.9 Divergence metadata

Divergence metadata is used to define how much energy is sent to locations other than that defined by the position metadata.

Divergence is a commonly used rendering mode in broadcast workflows. It is generally used to progressively spread the energy away from the Centre channel to the Left and Right channels.

Table F.8 shows the divergence metadata output by the decoder.

Table F.8: Divergence metadata output

Metadata	Description
`object_divergence`	The divergence value to be applied to an object in the rendering process.

The decoder derives the divergence metadata from:

- `object_div_mode`,
- `object_div_table`,
- `object_div_code`.

See also clause 6.3.9.8.21, clause 6.3.9.8.22, and clause 6.3.9.8.23.

F.10 Snap metadata

The snap metadata indicates that an object is expected to be rendered with maximal timbral fidelity and spatial localisation.

Table F.9 shows the snap metadata output by the decoder.

Table F.9: Snap metadata output

Metadata	Description
b_object_snap	Indicates whether the object is expected to be rendered with maximal timbral fidelity and spatial localisation.

The decoder derives the snap metadata from `b_object_snap`.

See also clause 6.3.9.8.9.

F.11 Timing metadata

Timing metadata specifies how updates of the object properties are synchronized to the PCM audio samples of the object essence.

Timing metadata can be used in the rendering process to control the slope of smoothing processes. The smoothing can be used to suppress audible artefacts (i.e. spectral cross-products and so-called "zipper noise") caused by rapid control parameter changes; this data enables rapid (less smoothed) signal changes when desired.

NOTE: Non-adaptive smoothing would limit the velocity of fast-moving objects; using this control, responsiveness can be traded off against artefacts caused by rapid gain changes.

Table F.10 shows the snap metadata output by the decoder.

Table F.10: Timing metadata output

Metadata	Description
num_obj_info_blocks	Indicates the number of metadata updates.
ramp_duration	Indicates the duration of smoothing processes between metadata updates in units of PCM samples.
sample_offset	Indicates a timing offset in units of PCM samples applicable to all metadata updates.
block_offset_factor	Indicates a timing offset in units of PCM samples for the corresponding metadata update.

The decoder derives the timing metadata from:

- `num_obj_info_blocks`,
- `ramp_duration`,
- `ramp_duration_table`,
- `ramp_duration_code`,
- `oa_sample_offset_type`,
- `oa_sample_offset`,
- `oa_sample_offset_code`,
- `block_offset_factor`.

See also clause 6.3.9.3.6, clause 6.3.9.3.8, clause 6.3.9.3.11, clause 6.3.9.3.9, clause 6.3.9.3.3, clause 6.3.9.3.5, clause 6.3.9.3.4, and clause 6.3.9.3.7.

F.12 Trim metadata

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

Table F.11 shows the trim metadata that is output by the decoder for each of the nine trim configurations.

Table F.11: Trim metadata output per trim configuration

Metadata	Description
trim_mode	One of {*disable_trim*; *default_trim*; *custom_trim*}
trim_centre	Centre trim value in dB.
trim_surround	Surround trim value in dB.
trim_height	Height trim value in dB.

The decoder derives the trim metadata from:

- `global_trim_mode`,
- `b_default_trim`,
- `b_disable_trim`,
- `trim_centre`,
- `trim_surround`,
- `trim_height`.

See also clause 6.3.9.10.

Annex G (normative): AC-4 in MPEG-DASH

G.1 Introduction

This annex describes the requirements for delivering AC-4 streams with `bitstream_version` > 0 compliant with DASH [5] and ISO base media file format (ISOBMFF) [3].

NOTE: ETSI TS 103 190-1 [1], Annex F specifies requirements for usage of AC-4 with DASH where `bitstream_version` = 0.

G.2 AC-4 specific settings

G.2.1 The @codecs attribute

The *@codecs* attribute shall be set to the value of the MIME *codecs* parameter, as specified in clause E.13.

Clients implementing AC-4 playback compliant with the present specification shall support an *@codecs* indication of `codecs="ac-4.02.01.0x"`, $0 \leq x \leq n$ where n is the compatibility level of the decoder (`md_compat`).

G.2.2 The @audioSamplingRate attribute

If the *@audioSamplingRate* attribute is present on any element, it shall be set to the sampling frequency derived from the parameters `fs_index` and `dsi_sf_multiplier`, contained in `ac4_dsi_v1()`.

EXAMPLE: `<AdaptationSet ... audioSamplingRate="48000">` for 48kHz sampling frequency.

G.2.3 The preselection element

If preselection elements are included in the given DASH profile, each instance of `ac4_presentation_v1_dsi()` should be signalled by a *preselection* element. *AdaptationSet* elements that are referenced by these *preselection* elements should signal the properties of the AC-4 presentation with the widest compatibility.

If preselection elements are not included in the given DASH profile, signalling of the properties of individual presentations is not possible. *AdaptationSet* elements should signal the properties of the AC-4 presentation with the widest compatibility.

G.3 AC-4 specific DASH descriptor schemes

G.3.1 Virtualized audio for headphones

If the content of an AC-4 presentation has been virtualized and tailored for exclusive consumption via headphones as signalled by `ac4_presentation_v1_dsi/b_pre_virtualized` flag, a virtualized audio for headphones DASH supplemental property descriptor should be used with the following requirements:

- the *@schemeIdUri* attribute shall be `tag:dolby.com,2016:dash:virtualized_content:2016`; and
- the *@value* attribute shall be 1.

EXAMPLE:
```
<SupplementalProperty
schemeIdUri="tag:dolby.com,2016:dash:virtualized_content:2016" value="1"/>.
```

G.3.2 Audio frame rate

The frame rate of AC-4 audio should be signalled by means of a DASH essential property descriptor or a DASH supplemental property descriptor with the following requirements:

- the *@schemeIdUri* attribute shall be set to `tag:dolby.com,2017:dash:audio_frame_rate:2017`; and
- the *@value* attribute shall be set to the value of *frame_rate* as specified in clause E.2, derived from `base_samp_freq` and `frame_rate_index`. Its format should be compliant to the XML schema definition of `FrameRateType` in [5].

EXAMPLE: `<SupplementalProperty schemeIdUri="tag:dolby.com,2017:dash:audio_frame_rate:2017" value="30000/1001"/>` for a frame rate of 29.97 fps.

G.3.3 Audio channel configuration schemes

G.3.3.1 Applicability of Audio channel configuration schemes

Where `bitstream_version` > 0, the *AudioChannelConfiguration* DASH descriptor should use one of the following schemes:

- for all AC-4 channel configurations that are mappable to the channel configuration scheme as specified in ISO/IEC 23091-3:2018 [i.6], the scheme described by the schemeIdUri `urn:mpeg:mpegB:cicp:ChannelConfiguration`; or
- the "Dolby:2015" *ChannelConfigurationDescriptor* scheme as specified in clause G.3.3.2.

NOTE: It is preferred to use the scheme described by the schemeIdUri `urn:mpeg:mpegB:cicp:ChannelConfiguration`. The prefix of this Uniform Resource Identifier (`urn:mpeg:mpegB:cicp:`) is defined in ISO/IEC 23091-1:2018 [i.5].

See clause G.3.3.3 for further details on mapping of AC-4 channel configurations to the DASH element *@value* utilizing the above-mentioned schemes.

G.3.3.2 The "Dolby:2015" audio channel configuration scheme

The "Dolby:2015" audio channel configuration descriptor is defined with the following requirements:

- the *@schemeIdUri* attribute shall be set to `tag:dolby.com,2015:dash:audio_channel_configuration:2015`.
- The *@value* attribute shall be set to a six-digit hexadecimal representation of a 24-bit integer as follows:
  - the most-significant bit, i.e. bit 23, indicates whether the content is object audio content - it is cleared if `b_presentation_channel_coded` is set to 1,
  - bits 18 ... 22 are reserved (cleared), and
  - bit *n* in {0 ... 17} is set to `ac4_dsi_v1/presentation_channel_group[17 - n]`.

NOTE: The contents of the *@value* field are case insensitive. Both uppercase and lowercase letters are allowed.

EXAMPLE 1: For a stream with an 5.1.2 immersive audio channel configuration using loudspeakers L, R, C, Ls, Rs, Tsl, Tsr, LFE, the signalling is `<AudioChannelConfiguration schemeIdUri="tag:dolby.com,2015:dash:audio_channel_configuration:2015" value="0000C7"/>` (the hexadecimal equivalent of the binary value 0000 0000 0000 0000 1100 0111).

EXAMPLE 2: For a stream with object-based audio content the signalling is `<AudioChannelConfiguration schemeIdUri="tag:dolby.com,2015:dash:audio_channel_configuration:2015" value="800000"/>` (the hexadecimal equivalent of the binary value 1000 0000 0000 0000 0000 0000).

G.3.3.3 Mapping of channel configurations to the MPEG audio channel configuration scheme (informative)

Most AC-4 channel configurations can be mapped to a value with `@schemeIdUri = urn:mpeg:mpegB:cicp:ChannelConfiguration`.

Table G.1 provides a mapping from `ac4_dsi_v1/presentation_v1_channel_groups[]` to a value with `@schemeIdUri = "urn:mpeg:mpegB:cicp:ChannelConfiguration"`. The mapping is based on the speaker configurations specified in ISO/IEC 23091-3:2018 [i.6], Table 9.

Table G.1: Mapping of AC-4 channel configurations to MPEG scheme

`presentation_v1_channel_groups[]` in hexadecimal representation	Value with `@schemeIdUri = "urn:mpeg:mpegB:cicp:ChannelConfiguration"`
000002	1
000001	2
000003	3
008003	4
000007	5
000047	6
020047	7
008001	9
000005	10
008047	11
00004F	12
02FF7F	13
06FF6F	13
000057	14
040047	14
00145F	15
04144F	15
000077	16
040067	16
000A77	17
040A67	17
000A7F	18
040A6F	18
00007F	19
04006F	19
01007F	20
05006F	20

Annex H (normative): AC-4 bit streams in CMAF

H.1 AC-4 CMAF Tracks

H.1.1 Introduction

This annex defines how to encode and package AC-4 in Common Media Application Format (CMAF), specified in ISO/IEC 23000-19 [6]. The definitions in the present annex convey encoding constraints that apply to all CMAF Tracks containing AC-4 audio as defined in the present document and storage constraints of those AC-4 CMAF Tracks in ISOBMFF files, conforming to annex E. Based on those requirements, clause H.4 defines the AC-4 CMAF Media Profiles and the corresponding CMAF Track Brands.

H.1.2 Track constraints

H.1.2.1 General

AC-4 CMAF tracks shall conform to the CMAF track format as specified in ISO/IEC 23000-19 [6], section 7 and to the CMAF audio track format as specified in ISO/IEC 23000-19 [6], section 10.2. Elementary streams contained in AC-4 CMAF tracks shall conform to the present document; the encapsulation into ISO base media file format (ISOBMFF) tracks shall conform to annex E.

The following additional constraints apply to AC-4 CMAF tracks:

- the `bitstream_version` shall be 2; and
- the `presentation_version` shall be 1; and
- the number of presentations in an elementary stream shall be 64 or less; and
- if the track contains multiple AC-4 presentations, each presentation shall have a unique `presentation_id`; and
- for every presentation `presentation_id` shall be present in every AC-4 sample; and
- all AC-4 samples in the AC-4 CMAF track shall have equivalent configurations as per clause H.1.2.4.

H.1.2.2 Track constraints for multi-stream scenarios

If a presentation references audio program components (i.e substream groups) that are distributed over multiple CMAF tracks, the following constraints apply to all contributing tracks:

- all tracks shall have time-aligned CMAF fragments with equal duration; and
- the value of `frame_rate_index` shall be identical for all tracks; and
- each presentation shall have synchronized `sequence_counter`; and
- all `ac4_presentation_v1_info()` that share the same `presentation_id` shall be identical except for the `group_index` of the `ac4_sgi_specifier()`; and
- all `ac4_substream_group_info()` that are referenced by presentations that share the same `presentation_id` shall be identical except for `b_substreams_present` and `substream_index`.

If an audio program references audio program components (i.e. substream groups) that are distributed over multiple CMAF tracks, at least one track shall contain a self-contained presentation, i.e. all referenced audio program components are present in that track.

H.1.2.3 Track constraints for single-stream scenarios

If an audio program references audio program components (i.e. substream groups) that are entirely contained in one single CMAF track, the following constraint applies to this track:

- the `b_multi_pid` flag shall be false for all presentations.

H.1.2.4 Constraints for equivalent configurations

Two `ac4_toc()` instances have an equivalent configuration if they have identical values of the following parameters:

- `frame_rate_index`,
- `fs_index`,
- `n_presentations`,
- for each `ac4_presentation_v1_info()[i]`, where i is the index into the list of `ac4_presentation_v1_info()` elements as they appear in the `ac4_toc()`:
  - `b_single_substream_group` and, if present, `presentation_config`;
- for each `ac4_substream_group_info()[j]`, where j is the index into the list of `ac4_substream_group_info()` elements as they appear in the `ac4_toc()`:
  - for the `content_type()`:
    - `content_classifier`
    - `b_language_indicator` and, if present, the primary language subtag of the `language_tag_bytes` or of the concatenated version of data from all `language_tag_chunk`
  - for each `ac4_substream_info_*()[k]`, where k is the index into the list of `ac4_substream_info_*()` elements as they appear in the corresponding `ac4_substream_group_info()`:
    - `channel_mode` for each `ac4_substream_info_chan()[k]`,
    - if present, `b_sf_multiplier` and `sf_multiplier`

H.1.3 Signalling

Clause E.13 specifies the MIME *codecs* parameter that is used for signalling of AC-4 tracks.

H.1.4 Codec delay compensation

H.1.4.1 Introduction

The information in this clause specifies how to compensate the need for the overlap-add delay of the codec either with the use of an edit list or before encapsulation.

H.1.4.2 Delay compensation using an edit list

An edit list can be used to compensate any codec delay. When an edit list is used, a single *EditListBox* shall be recorded in the CMAF Header, as specified in ISO/IEC 23000-19 [6], section 7.5.12.

H.1.4.3 Delay compensation before encapsulation

Delay can be compensated before stream encapsulation using the techniques specified in ISO/IEC 23000-19 [6], Annex G.5. No *EditListBox* is required and thus no further delay compensation in the receiver.

H.1.5 Dynamic Range Control and Loudness

If compliance with one of the loudness regulations from ETSI TS 103 190-1 [1], table 156 is required, the following constraints apply:

- the `dialnorm_bits` shall be used to indicate the audio programme loudness measured according to local loudness regulations; and
- the `loud_prac_type` shall accordingly be set to indicate the measurement practice used; and
- the `b_loudcorr_type` flag shall be set to zero, if the audio programme has been corrected with an infinite look-ahead (file-based); if the loudness correction was based on a combination of realtime loudness measurement and dynamic range compression, the flag shall be set to one.

NOTE: See EBU Tech 3344 [i.4] for further information on EBU R 128 [i.3] compliance.

H.2 Random access point and stream access point

Each AC-4 sync sample is a random access point and stream access point. The AC-4 sync sample is specified in clause E.3.

H.3 Switching sets

To enable seamless switching, the requirements in the present clause shall apply to CMAF switching sets containing AC-4 CMAF tracks.

All AC-4 samples in all CMAF tracks comprising a switching set shall have equivalent configurations as per clause H.1.2.4, with the exception that `b_sf_multiplier` and `sf_multiplier` may be different across different AC-4 CMAF tracks.

For time alignment between CMAF tracks, encoders shall time align all CMAF tracks contained in the CMAF switching set such that the decoded PCM samples align. AC-4 sync samples / stream access points as defined in clause E.3 shall be temporally aligned across all CMAF tracks in a CMAF switching set.

NOTE 1: According to ISO/IEC 23000-19 [6], section 7.3.4.1, item j and table 11, an optionally present offset edit list needs to be identical in all CMAF headers in a CMAF switching set.

NOTE 2: These constraints guarantee single initialization according to ISO/IEC 23000-19 [6], section 7.3.4.2. In consequence a CMAF switching set single initialization constraints identifier according to ISO/IEC 23000-19 [6], section as 7.3.4.3 may be used.

H.4 AC-4 CMAF media profiles

Table H.1 lists AC-4 CMAF media profiles with the corresponding CMAF track constraints and the compatibility brand.

The AC-4 CMAF main profile should always be the default profile, except where the system is limited to single-stream processing.

NOTE: The AC-4 CMAF single-stream profile is a subprofile of the AC-4 CMAF main profile.

Table H.1: AC-4 CMAF media profiles

Name	Compatibility brand	CMAF track constraints
AC-4 CMAF main	ca4m	shall conform to: • Clause H.1.2.1 • Clause H.1.2.2 • Clause H.3
AC-4 CMAF single-stream	ca4s	shall conform to: • Clause H.1.2.1 • Clause H.1.2.3 • Clause H.3

History

Document history		
V1.1.1	September 2015	Publication
V1.2.1	February 2018	Publication
V1.3.1	July 2025	Publication