ETSI TS 103 190-2 V1.1.1 (2015-09)

TECHNICAL SPECIFICATION

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

Reference

DTS/JTC-029-2

Keywords

audio, broadcasting, codec, content, digital, distribution, object audio, personalization

ETSI

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

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

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

Contents

Intellectual Property Rights

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

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

Foreword

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

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".

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

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

The symbolic source code for tables referenced throughout the present document is contained in archive ts_10319002v010101p0.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 [1] provides a bitrate-efficient coding scheme for common broadcast and broadband delivery environment use cases. [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*, providing the same high quality of experience for dialogue intelligibility, video description, and multiple languages as is provided by the main service.
- Advanced *Loudness & 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 simplifying audio and video time base correction (A/V sync management) in the compressed domain for contribution and distribution applications.
- Built-in *robustness*, enabling adaptive streaming and advertisement insertions, switching bitrate and channel configuration without audible artifacts.

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*, ensuring 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 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 [1], the present document specifies additional functionality that may 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 [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. Specification of such a renderer, and its use with the technology specified in the present document, is expected to be referenced or documented in an upcoming revision of, or supplement to, the present document.

2 References

2.1 Normative references

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

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

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

2.2 Informative references

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

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

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

[i.1] Recommendation ITU-R BS.1770-3: "Algorithms to measure audio programme loudness and true-peak audio level".

3 Definitions, symbols, abbreviations and conventions

3.1 Definitions

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

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

A-JOC substream: object audio substream containing A-JOC coded content

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

(audio) track: signal representing one channel or object essence comprising multiple audio samples

alternative presentation: presentation providing alternative object properties

associated audio substream: substream type specified in clause 4.8.3.2

bed: group of multiple bed objects

bed object: audio object whose location property specifies the loudspeaker the audio content is intended to be rendered to

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)

channel configuration: targeted speaker layout

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

coefficient: time-domain sample transformed into frequency domain

companding: compression and expanding in the QMF-domain to achieve temporal shaping of the core encoder quantization noise

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 possible decoding modes for AC-4 decoder implementations specified by the present document

dialogue substream: substream type specified in clause 4.8.3.2

dynamic object: audio object whose location property specifies a dynamic location

frame: cohesive section of bits in the bitstream

frame length: temporal extent of a frame when decoded to PCM

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

frame size: extent of a frame in the bitstream domain

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

full decoding: one of two possible decoding modes for AC-4 decoder implementations specified by the present document

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

I-frame: independently decodable frame

immersive channel audio: audio channel content with an immersive channel configuration

immersive channel configuration: one of the channel configurations listed in clause A.3 in Table A.31 through Table A.42

immersive object audio: audio object content extending beyond the plane

input channel mode: coded representation of the input channel configuration

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

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

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

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

OAMD substream: a portion of the AC-4 bitstream coded in `oamd_substream`

object essence: comprises the audio track information of an object, excluding corresponding object properties

object essence decoder: sum of all decoding tools that are required to decode the object essence

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

output channel configuration: channel configuration present at the output of the AC-4 decoder

presentation: set of one or more AC-4 substreams to be presented simultaneously

presentation configuration: set of metadata associated to a presentation

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

(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

speaker feed: audio data for a dedicated speaker in non-channel based use cases

spectral frontend: tool used to decode the encoded spectral data before feeding it into the windowing and IMDCT blocks

NOTE: There are two different frontends in AC-4: The Speech Spectral Frontend (SSF), and the Audio Spectral Frontend (ASF).

spectral line: frequency coefficient

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

window: weighting function associated with the IMDCT transform 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
$\lceil x \rceil$	round x towards plus infinity
$\lfloor x \rfloor$	round x towards minus infinity
(A|B)	concatenation of matrix A with matrix B

3.3 Abbreviations

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

ABR	Average Bit Rate
AC	Audio Codec
A-CPL	Advanced CouPLing
A-JCC	Advanced Joint Channel Coding
A-JOC	Advanced Joint Object Coding
ASF	Audio Spectral Frontend
ASPX	Advanced Spectral Extension
A-SPX	Advanced SPectral eXtension tool
BC	Back Centre
BL	Back Left
BR	Back Right
CAS	Channel Audio Substream
CBR	Constant Bit Rate
CPL	Coupling
CRC	Cyclic Redundancy Check
DE	Dialogue Enhancement
DRC	Dynamic Range Control
DSI	Decoder Specific Info
EBU	European Broadcasting Union
EMDF	Extensible Metadata Delivery Format
ESM	Elementary Stream Muxing
FC	Front Centre
FL	Front Left
FPS	Frames Per Second
FR	Front Right
IMDCT	Inverse Modified Discrete Cosine Transform
ISF	Intermediate Spatial Format
ISO	International Organization for Standardization
ITU-R	International Telecommunication Union - Radiocommunication
JCC	Joint Channel Coding
JOC	Joint Object Coding
LFE	Low-Frequency Effects
LSB	Least Significant Bit
MME	Main or Music & Effects
MPEG	Motion Picture Experts Group
MSB	Most Significant Bit
OAMD	Object Audio MetaData
OAMDI	Object Audio Metadata Interpreter
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
RTLL	Real Time Loudness Leveler
SAP	Stereo Audio Processing
SCPL	Simple Coupling
S-CPL	Simple CouPLing
SF	Spectral Frontend
SMP	Stereo and Multichannel Processing
SPX	Spectral Extension
SSF	Speech Spectral Frontend
STD	Standard
TL	Top Left
TOC	Table of Contents
TR	Top Right

T-STD	Transport System Target Decoder
UI	User Interface
UTF	Unicode Transformation Format
UUID	Universally Unique IDentifier

Please also refer to clause A.3 for a listing of speaker location abbreviations.

3.4 Conventions

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

- Constants are indicated by upper-case 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 lower-case names, e.g. **vector** or $\mathbf{vector}_{\mathbf{idx}}$.
- Matrices (and vectors of vectors) are indicated by bold upper-case 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. `dynrng`. All bitstream elements are described in clause 6.3.
- Descriptions of Boolean elements use the format "The `element name` flag indicates whether *statement about condition*". If the Boolean is true, the *statement about condition* applies.

EXAMPLE: The `b_element_present` flag indicates whether the element is present in the bitstream.

- Normal pseudocode interpretation of flowcharts is assumed, with no rounding or truncation unless explicitly stated.
- Units of [dB_2] refer to the approximation $1\ dB \equiv$ factor of $\sqrt[6]{2,0}$, i.e. 6 dB $\equiv$ 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.

Table 1: Shorthand notation for frame rates

Fractional framerate	Shorthand
$24 \times {}^{1000}/_{1001}$	23,976
$30 \times {}^{1000}/_{1001}$	29,97
$48 \times {}^{1000}/_{1001}$	47,952
$60 \times {}^{1000}/_{1001}$	59,94
$120 \times {}^{1000}/_{1001}$	119,88

Table 2: Standard functions used in pseudocode

Function	Semantic
`abs(arg)`	$\|arg\|$
`sqrt(arg)`	$arg^{0,5}$
`pow(arg1, arg2)`	$arg1^{arg2}$

Table 3: Symbols for speaker/channel configurations

Symbol	Speakers
f	front
s	surround
h	height
x	LFE
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 audio* coding and describes differences to traditional *channel 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 4.5.1 describes the 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 two 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 coding*, a set of tracks is implicitly assigned to specific loudspeakers by associating the set of tracks with a channel configuration. If the playback speaker configuration is different from the coded channel configuration, downmixing or upmixing specifications are required to redistribute audio to the available speakers.

In *object audio coding*, rendering is applied to *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 three different 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 (ISF) 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*. How to render ISF objects to speakers is described 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, potentially, floor speakers that are 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 come with pros and cons; it is up to content creators to pick the right tradeoff for them.

- In traditional **channel-based audio** mixing, sound elements are mixed and mapped to individual, fixed speaker channels, e.g. left, right, centre, left surround, and right surround. This paradigm is well known and works when the channel configuration at the decoding end can be predetermined, or assumed with reasonable certainty to be 2.0, 5.X, or 7.X. 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 audio** formats address this problem by providing audio in a form that can be rendered more easily to different speaker layouts. However, channels lose their traditional interpretation, and turn into components of a different base.
- 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.
- 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 consumer control over the broadcast experience. The present document specifies syntax and tools to support personalized audio experiences.

The ability to deliver multiple program elements 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.

While personalization is primarily used to deliver more choices, some choices are not driven by consumer preference but rather by playback device capabilities. Personalization can be based on the device that a consumer is using (called *target settings*): Each presentation can carry metadata for multiple target devices. Target-device metadata enables the content creator to define how the same audio elements will be combined on different devices.

EXAMPLE 2: For the "English" presentation the audio engineer may decide that a certain combination of dialogue and music and effects (M+E) is appropriate when replayed in 5.1, but that when the same "English" presentation is selected on a mobile device, louder dialogue and a lower M+E level are appropriate. Target-device rendering works in conjunction with DRC 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 will play back on different devices.

When the content creator wishes to give consumers wider freedom in the level of personalization, it is possible to enable a range of controls to be presented to consumers, allowing flexibility in creating preferred experiences.

NOTE 1: This presumes a suitable decoder UI or second-screen device that offers the consumer individual control of various audio elements.

Finally, AC-4 provides metadata fields allowing for the classification of objects, such as dialogue, making this available for a decoder to use as part of the personalization. This expands on the functionality offered by Dialogue Enhancement by enabling a wider range of control to adjust the balance between dialogue and other elements of the selected presentation.

NOTE 2: While the present document describes the metadata and controls available for creating personalization, it does not specify requirements. Specifying decoder behavior 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 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 (table of contents) lists the presentations that are carried in the stream.
The TOC contains a list of one or more presentations.

Presentation

Presentations are described by `presentation_info_v1` structures, if `bitstream_version` ≥ 1, and by `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 & Effects, Dialogue, Associated Audio, etc. 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 `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.

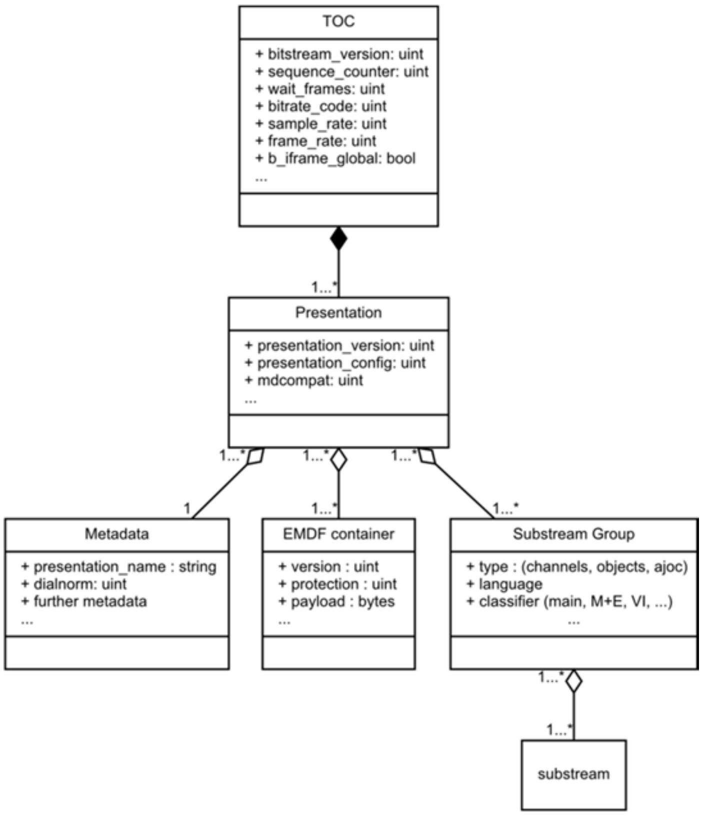

Figure 1: High level bitstream structure

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

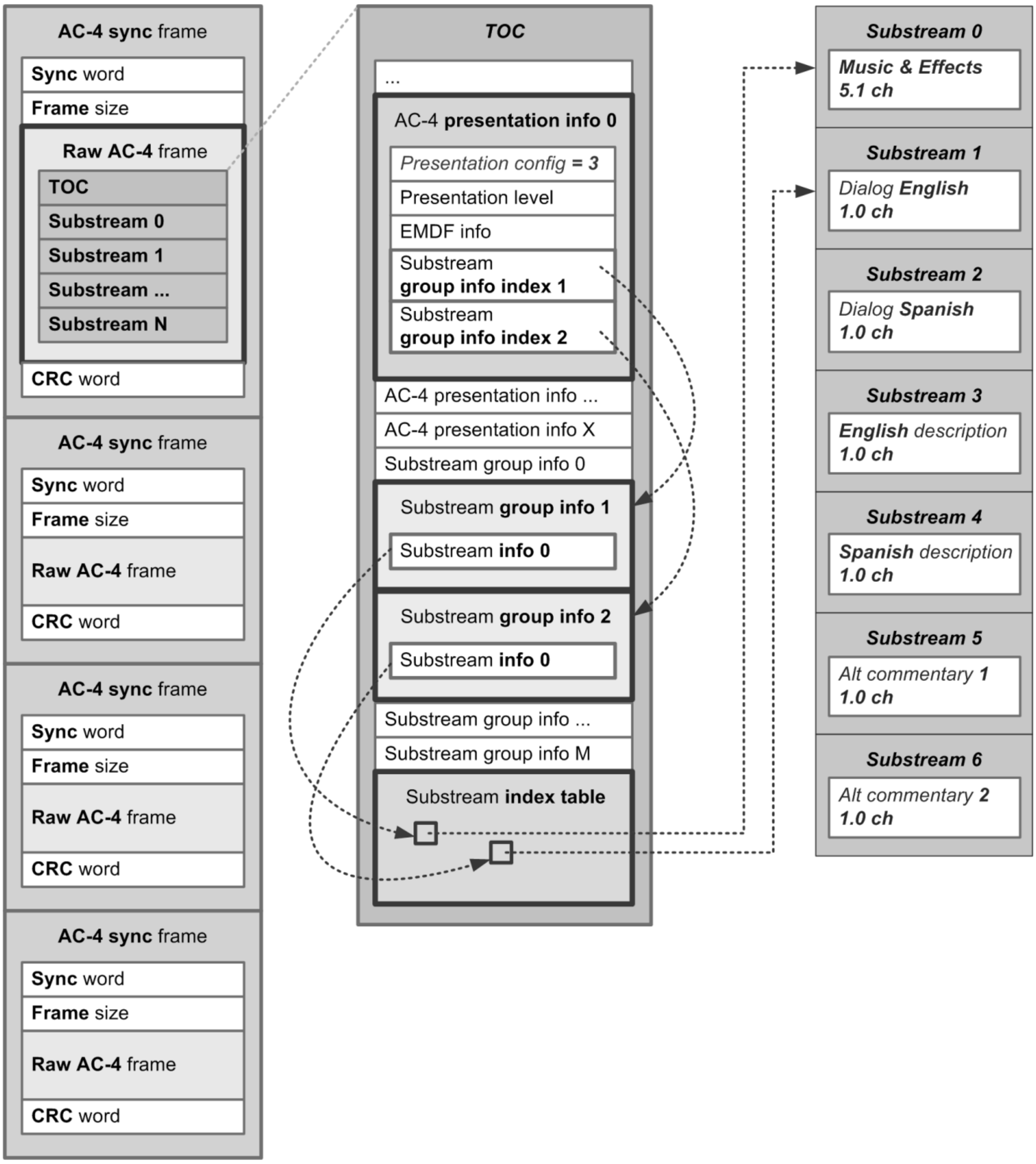

Figure 2: Example of complex frame with several presentations and substream groups

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` = 1.

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

b_audio_ndot
is valid for an `ac4_substream`

b_pres_ndot
is valid for the `ac4_presentation_substream`

b_oamd_ndot
is valid for the `oamd_substream`

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

4.6 Decoder compatibilities

Decoders conforming to the present document shall be capable of decoding bitstreams where $0 \leq$ bitstream_version ≤ 2. Decoders conforming to the present document shall be capable of decoding all presentations that conform on the one hand to the decoder compatibility level and, on the other hand, conform to either ETSI TS 103 190-1 [1] (presentation_version = 0) or to the present document (presentation_version = 1). Specification of the decoder compatibility level can be found in clause 6.3.2.2.3.

Table 4 shows the combinations of bitstream version, presentation version and presentation_config the decoder shall be able to decode. Table 5 shows the combinations of bitstream version, presentation version and substream version the decoder shall be able to decode. If the bitstream version is 1 and the presentation version is 1, the decoder shall decode the ac4_presentation_info element, and if the contained presentation_config=7, the decoder shall decode the ac4_presentation_v1_info element contained in ac4_presentation_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	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. presentation_config elements marked with N/A are not available, because the containing bitstream element 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	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 the full immersiveness in audio experience and the highest resolution of spatial information. Using 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. Using 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 may be reduced.

4.8 Decoding process

4.8.1 Overview

This clause describes how the decoder, specified in this documentation, 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 DRC and loudness processing

Figure 3 shows these steps as a flowchart.

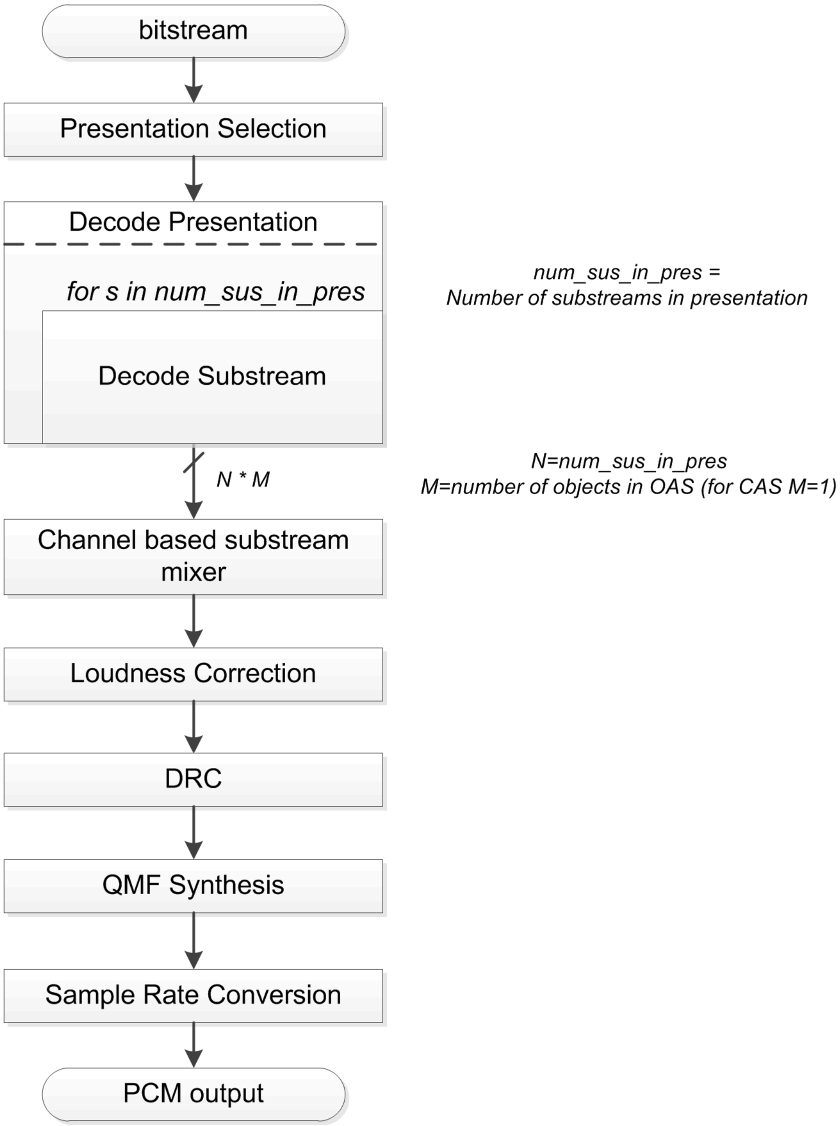

Figure 3: Flow diagram of the decoding process (AC-4)

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.

   a) 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.

   b) 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` = 1), the `presentation_name` provides a label that may be displayed in an appropriate UI.

NOTE 1: 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 a mdcompat 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 being disabled (b_enable_presentation = 0) shall not be selected.

NOTE 2: Presentations can come and go any time in the stream.

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 = 1, 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 further presentations 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 = 1) 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 = 1)
- Object Audio Substream (OAS) (b_channel_coded = 0)

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 Spectral Frontend Processing (ASF/SSF).

3) Apply Stereo and Multichannel Processing (SMP).

4) Depending on the content apply one of the following decoding tools:

   S-CPL
   applicable for core decoding and full decoding mode

   A-CPL
   applicable for full decoding mode (for core decoding mode gain factors shall be applied instead)

   A-JCC
   applicable for core decoding and full decoding mode

   A-JOC
   applicable for full decoding mode

5) Apply Dialogue Enhancement.

6) For CAS only: Apply DRC 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 4. 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 renderer, which produces N audio sample blocks associated to channels.

4.8.3.2 Identification of substream type

Substreams can be assigned to the categories: Main, Music & Effects, Associate or Dialogue. The Main and the Music & Effects substreams require the same processing, hence they are denoted as MME substreams from here onwards.

Substream types can be identified using the following information:

- If `presentation_version` = 0:
  - the substream type is "Associate" if either or both of these are true:
    - the substream is described by the last `ac4_substream_info` of the selected `presentation_info` for `presentation_config` ∈ [2, 3, 4]
    - the `ac4_substream_info` holds a `content_classifier` ∈ [0b010, 0b011, 0b101]
  - the substream type is "Dialogue" if either or both of these are true:
    - the substream is described by the second `ac4_substream_info` of the selected `presentation_info` for `presentation_config` ∈ [0, 3]
    - the `ac4_substream_info` holds a `content_classifier` = 0b100
- If `presentation_version` = 1:
  - the substream type is "Associate" if the substream is part of a substream group which is described by either or both of:
    - the last `ac4_sgi_specifier` of the selected `presentation_info` for `presentation_config` ∈ [2, 3, 4]
    - a `substream_group_info` that holds a `content_classifier` ∈ [0b010, 0b011, 0b101]
  - the substream type is "Dialogue" if the substream is part of a substream group which is described by either or both of:
    - the second `ac4_sgi_specifier` of the selected `presentation_info` for `presentation_config` ∈ [0, 3]
    - a `substream_group_info` that holds a `content_classifier` = 0b100

If none of these conditions are true, the substream type is "MME".

4.8.3.3 Substream decoding overview

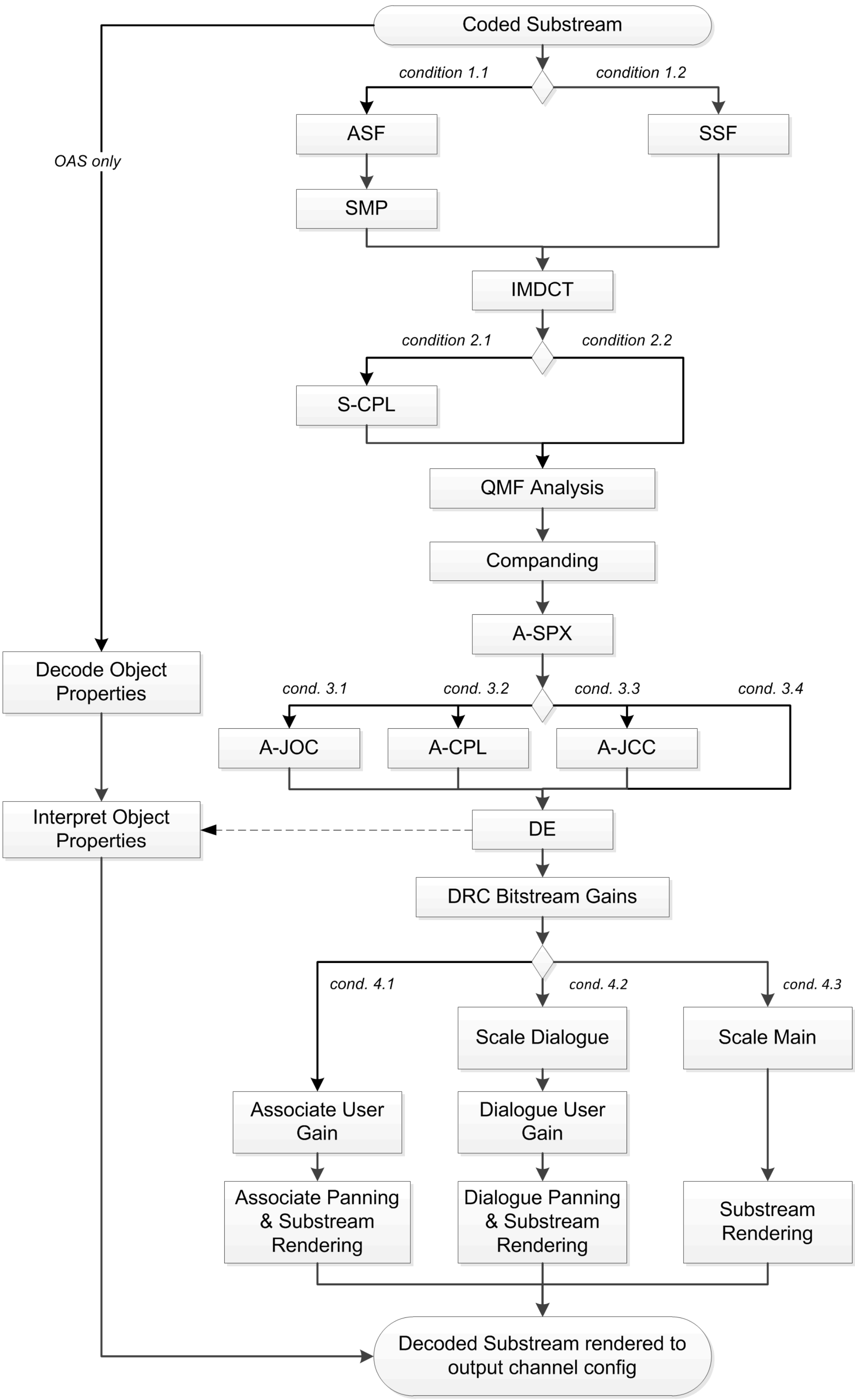

Figure 4: Decoding of a Substream

The conditions shown in Figure 4 are specified in Table 6.

Table 6: Conditions in flow chart 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 Audio Spectral Frontend (ASF)
1.2	if (spec_frontend == 1)	Tracks associated with the type SSF (spec_frontend=1) shall be processed by the Speech Spectral Frontend (SSF)
2.1	if ((channel element == immersive_channel_element) and (immersive_codec_mode in [SCPL, ASPX_SCPL]))	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])	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 is an *"Associate substream"*	Refer to clause 4.8.3.2
4.2	if the substream is a *"Dialogue substream"*	Refer to clause 4.8.3.2
4.3	if the substream is an *"MME substream"*	Refer to clause 4.8.3.2
NOTE: In this table, "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]`. Similarly, "codec mode" describes the 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 OAMD portions of the bitstream shall comprise the following steps:

1) Retrieve and decode the object audio metadata 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 alternative presentations, use the alternative metadata as indicated in the decoded presentation.

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

4.8.3.4.2 Object Audio Metadata location

Object properties are coded in multiple OAMD portions inside the bitstream, which are present on different bitstream layers. This saves bandwidth, because some of the transmitted properties are common to all objects, some are common to a group of objects, and some describe properties for individual objects. The location of specific OAMD information in the bitstream depends on whether alternative presentations are present (as indicated by `b_alternative`) and whether an object audio substream is coded with A-JOC or not (as indicated by `b_ajoc`). Table 7 shows the location of the OAMD portions for `b_alternative` = 0 and for possible values of `b_ajoc`. Table 8 shows the location of the OAMD portions for `b_alternative` = 1 and for possible values of `b_ajoc`.

If the object audio substream is coded with A-JOC and `b_static_dmx` is 0, two individual OAMD portions exist inside the corresponding substream. One portion comprises one `oamd_dyndata_single` element and an optional `oamd_timing_data` element, as indicated by a presence flag `b_dmx_timing` or `b_umx_timing` respectively. 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.

Table 7: OAMD locations in the bitstream for b_alternative = 0

OAMD portion	b_ajoc = 0	b_ajoc = 1
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	`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)

Table 8: OAMD locations in the bitstream for b_alternative = 1

OAMD portion	b_ajoc = 0	b_ajoc = 1
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	`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)	`OAMD_dyndata_single` (clause 6.3.9.4), contained in `audio_data_ajoc` (clause 6.2.3.4)

If `b_alternative` is 1, alternative object properties can be present in the `oamd_dyndata_single` element.

4.8.3.5 Spectral Frontend(s)

The Spectral Frontend (SF) is the first tool that shall be utilized to decode a present substream. The SF 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.

`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 respectively. 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 Frontend (ASF)
The 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 Frontend (SSF)
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 SF tool to decode all present `sf_data` elements taking account of 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 tool (SMP) shall be utilized to process the output tracks of the Audio Spectral Frontend (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 IMDCT stage, i.e. it is not passed into the Stereo and Multichannel Processing 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 Stereo and Multichannel Processing tool.

4.8.3.7 Inverse Modified Discrete Cosine Transformation (IMDCT)

After SSF processing or ASF processing plus 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 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 $0 \le ts <$ *`num_qmf_timeslots`* and $0 \le sb <$ *`num_qmf_subbands`*.

4.8.3.10 Companding

The companding tool is specified in ETSI TS 103 190-1 [1], clause 5.7.5.

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.

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 analogue to the description 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`.

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 according channel elements.

4.8.3.11 A-SPX

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 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 OAS, because objects are coded in some of the listed channel elements.

Table 9 specifies 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 9: Channels processed by A-SPX tool

Channel element	Decoding mode	Codec mode	*b_5fronts*	Channels in A-SPX
ice	Full	ASPX_SCPL	0	(L, R), C, (Ls, Lb), (Rs, Rb), (Tfl, Tbl), (Tfr, Tbr)
ice	Full	ASPX_SCPL	1	(L, Lscr), C, (R, Rscr), (Ls, Lb), (Rs, Rb), (Tfl, Tbl), (Tfr, Tbr)
ice	Core	ASPX_SCPL	X	(L, R), C, (Ls, Lb), (Tfl, Tbl)
ice	Full	ASPX_ACPL1	0	(A", B"), C", (D" ,F"), (E", G"), (H", J"), (I", K")
ice	Full	ASPX_ACPL1	1	(A", L"), C", (B", M"), (D", F"), (E", G"), (H", J"), (I", K")
ice	Core	ASPX_ACPL1	X	(A", B"), C", (D" ,F"), (E", G")
ice	Full/Core	ASPX_ACPL2	X	(A", B"), C", (D", F"), (E", G")
ice	Full/Core	ASPX_AJCC	X	(A", B"), C", (D", F")
22	Only Full decoding supported	ASPX_SIMPLE	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 either the *immersive_codec_mode* or the `22_2_codec_mode`.				

In core decoding mode and for *immersive_codec_mode* = ASPX_SCPL, the decoder shall utilize the A-SPX post-processing tool specified in clause 5.4 for the channels listed in Table 10.

Table 10: Channels to be processed by the A-SPX post-processing tool

b_5fronts	channels
0	Ls, Rs, Lt, Rt
1	L, R, Ls, Rs, Lt, Rt

In core decoding mode and for *immersive_codec_mode* = ASPX_SCPL, 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 $0 \le ts < num\_qmf\_timeslots$ and $0 \le sb < num\_qmf\_subbands$. If two input channels are processed in this operation, the decoder shall apply the same game factor g to each output QMF subsample $Qout_{\mathrm{ASPX},b}(ts, sb)$ of output channel b respectively.

In full decoding mode and for *immersive_codec_mode* = ASPX_SCPL and *b_5fronts* = 0, 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 $0 \le ts < num\_qmf\_timeslots$ and $0 \le sb < num\_qmf\_subbands$. If two input channels are processed in this operation, the decoder shall apply the same game 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* = 0.

Table 11: Channel-dependent gain for full decoding in immersive_codec_mode = ASPX_SCPL and b_5fronts = 0

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

In full decoding mode and for *immersive_codec_mode* = ASPX_SCPL and *b_5fronts* = 1, 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 $0 \le ts < num\_qmf\_timeslots$ and $0 \le sb < num\_qmf\_subbands$. If two input channels are processed in this operation, the decoder shall apply the same game factor g to each output QMF subsample $Qout_{\mathrm{ASPX},b}(ts, sb)$ of output channel b respectively. Table 12 shows the gain factor g for the processing of different input channels for *b_5fronts* = 1.

Table 12: 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 respectively, 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 1), 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. Which channels the decoder shall process is shown in Table 13. 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 13: Channels processed by A-CPL tool for immersive_channel_element

immersive_codec_mode	*b_5fronts*	Channels to be processed by A-CPL
ASPX_ACPL1, ASPX_ACPL2	0	C, L, R, (Ls, Lb), (Rs, Rb), (Tfl, Tbl), (Tfr, Tbr)
ASPX_ACPL1, ASPX_ACPL2	1	C, (L, Lscr), (R, Rscr), (Ls, Lb), (Rs, Rb), (Tfl, Tbl), (Tfr, Tbr)
NOTE: Channels grouped by parentheses shall be processed together.		

If the AC-4 decoder operates in core decoding mode, 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 operation for the application of Dialogue Enhancement is performed by different processes depending on the decoding mode and the substream type - CAS or OAS.

For CAS processing the decoder shall utilize the DE processing tool depending on the input channel configuration and the coding mode as specified in Table 14.

For OAS processing the decoder shall utilize the DE processing tool depending on `b_ajoc` as specified in Table 15.

Table 14: DE tools for CAS processing

Input channel configuration	Codec mode	DE tool for core decoding	DE 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_ACPL1	ETSI TS 103 190-1 [1], clause 5.7.8	ETSI TS 103 190-1 [1], clause 5.7.8
7.X.4	ASPX_ACPL2, 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_ACPL2	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 15: DE tools for OAS processing

`b_ajoc`	DE tool for core decoding	DE tool for full decoding
1	Clause 5.8.2.4	Clause 5.8.2.3
0	Clause 5.8.2.5	Clause 5.8.2.5

If `b_de_simulcast` is 1, the decoder shall use the second `de_data` in `dialog_enhancement` for the core decoding mode.

Table 16 specifies which channels shall be processed by the DE tool specified in ETSI TS 103 190-1 [1], clause 5.7.8.

Table 16: Channels processed by DE tool

Input channel configuration	DE 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 DRC bitstream gain application

When decoding a channel audio substream with coded DRC 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 respectively. The decoder shall utilize the channel groups specified in Table 91 to perform the gain-value decoding for the immersive and the 22.2 channel configurations analogue 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 which 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 set to 1 in the `ac4_presentation_substream` associated with this presentation, the gains related to associated audio decoding should be extracted from there.

If the selected presentation has a `presentation_version` = 0 and `b_associated` is set to 1 in the `extended_metadata` of the associated substream, the gains related to associated audio decoding should be extracted from there.

Associated user gain

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

Gain application

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

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

Scale MME substream

The decoding allows for a gain adjustment of the channels in the MME substream if an adjustment needs to be done. If no gain values are given, a default of 0 dB shall be used.

First, the centre channel of the MME substream, if available, should be gain adjusted using `scale_main_centre`. Next, the two front channels L and R of the MME substream should be gain adjusted using `scale_main_front`. And finally, all channels of the MME substream should be gain adjusted using `scale_main`.

Scale Dialogue substream

If the selected presentation includes one or more dialogue substreams alongside an associated audio substream, all channels of the dialogue substreams should be gain adjusted using `scale_main_centre`, `scale_main_front` and `scale_main`, analogue to the MME substream. If no gain values are given, a default 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 set to 1 in the `ac4_presentation_substream` associated with this presentation, the gains related to decoding of dialogue substreams should be extracted from there.

If the selected presentation has a `presentation_version` = 0 and `b_dialog` is set to 1 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 $g\_dialog \in [-\infty, g\_dialog\_max]$ 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 set to 1, 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:

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

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 - CAS and OAS.

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 0, 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. If `b_isf` is 1, 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 OAR 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 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 92, shall be applied to the audio signal $X_{s,sg}$ of each substream s which is part of a substream group sg according to:

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

Mixing of substreams

The actual mixing is done for each channel ch by adding up all channels of all substreams X_s, according to:

$$Y_{\mathrm{ch}} = \sum_{s=0}^{\mathrm{n\_sub}} X_{s,\mathrm{ch}}$$

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:

- downmixing to a lower channel configuration
- decoding of an alternative presentation
- realtime loudness correction data (derived from realtime 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 `basic_metadata` of the associated substream for presentations containing associated audio, and
- from the substream indicated in Table 17 for main audio decoding.

Table 17: Substream containing valid dialnorm information

presentation_config	Substream
0	Dialogue
1	Main
2	Main
3	Dialogue
4	Main
5	Main

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* shall be used:

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

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

Here, $s_{\mathrm{in},ch}$ and $s_{\mathrm{loud\_corr},ch}$ refer to the input and output samples of each channel ch, respectively.

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` = 1, a target-specific loudness correction shall be applied:

$$\boldsymbol{s}_{\mathrm{target\_corr,ch}}(\mathrm{ts,sb}) = 2^{\mathrm{target\_corr\_gain}/6} \times \boldsymbol{s}_{\mathrm{in,ch}}(\mathrm{ts,sb})$$

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

Here, $\boldsymbol{s}_{\mathrm{in},ch}$ and $\boldsymbol{s}_{\mathrm{target\_corr},ch}$ refer to the input and output samples of each channel ch, respectively. *target_corr_gain* is the target-specific loudness correction factor specified for the target-device category (see Table 89) 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 18.

Table 18: 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 Realtime loudness correction data

When realtime loudness correction data `rtll_comp_gain` is present in the bitstream, this loudness correction factor shall be applied as:

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

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

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

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 91.

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 for this presentation as specified in clause 4.8.5.2.

The DRC tool requires two inputs per channel ch: The audio signal which is the output of 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 signal driving the side chain should be identical to the input to Dialogue Enhancement (i.e. it does not include dialogue enhancements) as specified in clause 4.8.3.15. Alternatively, the side chain 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 Muxing Tool

The Elementary Stream Muxing (*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 bitstreams 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_info()` indicates the `presentation_config` (e.g. `presentation_config` = 0, "Music and Effects (M+E) + Dialogue").

- If the `b_multi_pid` bit in the `ac4_presentation_info()` is set to 0, the presentation is fully contained within a single AC-4 bitstream, and each substream belonging to the presentation can be referenced from the `substream_index_table` as contained in the TOC.
- If the `b_multi_pid` bit is set to 1, this indicates that the presentation is split over more than one AC-4 bitstream, and not all substreams required by the presentation are self-contained within a single AC-4 bitstream. 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 TOCs of all available AC-4 streams for matching `presentation_config` 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 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 `presentation_config` and `ac4_substream_group_info` elements, as well as the substream payloads, into a new multiplex, building a self-contained elementary stream.

EXAMPLE: Figure 5 shows a simple example with two input AC-4 bitstreams, ES1 and ES2, both containing the same presentation information. Two presentations exist offering the possibility of *Main and Effects* (*M+E*) with English dialogue, or *Main and Effects* with French dialogue. The ES1 AC-4 bitstream contains both the *M+E* and English dialogue substreams, and the `substream_index_table` indicates the location of these in the bitstream. However, it does not indicate a 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 *M+E* with English or French dialogue, and updates `ac4_substream_group_info` and `substream_index_table` to reflect this.

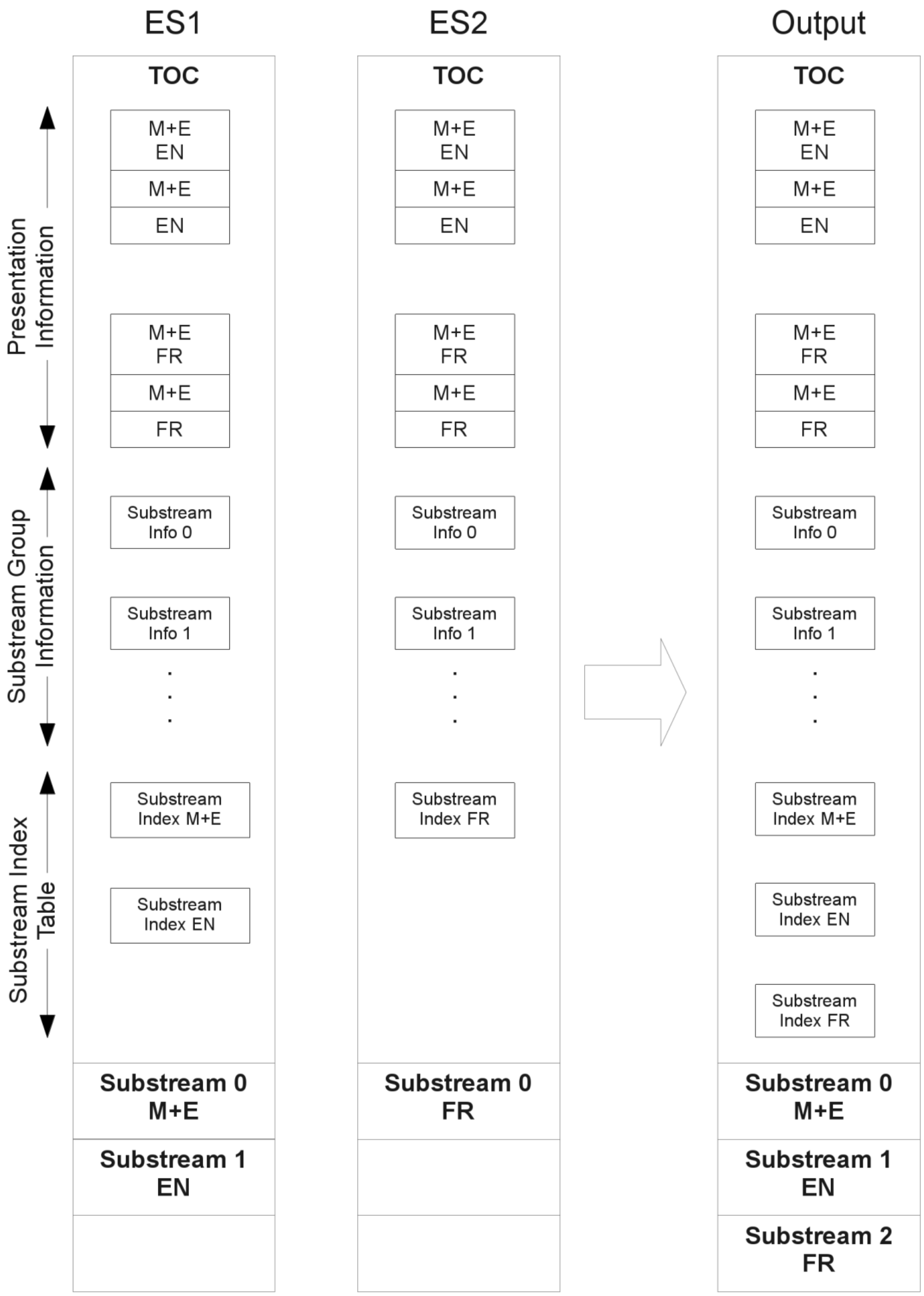

Figure 5: M+E with English and French dialogue, distributed over two elementary streams

5.1.3 Efficient High Frame Rate mode

AC-4 is capable of aligning the frame rates with video signals of up to 120 fps. 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 `raw_ac4_frames` at the nominal frame rate. The decoder assembles a decodable audio payload from a group of *`frame_rate_fraction`* = 2 or 4 frames of the elementary stream at the nominal frame rate. Assembling a decodable audio payload starts at a frame where $\text{sequence\_counter} \bmod \text{frame\_rate\_fraction} \equiv 0$ (called "the first frame"). Assembling ends at a frame where $(\text{sequence\_counter} + 1) \bmod \text{frame\_rate\_fraction} \equiv 0$ (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 6 for an example.

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

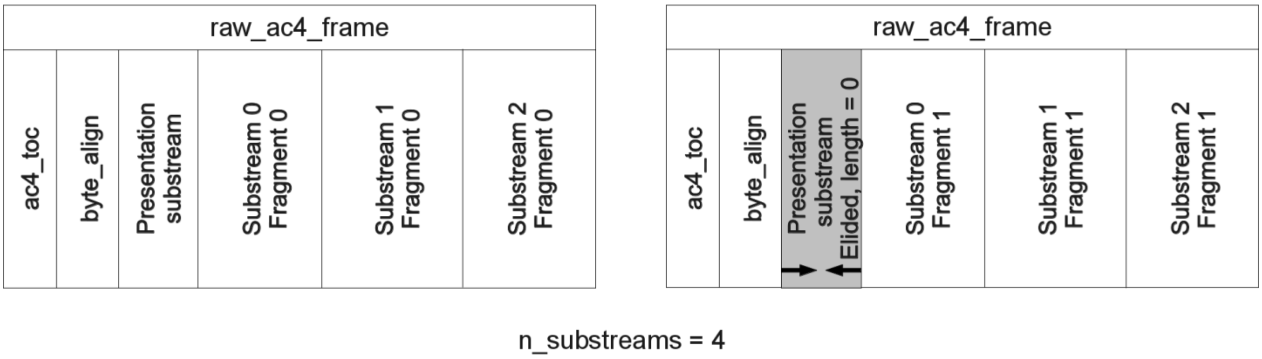

Figure 6: Example of fragmented payload

This feature 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
- the *`frame_rate_fraction`* as transmitted in `frame_rate_fractions_info` is two or four

The resulting audio frame rates are shown in Table 19.

Table 19: Determining the codec internal audio frame rate

stream frame_rate_index	stream fps	frame_rate_fraction	audio_frame_rate_index	audio 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 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 sequence_countermodframe_rate_fraction $\equiv 0$.

EXAMPLE: A possible implementation of the FIFO is shown in Figure 7.

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.

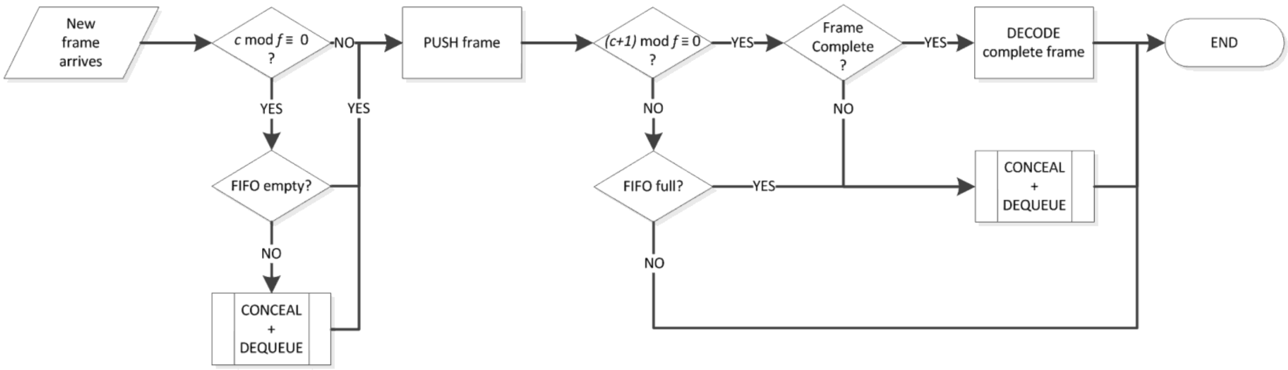

Figure 7: Framing algorithm

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 8.

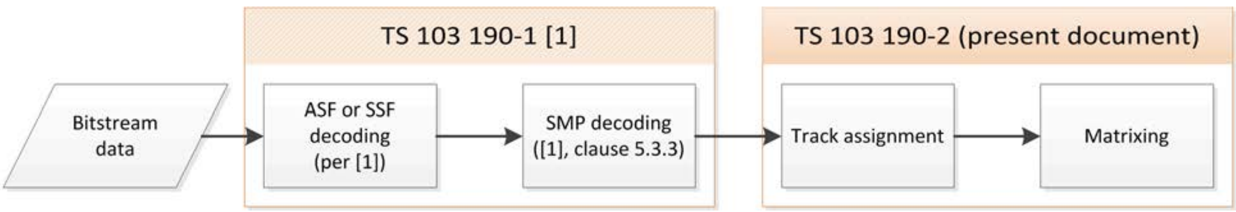

Figure 8: General flow of processing in the extended SMP tool

5.2.2 Interface

5.2.2.1 Inputs

The input to the stereo and multichannel processing 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:

$\mathbf{s}_{SMP,[0|1|...]}$

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 stereo and multichannel processing tool are n_{SAP} blocks of scaled spectral lines:

$\mathbf{s}_{SMP,[A''|B''|C''|...]}$

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 through Table A.41. 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), similar 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` = 0, 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 20.

Table 20: Track assignment

core_5ch_grouping	0			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`=2. 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`=1, 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 $0 \le j <$ *n_elem*
- otherwise the parameters a'_j shall be set to zero.

6) Produce the outputs of the Stereo and Multichannel Tool as specified in Table 21.

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 21: 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 = 0, silence tracks *H'* through *K'*.
   - If b_5fronts = 1, 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* = 0.

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

- If *b_5fronts* = 0, silence tracks *F'* through *K'*.
- If *b_5fronts* = 1, 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.42. 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. In a second step, the tracks shall be assigned to output channels as shown in Table 22.

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

element	Input	Output
1st mono_data	[0]	[LFE]
2nd mono_data	[1]	[LFE2]
1st two_channel_data	[2,3]	[L,R]
2nd two_channel_data	[4,5]	[C,Tc]
3rd two_channel_data	[6,7]	[Ls,Rs]
4th two_channel_data	[8,9]	[Lb, Rb]
5th two_channel_data	[10,11]	[Tfl, Tfr]
6th two_channel_data	[12,13]	[Tbl,Tbr]
7th two_channel_data	[14,15]	[Tsl,Tsr]
8th two_channel_data	[16,17]	[Tfc,Tbc]
9th two_channel_data	[18,19]	[Bfl,Bfr]
10th two_channel_data	[20,21]	[Bfc,Cb]
11th two_channel_data	[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* ∈ {SCPL, ASPX_SCPL}.

5.3.2 Interface

5.3.2.1 Inputs

$\mathbf{Xin}_{\mathbf{SCPL,[A|B|...]}}$

$n_{\mathrm{SCPL,in}}$ time domain signals, each being an IMDCT processed output signal of the SMP tool.

The $\mathbf{Xin}_{\mathrm{SCPL}}$ 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 23.

Table 23: Number of S-CPL input signals

b_5fronts	$n_{\mathrm{SCPL,in}}$
0	11
1	13

5.3.2.2 Outputs

Xout$_{\mathbf{SCPL,[a|b|...]}}$

$n_{\mathrm{SCPL,out}}$ 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 7 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 eleven or thirteen IMDCT processed output signals (A", B", C", ...) of the Stereo and Multichannel tool (see Table 21). 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 24, 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 24: S-CPL channel mapping for immersive_codec_mode ∈{SCPL, ASPX_SCPL} for full decoding

b_5fronts	Output channel mapping
0	$\begin{bmatrix} L \\ R \\ C \end{bmatrix} = \mathrm{c\_gain} \times \begin{bmatrix} \mathrm{A}'' \\ \mathrm{B}'' \\ \mathrm{C}'' \end{bmatrix}$ $\begin{bmatrix} \mathrm{Ls} \\ \mathrm{Lrs} \\ \mathrm{Rs} \\ \mathrm{Rrs} \\ \mathrm{Ltf} \\ \mathrm{Ltb} \\ \mathrm{Rtf} \\ \mathrm{Rtb} \end{bmatrix} = \mathrm{m\_gain} \times 2 \times \begin{bmatrix} ½ & ½ & 0 & 0 & 0 & 0 & 0 & 0 \\ ½ & -½ & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & ½ & ½ & 0 & 0 & 0 & 0 \\ 0 & 0 & ½ & -½ & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & ½ & ½ & 0 & 0 \\ 0 & 0 & 0 & 0 & ½ & -½ & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & ½ & ½ \\ 0 & 0 & 0 & 0 & 0 & 0 & ½ & -½ \end{bmatrix} \times \begin{bmatrix} \mathrm{D}'' \\ \mathrm{F}'' \\ \mathrm{E}'' \\ \mathrm{G}'' \\ \mathrm{H}'' \\ \mathrm{J}'' \\ \mathrm{I}'' \\ \mathrm{K}'' \end{bmatrix}$
1	$[C] = \mathrm{c\_gain} \times [\mathrm{C}'']$ $\begin{bmatrix} \mathrm{Lw} \\ \mathrm{Lscr} \\ \mathrm{Rw} \\ \mathrm{Rscr} \end{bmatrix} = 2 \times \begin{bmatrix} ½ & ½ & 0 & 0 \\ ½ & -½ & 0 & 0 \\ 0 & 0 & ½ & ½ \\ 0 & 0 & ½ & -½ \end{bmatrix} \times \begin{bmatrix} \mathrm{A}'' \\ \mathrm{L}'' \\ \mathrm{B}'' \\ \mathrm{M}'' \end{bmatrix}$ $\begin{bmatrix} \mathrm{Ls} \\ \mathrm{Lrs} \\ \mathrm{Rs} \\ \mathrm{Rrs} \\ \mathrm{Ltf} \\ \mathrm{Ltb} \\ \mathrm{Rtf} \\ \mathrm{Rtb} \end{bmatrix} = \mathrm{m\_gain} \times 2 \times \begin{bmatrix} ½ & ½ & 0 & 0 & 0 & 0 & 0 & 0 \\ ½ & -½ & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & ½ & ½ & 0 & 0 & 0 & 0 \\ 0 & 0 & ½ & -½ & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & ½ & ½ & 0 & 0 \\ 0 & 0 & 0 & 0 & ½ & -½ & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & ½ & ½ \\ 0 & 0 & 0 & 0 & 0 & 0 & ½ & -½ \end{bmatrix} \times \begin{bmatrix} \mathrm{D}'' \\ \mathrm{F}'' \\ \mathrm{E}'' \\ \mathrm{G}'' \\ \mathrm{H}'' \\ \mathrm{J}'' \\ \mathrm{I}'' \\ \mathrm{K}'' \end{bmatrix}$

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 $\mathbf{n}_{SCPL,out}$ processed IMDCT output signals (A", B", C", ...) of the Stereo and Multichannel tool (see Table 21) and assigning them to the output channels.

The output channels shall be created as specified in Table 25, 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 25: S-CPL channel mapping for immersive_codec_mode ∈{SCPL, ASPX_SCPL} for core decoding

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

5.4 Advanced Spectral Extension (A-SPX) post-processing 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 Inputs

$\mathbf{Qin}_{\mathbf{ASPX\_PP,[0|1|2|...]}}$

num_sig complex valued QMF matrices

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

If `b_5fronts` is 1, *num_sig* is 6, otherwise *num_sig* is 4.

5.4.3 Processing

The decoder shall process the input signals

$\mathbf{Qin}_{ASPX\_PP,[0|1|2|...]}$ = *Q_in_ASPX_PP[i]*, for i=0,1,2,...

to calculate the output signals

$\mathbf{Qout}_{ASPX\_PP,[0|1|2|...]}$ = *Q_out_ASPX_PP[i]*, for i=0,1,2,...

as specified in Table 26.

Table 26: Pseudocode

```
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]; // -1.5dB
    }
  }
}
```

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

This Advanced Coupling (A-CPL) tool specification extends the specification of the A-CPL tool in ETSI TS 103 190-1 [1], clause 5.7.7 to support A-CPL for immersive channel 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 7_CH_STATIC and either four or six parallel A-CPL modules are utilized. If *b_5front* = 1, thirteen input channels are present and are processed by six A-CPL modules. If *b_5front* = 0, eleven input channels are present and are processed by four A-CPL modules. The mapping of the channels to A-CPL input and output variables is specified in Table 27.

Table 27: Input/Output channel mapping for immersive_channel_element and b_5fronts

input/output	channel	b_5fronts
x0/z0	L	0
x1/z2	R	0
x2/z4	C	0
x3/z1	Lscr	1
x4/z3	Rscr	1
x5/z5	Ls	0
x6/z7	Rs	0
x7/z6	Lb	0
x8/z8	Rb	0
x9/z9	Tfl	0
x10/z11	Tfr	0
x11/z10	Tbl	0
x12/z12	Tbr	0

How the decoder shall calculate the output signals, is described in the pseudocode shown in Table 28.

Table 28: Pseudocode

```
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_5front) {
        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_5front) {
        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);
```

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, respectively.

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 29.

Table 29: ajcc_num_out

decoding mode	*b_5fronts*	*ajcc_num_out*
full decoding	0	11
	1	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 191. 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

To get the quantized values *ajcc_<SET>_q* from the Huffman decoded values *ajcc_<SET>*, where <SET> is an identifier for the A-JCC parameter set type and <SET> ∈ {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}, differential decoding as described in the pseudocode shown in Table 30 shall be done.

Table 30: Pseudocode

```
// 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], tables 197 and 198 if the quantization mode is set to fine (ajcc_qm_ab = 0), and using ETSI TS 103 190-1 [1], tables 199 and 200 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], tables 197 or 199 during the dequantization of the alpha values. This value is used in ETSI TS 103 190-1 [1], tables 198 or 200, 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 described by the pseudocode shown in Table 31.

Table 31: Pseudocode

```
// 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 described by the pseudocode shown in Table 32.

Table 32: Pseudocode

```
// 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 timeslot indicated by ajcc_param_timeslot.

The function *interpolate_ajcc()*, used in clause 5.6.3.5, is described by the pseudocode shown in Table 33.

Table 33: Pseudocode

```
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;
}
```

Function *sb_to_pb()* maps from QMF subbands to parameter bands according to ETSI TS 103 190-1 [1], table 191. 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 Table 34 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.

Table 34: Pseudocode

```
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 35.

Table 35: ajcc_num_decorr

decoding mode	b_5fronts	ajcc_num_decorr
full decoding	0	6
	1	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 like 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 decorralator instances are also processed by the transient ducker algorithm like 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
$z_0(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,L}}$
Right output channel
$z_1(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,R}}$
Centre output channel
$z_2(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,C}}$
Left screen output channel
$z_3(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,Lscr}}$
Right screen output channel
$z_4(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,Rscr}}$
Left surround output channel
$z_5(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,Ls}}$
Right surround output channel
$z_6(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,Rs}}$
Left back output channel
$z_7(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,Lb}}$
Right back output channel
$z_8(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,Rb}}$
Left top front output channel
$z_9(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,Ltf}}$
Right top front output channel
$z_{10}(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,Rtf}}$
Left top back output channel
$z_{11}(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,Ltb}}$
Right top back output channel
$z_{12}(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,Rtb}}$

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

The present outputs are derived according to the pseudocode in Table 36.

Table 36: Pseudocode

```
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 Table 37 describes the function *input_sig_pre_modification()*.

Table 37: Pseudocode

```
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);
}
```

ajcc_core_mode_prev is a helper variable which shall be initialized to `ajcc_core_mode`.

The pseudocode in Table 38 describes the function *ajcc_module_1()*.

Table 38: Pseudocode

```
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 Table 39 describes the function *ajcc_module_2()*.

Table 39: Pseudocode

```
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

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

Right output channel

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

Centre output channel

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

Left surround output channel

$z_3(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,Ls}}$

Right surround output channel

$z_4(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,Rs}}$

Left top output channel

$z_5(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,Lt}}$

Right top output channel

$z_6(ts,sb) \in \mathbf{Qout}_{\mathrm{AJCC,Rt}}$

The output signals shall be derived according to the pseudocode in Table 40.

Table 40: Pseudocode

```
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 Table 41 describes the function `ajcc_module_3()`.

Table 41: Pseudocode

```
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_we
t2f[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_we
t3b[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 Table 42 describes the function *ajcc_module_4()*.

Table 42: Pseudocode

```
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_wet
3[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]+ajc
c_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 multiple 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 Inputs

$\mathbf{Qin}_{\mathrm{AJOC},[a|b|\ldots]}$

m_{AJOC} complex valued QMF matrices for m_{AJOC} objects to be processed by the A-JOC tool

The $\mathbf{Qin}_{\mathrm{AJOC}}$ 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.

5.7.2.2 Outputs

$\mathbf{Qout}_{\mathrm{AJOC},[a|b|\ldots]}$

n_{AJOC} complex valued QMF matrices corresponding to the number of reconstructed objects

The $\mathbf{Qout}_{\mathrm{AJOC}}$ matrices each consist of *num_qmf_timeslots* columns and *num_qmf_subbands* rows. n_{AJOC} is the number for *num_umx_signals* reconstructed objects.

5.7.2.3 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 parameters inside the Advanced Joint Object Coding sideinfo are transmitted per parameter band. 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 102.

The mapping of grouping of QMF subbands to parameter bands, corresponding to the specified number of parameter bands, is shown in Table 43.

Table 43: 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 Table 44 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 $0 \le o < n_{AJOC}$ and $0 \le ch < m_{AJOC}$.

Table 44: Pseudocode

```
// 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] == 1) {
            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] == 1) {
            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 45 and Table 47 if the quantization mode is set to coarse (ajoc_quant_select=1), and using Table 46 and Table 48 if the quantization mode is set to fine (ajoc_quant_select=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 45: 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
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 46: 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
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

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

Table 47: 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 48: 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 (OAMD). 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. The pseudocode in Table 49 describes the function *`ajoc_interpolate()`*, which is used in clause 5.7.3.6.1.

Table 49: Pseudocode

```
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:

Qdecorr_out$_{\mathbf{AJOC,[a|b|...]}}$
: 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:

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

The **Qdecorr_in**$_{\text{AJOC}}$ and **Qdecorr_out**$_{\text{AJOC}}$ 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 like 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, 1, 2, 0, 1, 2, 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

$x_{\text{ch}}(ts,sb) \in \mathbf{Qin}_{\text{AJOC,ch}}$ for ch = 0,...,*num_dmx_signals* - 1

The *num_umx_signals* output objects are addressed as:

Output signals

$z_{\text{o}}(ts,sb) \in \mathbf{Qout}_{\text{AJOC,o}}$ for o = 0,...,*num_umx_signals* - 1

The reconstruction of the output signals requires additional internal signals:

Decorrelator input signals

$u_{de}(ts,sb) \in$ **Qdecorr_in**$_{AJOC,de}$ for de = 0,...,*ajoc_num_decorr* - 1

Decorrelator output signals

$y_{de}(ts,sb) \in$ **Qdecorr_out**$_{AJOC,de}$ 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*.

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

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 **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 **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 sideinfo 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 Table 50 describes the reconstruction process.

Table 50: Pseudocode

```
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++) {
       if (ajoc_object_present[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 which 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 Table 50 to create the decorrelator input signals.

sb_to_pb() is a helper function which maps from QMF subbands to A-JOC parameter bands according to Table 43.

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 which 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 which 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 which stores the incremental delta value to be added by the interpolation process with the dimension *delta_inc_wet[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 which 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

$u_{\mathrm{de}}(ts,sb) \in \mathbf{Qdecorr\_in}_{\mathrm{AJOC,de}}$ 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:

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

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

$$\mathbf{D}(\mathrm{ts},\mathrm{sb}) = |\mathrm{Csub2}(\mathrm{ts},\mathrm{sb})^T| \times \mathrm{Csub1}(\mathrm{ts},\mathrm{sb})$$

5.8 Dialogue Enhancement (DE) for immersive audio

5.8.1 Introduction

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

5.8.2 Processing

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

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

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` = 1. This clause specifies the DE processing for core decoding of A-JCC coded 9.X.4 content where the DE operation is performed right after the A-JCC core decoding specified in clause 5.6.3.5.3. The input signals for the DE 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:

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

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:

$$m = \begin{pmatrix} \mathrm{Qin}_{\mathrm{AJCC,A}} \\ \mathrm{Qin}_{\mathrm{AJCC,B}} \\ \mathrm{Qin}_{\mathrm{AJCC,C}} \end{pmatrix}$$

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

$$u = \begin{pmatrix} \mathrm{Qout}_{\mathrm{AJCC,L}} \\ \mathrm{Qout}_{\mathrm{AJCC,R}} \\ \mathrm{Qout}_{\mathrm{AJCC,C}} \end{pmatrix}$$

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

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

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

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

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

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 $M_{\text{interp}}(ts, sb)$ is specified in Table 52. The input parameters `int_type`, `num_ps` and `psts` shall be initialized to:

- *int_type* = (`ajcc_it_lf`, `ajcc_it_rf`)
- *num_ps* = (`ajcc_nps_lf`, `ajcc_nps_rf`)
- *psts* = (`ajcc_psts_lf`, `ajcc_psts_rf`)

The calculation of the coefficients C_{L} = `C_L` and C_{R} = `C_R` is specified in Table 51.

Table 51: Pseudocode

```
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];
  }
}
```

Table 52: Pseudocode

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

Function `sb_to_pb()` is a helper function which maps from QMF subbands to A-JCC parameter bands according to clause 5.6.3.1. `sb_to_pb_de()` is a helper function which maps from QMF subbands to DE parameter bands according to ETSI TS 103 190-1 [1], clause 5.7.8.4.

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

NOTE: This tool is an extension to the DE 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 DE 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:

$$y(ts, sb) = H_{\text{DE,ACPL,Core}}(ts, sb) \times m(ts, sb)$$

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:

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

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

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

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

The matrix $M_{\text{interp}}(ts, sb)$ shall be calculated from two A-CPL coefficients C_L and C_R and a modified DE matrix

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

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 $M_{\text{interp}}(ts, sb)$ is specified in Table 52. 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 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_L = C_L and C_R = C_R is specified in Table 53.

Table 53: Pseudocode

```
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]);
  }
}
```

Functions *acpl_alpha5_dq* shall be derived from the acpl_alpha1 value in the fifth acpl_data_1ch element and *acpl_alpha6_dq* 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 DE for full decoding of A-JOC coded content

NOTE: This tool is an extension to the DE 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 DE is enabled for full decoding of A-JOC coded content the DE processing shall be done by a modification of the A-JOC parameters. The function *ajoc_de_process()*, which is specified in Table 54, shall be called during the A-JOC processing as specified in Table 50. The used *de_gain* value for full decoding is specified as:

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

where $G_{\max}$ shall be derived from de_max_gain.

Table 54: Pseudocode

```
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 function *dlg_obj()* is specified in Table 107.

5.8.2.4 DE for core decoding of A-JOC coded content

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

When decoding A-JOC content in core decoding mode DE 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 $Qin_{AJOC,[a|b|...]}$ 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 Table 50. The function *ajoc_de_process()* is specified in Table 54, where *de_gain* for core decoding is specified as:

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

where G_{max} shall be derived from de_max_gain. $H_A(ts, sb)$ shall be calculated from *mtx_dry* as specified in Table 55.

Table 55: Pseudocode

```
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:

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

where the calculation of H'_M shall be done as specified in Table 56. The variable *de_dlg_dmx_coeff* shall be derived from de_dlg_dmx_coeff_idx according to Table 108.

Table 56: Pseudocode

```
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 helper function *dlg_obj()* is specified in Table 107.

5.8.2.5 DE for non A-JOC coded object audio content

NOTE: This tool is an extension to the DE 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 DE is enabled for decoding of a non A-JOC coded object audio content and the processed substream is a dialogue substream, the decoder shall apply a gain factor:

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

to the corresponding audio objects for which b_dialog is 1. Hence, the decoder should apply the gain factor to the corresponding object essences.

If b_dialog_max_gain is 1, G_{max} shall be calculated from dialog_max_gain as $G_{\mathrm{max}} = 3 \times (1+\mathrm{dialog\_max\_gain})[\mathrm{dB}]$, otherwise G_{max} shall be 0 dB.

5.9 Object Audio Metadata Timing

5.9.1 Introduction

The AC-4 bitstream supports multiple updates for 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 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 $update\_sample_n$=sample_offset+(32*$block\_offset\_factor_n$).

Figure 9 shows the update PCM samples for received block updates and how block updates are stored in the AC-4 bitstream.

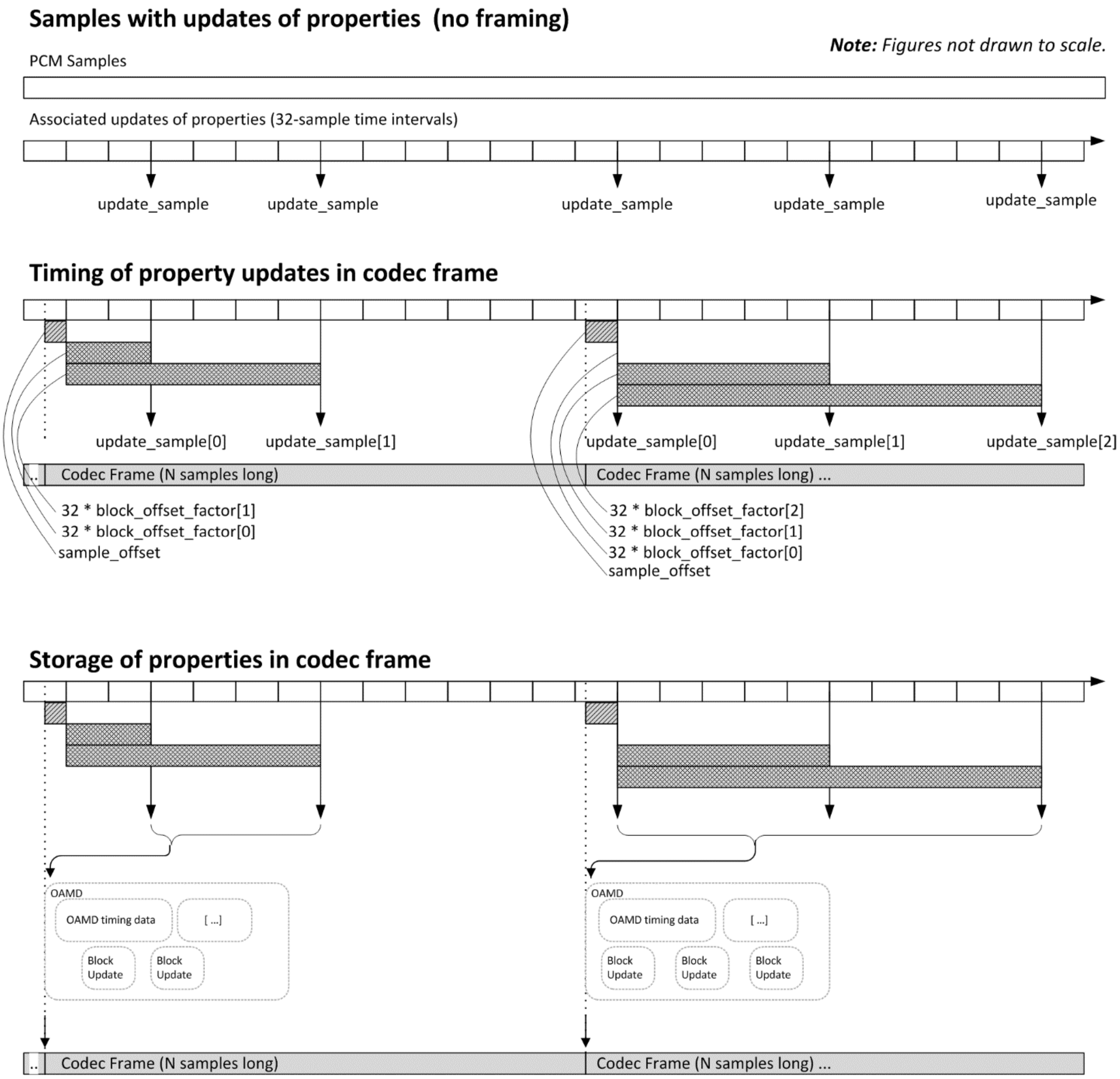

Figure 9: OAMD block updates

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:

- a channel based renderer, downmixing or up-mapping input channels to output channels
- an ISF renderer, rendering 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 10, applicable to the processing of each channel audio substream of a presentation.

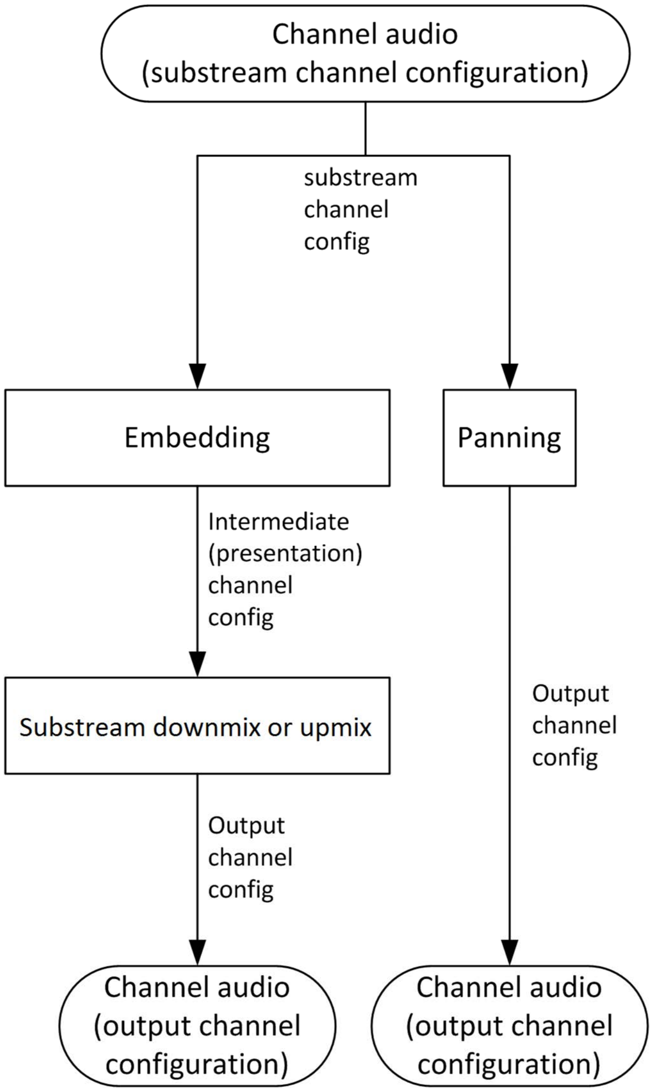

Figure 10: Channel rendering process

The channel rendering process shall perform the following steps:

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

2) All other present signals shall be embedded into a channel configuration which reflects the presentation channel mode (as indicated by `presentation_channel_mode` for full decoding mode or `pres_ch_mode_core` for core decoding mode) by adding silent signals.

3) The resulting speaker 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. Generally 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.

$$\begin{bmatrix} out_{L} \\ out_{R} \\ out_{C} \\ out_{Ls} \\ out_{Rs} \\ out_{Lb} \\ out_{Rb} \\ out_{Tfl} \\ out_{Tfr} \\ out_{Tbl} \\ out_{Tbr} \\ out_{Lfe} \\ out_{Tl} \\ out_{Tr} \\ out_{Tsl} \\ out_{Tsr} \\ out_{Tfc} \\ out_{Tbc} \\ out_{Tc} \\ out_{Lfe2} \\ out_{Bfl} \\ out_{Bfr} \\ out_{Bfc} \\ out_{Cb} \\ out_{Lscr} \\ out_{Rscr} \\ out_{Lw} \\ out_{Rw} \\ out_{Vhl} \\ out_{Vhr} \end{bmatrix} = \begin{bmatrix} r_{0,0} & r_{0,1} & \cdots & r_{0,29} \\ r_{1,0} & r_{1,1} & \cdots & r_{1,29} \\ \vdots & \vdots & \cdots & \vdots \\ r_{29,0} & r_{29,1} & \cdots & r_{29,29} \end{bmatrix} \cdot \begin{bmatrix} in_{L} \\ in_{R} \\ in_{C} \\ in_{Ls} \\ in_{Rs} \\ in_{Lb} \\ in_{Rb} \\ in_{Tfl} \\ in_{Tfr} \\ in_{Tbl} \\ in_{Tbr} \\ in_{Lfe} \\ in_{Tl} \\ in_{Tr} \\ in_{Tsl} \\ in_{Tsr} \\ in_{Tfc} \\ in_{Tbc} \\ in_{Tc} \\ in_{Lfe2} \\ in_{Bfl} \\ in_{Bfr} \\ in_{Bfc} \\ in_{Cb} \\ in_{Lscr} \\ in_{Rscr} \\ in_{Lw} \\ in_{Rw} \\ in_{Vhl} \\ in_{Vhr} \end{bmatrix}$$

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

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`. The coefficients denote $r_{\text{out,in}}$.

Input signals which are not present in the input channel mode should be silenced signals. Output signals which 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 1, n `pan_dialog` value(s) are present (if the input channel configuration is mono, n=1, else n=2). If a channel audio substream is a dialogue substream and `b_pan_dialog_present` is 0, then the pan_dialog value(s) shall default. 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_associate` value is present.

Each `pan_dialog` or `pan_associate` value is indicating 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_associate` 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_associate` 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:

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

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 57, 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 × 90°)` and $r_{k,m}$ = `cos(r × 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 57.

Table 57: 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

After the embedding process specified in step two in clause 5.10.2.1, the substream to be processed is present in the presentation channel mode `pres_ch_mode` (specified in clause 6.3.3.1.27). The decoder shall calculate the signals of the output channel configuration from the signals of the presentation 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
- customized downmix coefficients present in the bitstream

as specified in clause 5.10.2.5.

Table 58 shows how to derive the matrix coefficients by referencing specific tables in clause 5.10.2.5 depending on the output channel configuration.

Table 58: 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 59	Table 60	Table 61	Table 62	Table 63	Table 64	Table 65	Table 66	Table 67

5.10.2.5 Matrix coefficients for channel based renderer for full decoding

This clause contains one table of coefficients $r_{out,in}$ for the rendering matrix for each output channel configuration referenced by Table 58.

The matrix coefficients depend on the input channel configuration indicated by the presentation channel mode `pres_ch_mode` (specified in clause 6.3.3.1.27).

NOTE 1: Coefficients default to 0. The following tables specify non-zero coefficients.

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

Table 59: Channel rendering coefficients for output to 9.X.4 for full decoding

input channel config	coefficients
9.X.4	$r_{i,i}$=0 dB for i={0…10, 24, 25}
9.X.2	$r_{i,i}$=0 dB for i={0…6, 24, 25}, $r_{7,12}=r_{9,12}=r_{8,13}=r_{10,13}=-3$ dB
9.X.0	$r_{i,i}$=0 dB for i={0…6, 24, 25}
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 60: Channel rendering coefficients for output to 9.X.2 for full decoding

input channel config	coefficients
9.X.4	$r_{i,i}$=0 dB for i={0…6, 24..25}, $r_{12,7}=r_{12,9}=r_{13,8}=r_{13,10}=-3$ dB
9.X.2	$r_{i,i}$=0 dB for i={0…6, 12…13, 24…25}
9.X.0	$r_{i,i}$=0 dB for i={0…6, 24…25}
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 61: Channel rendering coefficients for output to 9.X.0 for full decoding

input channel config	coefficients
9.X.4	$r_{i,i}$=0 dB for i={0…6, 24…25}, $r_{3,7}=r_{4,8}=r_{3,9}=r_{4,10}=-3$ dB
9.X.2	$r_{i,i}$=0 dB for i={0…6, 24…25}, $r_{3,12}=r_{4,13}=-3$ dB
9.X.0	$r_{i,i}$=0 dB for i={0…6, 24…25}
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 62: Channel rendering coefficients for output to 7.X.4 for full decoding

input channel config	coefficients
9.X.4	$r_{i,i}$=0 dB for i={0…10}, $r_{0,24}$=$r_{1,25}$=0dB
9.X.2	$r_{i,i}$=0 dB for i={0…6}, $r_{0,24}$=$r_{1,25}$=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}, $r_{0,24}$=$r_{1,25}$=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 63: Channel rendering coefficients for output to 7.X.2 for full decoding

input channel config	coefficients
9.X.4	$r_{i,i}$=0 dB for i={0...6}, $r_{0,24}$=$r_{1,25}$=`gain_f1`, $r_{2,24}$=$r_{2,25}$=`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}, $r_{0,24}$=$r_{1,25}$=`gain_f1`, $r_{2,24}$=$r_{2,25}$=`gain_f2`
9.X.0	$r_{i,i}$=0 dB for i={0...6}, $r_{0,24}$=$r_{1,25}$=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 64: Channel rendering coefficients for output to 7.X.0 for full decoding

input channel config	coefficients
9.X.4	$r_{i,i}$=0 dB for i={0…6}, $r_{0,24}$=$r_{1,25}$=`gain_f1`, $r_{2,24}$=$r_{2,25}$=`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}, $r_{0,24}$=$r_{1,25}$=`gain_f1`, $r_{2,24}$=$r_{2,25}$=`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}, $r_{0,24}$=$r_{1,25}$=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 65: Channel rendering coefficients for output to 5.X.4 for full decoding

input channel config	coefficients
9.X.4	$r_{i,i}$=0 dB for i={0...2, 7...10}, $r_{0,24}=r_{1,25}=\mathit{gain\_f1}$, $r_{2,24}=r_{2,25}=\mathit{gain\_f2}$, $r_{3,3}=r_{3,5}=r_{4,4}=r_{4,6}=\mathit{gain\_b}$
9.X.2	$r_{i,i}$=0 dB for i={0...2}, $r_{0,24}=r_{1,25}=\mathit{gain\_f1}$, $r_{2,24}=r_{2,25}=\mathit{gain\_f2}$, $r_{3,3}=r_{3,5}=r_{4,4}=r_{4,6}=\mathit{gain\_b}$, $r_{7,12}=r_{9,12}=r_{8,13}=r_{10,13}=-3$ dB
9.X.0	$r_{i,i}$=0 dB for i={0...2}, $r_{0,24}=r_{1,25}=0$ 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}, $r_{3,3}=r_{3,5}=r_{4,4}=r_{4,6}=\mathit{gain\_b}$
7.X.2	$r_{i,i}$=0 dB for i={0…2}, $r_{3,3}=r_{3,5}=r_{4,4}=r_{4,6}=\mathit{gain\_b}$, $r_{7,12}=r_{9,12}=r_{8,13}=r_{10,13}=-3$ 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 66: Channel rendering coefficients for output to 5.X.2 for full decoding

input channel config	coefficients
9.X.4	$r_{i,i}$=0 dB for i={0…2}, $r_{0,24}=r_{1,25}=\mathit{gain\_f1}$, $r_{2,24}=r_{2,25}=\mathit{gain\_f2}$, $r_{3,3}=r_{3,5}=r_{4,4}=r_{4,6}=\mathit{gain\_b}$, $r_{12,7}=r_{12,9}=r_{13,8}=r_{13,10}=\mathit{gain\_t1}$
9.X.2	$r_{i,i}$=0 dB for i={0...2, 12, 13}, $r_{0,24}=r_{1,25}=\mathit{gain\_f1}$, $r_{2,24}=r_{2,25}=\mathit{gain\_f2}$, $r_{3,3}=r_{3,5}=r_{4,4}=r_{4,6}=\mathit{gain\_b}$
9.X.0	$r_{i,i}$=0 dB for i={0...2}, $r_{0,24}=r_{1,25}=0$ 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}, $r_{3,3}=r_{3,5}=r_{4,4}=r_{4,6}=\mathit{gain\_b}$, $r_{12,7}=r_{12,9}=r_{13,8}=r_{13,10}=\mathit{gain\_t1}$
7.X.2	$r_{i,i}$=0 dB for i={0…2, 12…13}, $r_{3,3}=r_{3,5}=r_{4,4}=r_{4,6}=\mathit{gain\_b}$
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}, $r_{12,7}=r_{12,9}=r_{13,8}=r_{13,10}=\mathit{gain\_t1}$
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 67: Channel rendering coefficients for output to 5.X.0 for full decoding

input channel config	coefficients
9.X.4	$r_{i,i}$=0 dB for i={0…2}, $r_{0,24}=r_{1,25}=\mathit{gain\_f1}$, $r_{2,24}=r_{2,25}=\mathit{gain\_f2}$, $r_{3,3}=r_{3,5}=r_{4,4}=r_{4,6}=\mathit{gain\_b}$, $r_{0,7}=r_{1,8}=\mathit{gain\_t2a}$, $r_{3,7}=r_{4,8}=\mathit{gain\_t2b}$, $r_{0,9}=r_{1,10}=\mathit{gain\_t2d}$, $r_{3,9}=r_{4,10}=\mathit{gain\_t2e}$
9.X.2	$r_{i,i}$=0 dB for i={0…2}, $r_{0,24}=r_{1,25}=\mathit{gain\_f1}$, $r_{2,24}=r_{2,25}=\mathit{gain\_f2}$, $r_{3,3}=r_{3,5}=r_{4,4}=r_{4,6}=\mathit{gain\_b}$, $r_{0,12}=r_{1,13}=\mathit{gain\_t2a}$, $r_{3,12}=r_{4,13}=\mathit{gain\_t2b}$
9.X.0	$r_{i,i}$=0 dB for i={0…2}, $r_{0,24}=r_{1,25}=0$ 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}, $r_{3,3}=r_{3,5}=r_{4,4}=r_{4,6}=\mathit{gain\_b}$, $r_{0,7}=r_{1,8}=\mathit{gain\_t2a}$, $r_{3,7}=r_{4,8}=\mathit{gain\_t2b}$, $r_{0,9}=r_{1,10}=\mathit{gain\_t2d}$, $r_{3,9}=r_{4,10}=\mathit{gain\_t2e}$
7.X.2	$r_{i,i}$=0 dB for i={0…2}, $r_{3,3}=r_{3,5}=r_{4,4}=r_{4,6}=\mathit{gain\_b}$, $r_{0,12}=r_{1,13}=\mathit{gain\_t2a}$, $r_{3,12}=r_{4,13}=\mathit{gain\_t2b}$
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}, $r_{0,7}=r_{1,8}=\mathit{gain\_t2a}$, $r_{3,7}=r_{4,8}=\mathit{gain\_t2b}$, $r_{0,9}=r_{1,10}=\mathit{gain\_t2d}$, $r_{3,9}=r_{4,10}=\mathit{gain\_t2e}$
5.X.2	$r_{i,i}$=0 dB for i={0…4}, $r_{0,12}=r_{1,13}=\mathit{gain\_t2a}$, $r_{3,12}=r_{4,13}=\mathit{gain\_t2b}$
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

After the embedding process specified in step two in clause 5.10.2.1, 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). The decoder shall calculate the signals of the output channel configuration from the signals of the presentation channel mode for core decoding, 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 this documentation
- customized downmix coefficients present in the bitstream

as specified in clause 5.10.2.7.

Table 68 shows how to derive the matrix coefficients by referencing specific tables in clause 5.10.2.7 depending on the output channel configuration.

Table 68: Mix matrix coefficients for different output channel configurations in core decoding mode

output channel configuration	Location of coefficient definition
5.X.2	Table 69
5.X.0	Table 70

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 68.

The matrix coefficients depend on the input channel configuration indicated by the presentation channel mode `pres_ch_mode` (specified in clause 6.3.3.1.27).

NOTE 1: Coefficients default to 0. The following tables specify non-zero coefficients.

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

NOTE 3: The coefficients depend on additional conditions specified in the third column.

Table 69: Channel rendering coefficients for output to 5.X.2 for core decoding

input channel config	condition	coefficients
9.X.X, 7.X.X	always	$r_{i,i}$=0 dB for i={0…2}
	`top_channels_present=3`	$r_{12,12}$=$r_{13,13}$ =min(0dB,*gain_t1*+3dB)
	`top_channels_present={1,2}`	$r_{12,12}$=$r_{13,13}$=0 dB
	`b_4_back_channels_present=1`	$r_{3,3}$=$r_{3,5}$=$r_{4,4}$=$r_{4,6}$ =min(0dB,*gain_b*+3dB)
	`b_4_back_channels_present=0`	$r_{3,3}$=$r_{4,4}$=0 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 70: Channel rendering coefficients for output to 5.X.0 for core decoding

input channel config	condition	coefficients
9.X.X, 7.X.X	always	$r_{i,i}$=0 dB for i={0…2}
	`top_channels_present={1...3}`	$r_{0,12}$=$r_{1,13}$=min(0dB,*gain_t2a*+3dB), $r_{3,12}$=$r_{4,13}$=min(0dB,*gain_t2b*+3dB)
	`b_4_back_channels_present=1`	$r_{3,3}$=$r_{3,5}$=$r_{4,4}$=$r_{4,6}$=min(0dB,*gain_b*+3dB)
	`b_4_back_channels_present=0`	$r_{3,3}$=$r_{4,4}$=0 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 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 *Intermediate Spatial Format* (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. ISFs preserve 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 signals in the Middle, Upper, Lower and Zenith rings respectively - see Figure 11.

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.

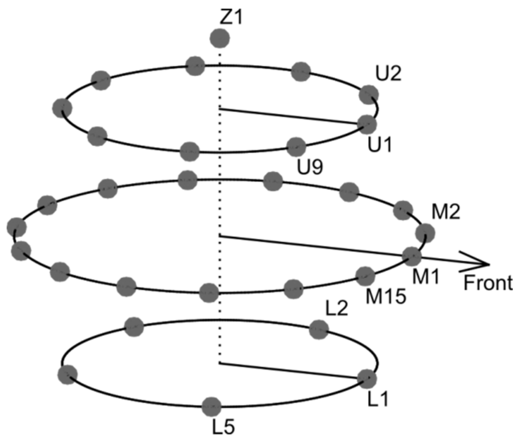

Figure 11: schematic representation of the ISF stacked ring format

5.10.3.3 Interface

5.10.3.3.1 Inputs

$x_{1,...,N\_in}$

N_in ISF input tracks available from the object essence decoder

N_in is the number of ISF tracks available from the object essence decoder.

5.10.3.3.2 Outputs

$y_{1,...,N\_out}$

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 tracks as defined in clause 5.10.3.2.

5.10.3.4 Processing

The ISF rendering algorithm maps ISF tracks to speaker feeds with coefficients configured by the output channel config.

To process ISF tracks, 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 outputs from the object essence decoder successively to elements of column vector t = [M_1..., U_1..., L_1..., Z].

NOTE 1: The exact layout of t is determined by `isf_config`, see Table 83. LFE channels stays unassigned, i.e. pass by the ISF tool.

3) Transform the signal into speaker feeds by $y = \mathbf{M} \times t$.

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 $29{,}97[\text{Hz}] = 30 \cdot \frac{1000}{1001}[\text{Hz}]$, each frame measures $\frac{48000}{30} \cdot \frac{1001}{1000} = 8 \cdot \frac{1001}{5}$ 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 71, with:

$$\boldsymbol{\phi_t} = \begin{cases} \mathbf{cnt}\,\boldsymbol{mod}\mathbf{5} & \boldsymbol{if}\,\mathbf{cnt} \neq \mathbf{0} \\ (\boldsymbol{\phi_{t-1}} + \mathbf{1})\boldsymbol{mod}\mathbf{5} & \boldsymbol{if}(\mathbf{cnt} = \mathbf{0}) \wedge \text{(this is not the first frame)} \\ \mathbf{0} & \boldsymbol{else} \end{cases}$$

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 71.

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 71: 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 current specification 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 72. Some syntactic elements are amended as indicated in Table 73. This clause specifies amended and newly defined syntactic elements.

Table 72: 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)
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)

syntactic element	location in ETSI TS 103 190-1 [1]
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 73: Amended syntactic elements specified in ETSI TS 103 190-1 [1]

syntactic element	location in ETSI TS 103 190-1 [1]	location in present document
ac4_toc	clause 4.2.3.1 (table 4)	
ac4_presentation_info	clause 4.2.3.2 (table 5)	
presentation_config_ext_info	clause 4.2.3.8 (table 11)	
ac4_hsf_ext_substream_info	clause 4.2.3.9 (table 12)	
ac4_substream	clause 4.2.4.1 (table 16)	
audio_data	clause 4.2.5 (table 19)	
metadata	clause 4.2.14.1 (table 66)	
basic_metadata	clause 4.2.14.2 (table 67)	
further_loudness_info	clause 4.2.14.3 (table 68)	
extended_metadata	clause 4.2.14.4 (table 69)	
dialog_enhancement	clause 4.2.14.11 (table 76)	
de_data	clause 4.2.14.13 (table 78)	

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 {
    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;
```

```
Syntax                                                                              No of bits
    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 {
    mdcompat; ......................................................................... 3
    b_presentation_group_index; ....................................................... 1
    if (b_presentation_group_index) {
      presentation_group_index = 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();
          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();
```

Syntax	No of bits

```
        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();
  }
  if (b_single_substream_group != 1 and presentation_config == 6) {
    b_add_emdf_substreams = 1;
  }
  else {
    if (bitstream_version != 1) {
      mdcompat; .............................................. 3
    }
    b_presentation_group_index; .............................. 1
    if (b_presentation_group_index) {
      presentation_group_index = variable_bits(2);
    }
    frame_rate_multiply_info();
    frame_rate_fractions_info();
    emdf_info();
```

Syntax	No of bits

```
    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:
          ac4_sgi_specifier();
          ac4_sgi_specifier();
          n_substream_groups = 2;
          break;
        case 1:
          ac4_sgi_specifier();
          ac4_sgi_specifier();
          n_substream_groups = 1;
          break;
        case 2:
          ac4_sgi_specifier();
          ac4_sgi_specifier();
          n_substream_groups = 2;
          break;
        case 3:
          ac4_sgi_specifier();
          ac4_sgi_specifier();
          ac4_sgi_specifier();
          n_substream_groups = 3;
          break;
        case 4:
          ac4_sgi_specifier();
          ac4_sgi_specifier();
          ac4_sgi_specifier();
          n_substream_groups = 2;
          break;
        case 5:
          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:
          presentation_config_ext_info();
          break;
      }
    }
    b_pre_virtualized; .......................................................... 1
    b_add_emdf_substreams; ...................................................... 1
    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

<pre>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;
      if (b_frame_rate_fraction == 1) {
        frame_rate_fraction = 2;
      }
    }
  }
  if (frame_rate_index in [10, 11, 12]) {
    b_frame_rate_fraction;
    if (b_frame_rate_fraction == 1) {
      b_frame_rate_fraction_is_4;
      if (b_frame_rate_fraction_is_4 == 1) {
        frame_rate_fraction = 4;
      }
      else {
        frame_rate_fraction = 2;
      }
    }
  }
}</pre>
	
	1
	
	1
	
	1

6.2.1.5 presentation_config_ext_info

Syntax	No of bits

<pre>presentation_config_ext_info()
{
  n_skip_bytes;
  b_more_skip_bytes;
  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_bits_read += n_skip_bits;
    }
    n_skip_bytes = n_skip_bytes - (n_bits_read / 8);
  }
  for (i = 0; i < n_skip_bytes; i++) {
    reserved;
  }
}</pre>
	
	5
	1
	
	n_skip_bits
	
	8

6.2.1.6 ac4_substream_group_info

Syntax	No of bits

<pre>ac4_substream_group_info()
{
  b_substreams_present;
  b_hsf_ext;
  b_single_substream;
  if (b_single_substream) {
    n_lf_substreams = 1;
  }
  else {
    n_lf_substreams_minus2;
    n_lf_substreams = n_lf_substreams_minus2 + 2;
    if (n_lf_substreams == 5) {
      n_lf_substreams += variable_bits(2);
    }
  }
  b_channel_coded;
  if (b_channel_coded) {
    for (sus = 0; sus < n_lf_substreams; sus++) {</pre>
	
	1
	1
	1
	
	2
	
	1

Syntax	No of bits
`      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);`	
`    }`	
`    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) {`	

Syntax	No of bits
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) {	
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)	
{	
b_dyn_objects_only;	1
if (b_dyn_objects_only == 0) {	
b_isf;	1
if (b_isf) {	
isf_config;	3
}	
else {	
b_ch_assign_code;	1
if (b_ch_assign_code) {	
bed_chan_assign_code;	3
}	
else {	
b_chan_assign_mask;	1
if (b_chan_assign_mask) {	
b_nonstd_bed_channel_assignment;	1
if (b_nonstd_bed_channel_assignment) {	
nonstd_bed_channel_assignment_mask;	17
}	
else {	
std_bed_channel_assignment_mask;	10
}	
}	
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
}	
}	
}	
}	
}	
}	

6.2.1.11 ac4_substream_info_obj

Syntax	No of bits
ac4_substream_info_obj(b_substreams_present)	
{	
n_objects_code;	3
b_dynamic_objects;	1
if (b_dynamic_objects) {	
b_lfe;	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
}	
else {	
b_nonstd_bed_channel_assignment;	1
if (b_nonstd_bed_channel_assignment) {	
nonstd_bed_channel_assignment_mask;	17
}	
else {	
std_bed_channel_assignment_mask;	10
}	
}	
}	
}	

Syntax	No of bits
`    else {`	
`      b_isf;`	1
`      if (b_isf) {`	
`        b_isf_start;`	1
`        if (b_isf_start) {`	
`          isf_config;`	3
`        }`	
`      }`	
`      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
`  }`	
`  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 which refers to this specific substream. The mapping for the info elements defined in the present document is given in Table 74, which is an extension of ETSI TS 103 190-1 [1], table 15.

Table 74: 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()

ac4_hsf_ext_substream() and emdf_payloads_substream() are specified in ETSI TS 103 190-1 [1] and 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, sus_ver);
  byte_align; ............................................................................. 0…7
}
```

NOTE: n_objects is derived from n_objects_code according to Table 87.

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);
    }
    for (t = 0; t < n_targets; t++) {
      target_level; ............................................................ 3
      target_device_category; .................................................. 5
      if (target_device_category & 0b00001 == 1) {
        target_device_category <<= 4;
        tdc_extension; ......................................................... 4
        target_device_category += tdc_extension;
      }
      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;
    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) {
```

Syntax	No of bits
`      for (sg = 0; sg < n_substream_groups; sg++) {`	
`        **sg_gain[sg]**;`	6
`      }`	
`    }`	
`  }`	
`}`	
`**b_associated**;`	1
`if (b_associated == 1) {`	
`  **b_scale_main**;`	1
`  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);`	
`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
`}`	

6.2.3 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:`	

Syntax	No of bits

<pre>   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;
  }
}</pre>	

6.2.3.2 audio_data_objs

Syntax	No of bits

<pre>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);
  }
}</pre>	

6.2.3.3 objs_to_channel_mode

Syntax	No of bits

<pre>objs_to_channel_mode(n_objects)
{
  switch (n_objects) {
    case 1:
      return mono;
      break;
    case 2:
      return stereo;
      break;
    case 3:
      return 3.0;
      break;
    case 5:
      return 5.0;
      break;
  }
}</pre>	

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_bytes = variable_bits(3) + 1;
      skip_data; ................................................. 8 * skip_bytes
    }
  }
  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_mask; .......................................... num_umx_signals
  }
  if (b_keep_dmx_de_coeffs == 0) {
    for (dio = 0; dio < num_dlg_objs; dio++) {
      for (dmxo = 0; dmxo < num_dmx_signals; dmxo++) {
        de_dlg_dmx_coeff_idx[dio][dmxo]; ................................. VAR
      }
    }
  }
}
```

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;	1/3
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) {	
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();	

Syntax	No of bits

```
    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 Table 97. core_channel_config is derived from immersive_codec_mode as specified in Table 98.

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) {
    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) {
    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_sparse_mask_dry[o][ch]; ............................ 1
          }
          for (d = 0; d < ajoc_num_decorr; d++) {
            if (ajoc_decorr_enable[d]) {
              ajoc_sparse_mask_wet[o][d]; ........................... 1
            }
            else {
              ajoc_sparse_mask_wet[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++) {
    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];
        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;
        }
        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_sparse_mask_dry[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_sparse_mask_wet[o][de]) {
                mix_mtx_wet[o][dp][de] = ajoc_huff_data(WET, nb, qs, b_dfonly);
              }
            }
            break;
```

Syntax	No of bits
` }`	
` }`	
` }`	
` }`	
`}`	

NOTE: The ajoc_num_bands values are derived using Table 102.

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
  }
  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
```

Syntax	No of bits
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);	
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);
    }
  }
  return a_huff_data;
}
```

6.2.7 Metadata

6.2.7.1 metadata

Syntax	No of bits

```
metadata(b_alternative, b_ajoc, b_iframe, sus_ver)
{
  basic_metadata(sus_ver);
  extended_metadata(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) {
        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
          }
```

Syntax	No of bits
`        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
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
}	

```
Syntax                                                                No of bits
      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
    }
  }
  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, b_associated, b_dialog, 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
      }
```

Syntax	No of bits

```
      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) {
        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_LrsRrs()) {
      b_lrs_active; ............................................................ 1
      b_rrs_active; ............................................................ 1
    }
    if (channel_mode_contains_LwRw()) {
      b_lw_active; ............................................................. 1
      b_rw_active; ............................................................. 1
    }
    if (channel_mode_contains_VhlVhr()) {
      b_vhl_active; ............................................................ 1
      b_vhr_active; ............................................................ 1
    }
    if (channel_mode_contains_Lfe()) {
      b_lfe_active; ............................................................ 1
    }
  }
  b_event_probability; ......................................................... 1
  if (b_event_probability) {
    event_probability; ......................................................... 4
  }
}
```

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) {	
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 {	

Syntax	No of bits

```
          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) {
    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;
    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
```

```
Syntax                                                                                      No of bits
      }
    }
  }
}
```

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);
    }
  }
  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_bytes = variable_bits(2) + 1;
        skip_data; ............................................................ 8 * skip_bytes
      }
```

Syntax	No of 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;
      }
      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;
          }
        }
```

```
Syntax                                                                                 No of bits
      }
    }
    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;
    add_table_data; ........................................................ 8 * atd_size
  }
}
```

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) {
    obj_render_info_mask = 0b111;
  }
  else {
    obj_render_info_mask; .......................................................... 3
  }
  if (obj_render_info_mask & 0b001) {
    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 (obj_render_info_mask & 0b010) {
    b_grouped_zone_defaults; ....................................................... 1
    if (b_grouped_zone_defaults == 0) {
      group_zone_mask; ............................................................. 3
      if (group_zone_mask & 0b001) {
        zone_mask; ................................................................. 3
```

Syntax	No of bits

```
      }
      if (group_zone_mask & 0b010) {
        b_enable_elevation = 0;
      }
      if (group_zone_mask & 0b100) {
        b_object_snap = 1;
      }
    }
  }
  if (obj_render_info_mask & 0b100) {
    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
          }
        }
      }
    }
  }
}
```

6.2.9 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
    }
```

```
Syntax                                                          No of bits
    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) {
      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
}	
}	
}	

6.2.9.3 cdmx_parameters

Syntax	No of bits

<pre>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;
    }
  }
}</pre>	

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
}	
else {	
b_top_front_to_side;	1
if (b_top_front_to_side == 1) {	
gain_t2b_code;	3
}	
else {	
gain_t2c_code;	3
}	
}	
b_top_back_to_front;	1
if (b_top_back_to_front == 1) {	
gain_t2d_code;	3
}	
else {	
b_top_back_to_side;	1
if (b_top_back_to_side == 1) {	
gain_t2e_code;	3
}	
else {	
gain_t2f_code;	3
}	
}	
}	

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
`  }`	
`  else {`	
`    gain_t2b_code;`	3
`  }`	
`  b_top_back_to_front;`	1
`  if (b_top_back_to_front == 1) {`	
`    gain_t2d_code;`	3
`  }`	
`  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
`  }`	
`  else {`	
`    b_top_to_side;`	1
`    if (b_top_to_side == 1) {`	
`      gain_t2b_code;`	3
`    }`	
`    else {`	
`      gain_t2c_code;`	3
`    }`	
`  }`	
`}`	

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
`  }`	
`  else {`	
`    gain_t2b_code;`	3
`  }`	
`}`	

6.3 Description of bitstream elements

6.3.1 Introduction

The description of bitstream elements is done similar to the description in ETSI TS 103 190-1 [1]. Elements which 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 info

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 75: 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 content 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 bitrate for average bitrate streams.

For average bitrate streams, over successive frames a br_code value of 0b11 is followed by a sequence of br_code values different from 0b11. That sequence of values can be used to improve the bitrate estimation compared to simple averaging of the size of the analysed frames. An algorithm for calculating the signalled bitrate is described in Annex B.

Table 76: 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 flag indicates whether all substreams in all presentations are encoded independently from preceding frames (no dependency over time). b_audio_ndot = 1, b_pres_ndot = 1 and b_oamd_ndot = 1 indicate whether there is no dependency over time for the corresponding substream type. In case frame_rate_factor ≠ 1, the first b_audio_ndot of a series of 2 or 4 substreams has to be 1 (true) to fulfil this.

6.3.2.1.4 b_program_id

This flag 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 flag indicates whether program identification as 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 flag indicates whether a single substream group is present. If not set, the number of substream groups, *n_substream_groups*, is 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 77. The presentation configuration for presentation_version = 0 is specified in ETSI TS 103 190-1 [1], clause 4.3.3.3.4.

Table 77: presentation_config for presentation_version = 1

presentation_config	Presentation configuration
0	Music and Effects (M+E) + Dialogue
1	Main + DE
2	Main + Associate
3	Music and Effects (M+E) + Dialogue + Associate
4	Main + DE + Associate
5	Arbitrary substream groups
6	EMDF only
≥ 7	Reserved

6.3.2.2.3 mdcompat

This field indicates the decoder compatibility as shown in Table 78.

The mdcompat element indicates which decoder systems a presentation is compatible with.

A decoder system with compatibility level *n* shall be able to decode all presentations with mdcompat ≤ *n;* and

- presentation_version = 0 as defined in ETSI TS 103 190-1 [1], clause 4.3.3.3.8;
- presentation_version = 1 as defined in Table 78.

A system with compatibility level *n* shall not decode (i.e. select) presentations with mdcompat > *n*.

NOTE: A presentation may consist of several substreams, which may be distributed over several elementary streams.

Table 78: mdcompat for presentation_version = 1

<table>
<tr><th rowspan="2">mdcompat</th><th colspan="2">Maximum number of tracks if presentation includes</th><th rowspan="2">Maximum input channel configuration for channel based immersive</th><th rowspan="2">Maximum reconstructed A-JOC objects</th></tr>
<tr><th>no object audio</th><th>object audio</th></tr>
<tr><td>0</td><td>2</td><td>n/a</td><td>n/a</td><td>n/a</td></tr>
<tr><td>1</td><td>6</td><td>6</td><td>n/a</td><td>15.1</td></tr>
<tr><td>2</td><td>9</td><td>8</td><td>7.1.4</td><td>15.1</td></tr>
<tr><td>3</td><td>11</td><td>10</td><td>7.1.4</td><td>17.1</td></tr>
<tr><td>4-6</td><td colspan="4">Reserved</td></tr>
<tr><td>7</td><td colspan="4">Unrestricted</td></tr>
</table>

6.3.2.2.4 b_presentation_group_index

This flag indicates whether the presentation belongs to a presentation group and that a presentation group index is present in the bitstream. The presentation group index *presentation_group_index* is derived from additional variable_bits().

6.3.2.2.5 b_presentation_filter

This flag indicates whether the b_enable_presentation bit is present.

6.3.2.2.6 b_enable_presentation

This flag indicates whether a presentation is enabled.

6.3.2.2.7 b_multi_pid

This flag indicates whether the presentation is a multiple PID presentation.

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 which is decodable by an AC-4 decoder conforming to [1] or to the present document. A decoder implemented in accordance to the present document shall decode a presentation with a presentation version value of 1. A decoder implemented in accordance to the present document shall skip the ac4_presentation_info or ac4_presentation_v1_info if the version of the presentation is not 0 or 1.

6.3.2.4 frame_rate_fractions_info - Frame rate fraction information

6.3.2.4.1 b_frame_rate_fraction

This flag 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 flag 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 flag 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 lf substreams present in a presentation. To get the number of lf substreams, *n_lf_substreams*, a value of 2 needs to be added to n_lf_substreams_minus2. Additional variable_bits() are used to derive the number of lf substreams for values of *n_lf_substreams* exceeding 4.

6.3.2.5.3 b_channel_coded

This flag 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 [1].

A value of 1 indicates the usage of an extended syntax for the AC-4 substream. If `sus_ver` is not set, the value defaults to 0.

6.3.2.5.5 b_oamd_substream

This flag indicates whether a substream containing object audio metadata is present.

6.3.2.5.6 b_ajoc

This flag 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 77.

EXAMPLE: For `presentation_config` = 4, the "Main" substream group is indicated first, followed by the "DE" substream group and the "Associate" 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 to signal and represent 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 chose not to decode them.

6.3.2.7.2 channel_mode

This variable length field indicates the channel mode and the variable *`ch_mode`* as shown in Table 79, which is an extension of ETSI TS 103 190-1 [1], table 87.

Table 79: 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,Lrs,Rrs)	5
0b1111001	7.1: 3/4/0.1 (L,C,R,Ls,Rs,Lrs,Rrs,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,Vhl,Vhr)	9
0b1111101	7.1: 3/2/2.1 (L,C,R,Ls,Rs,Vhl,Vhr,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 79, this fixed length field signals whether those channels are present in the original content or only contain encoded silence as shown in Table 80.

Table 80: b_4_back_channels_present

`b_4_back_channels_present`	Original content	Encoded AC-4 bitstream
0	Original content contains two surround channels: Ls, Rs	Surround channels of original content are carried in Ls, Rs; Lb, Rb contain silence
1	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 centre channel C is present in the bitstream as signalled by Table 79, this fixed length field signals whether it actually is present in the original content or contains encoded silence as shown in Table 81.

Table 81: b_centre_present

`b_centre_present`	Original content	Encoded AC-4 bitstream
0	Original content does not contain a centre channel C	C contains silence
1	Original content contains a centre channel C	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 79, this fixed length field signals, as shown in Table 82, whether all of those channels are present in the original content, whether the original content has only two top channels (Tl, Tr) and how they are carried, or whether these channels contain encoded silence.

Table 82: 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: Tl and Tr	Original content of Tl, Tr is carried in Tfl, Tfr; Tbl, Tbr contain silence
2	Original content contains two top channels: Tl and Tr	Original content of Tl, Tr 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 flag indicates whether an audio substream can be decoded independently of preceding frames.

6.3.2.8 ac4_substream_info_ajoc - AC-4 substream information for object based substreams using A-JOC

6.3.2.8.1 b_lfe

This flag indicates whether a LFE is present.

6.3.2.8.2 b_static_dmx

This flag indicates whether a static downmix is present.

6.3.2.8.3 n_fullband_dmx_signals_minus1

The `n_fullband_dmx_signals_minus1` element indicates the number of fullband downmix signals minus 1. To get the number of fullband downmix signals, *`n_fullband_dmx_signals`*, a value of 1 needs to be added to `n_fullband_dmx_signals_minus1`.

6.3.2.8.4 b_oamd_common_data_present

This flag indicates whether OAMD common data is present.

6.3.2.8.5 n_fullband_upmix_signals_minus1

The `n_fullband_upmix_signals_minus1` element indicates the number of fullband upmix signals minus 1. To get the number of fullband upmix signals, *`n_fullband_upmix_signals`*, a value of 1 needs to be added to `n_fullband_upmix_signals_minus1`.

6.3.2.9 bed_dyn_obj_assignment - Bed and dynamic object assignment

6.3.2.9.1 b_dyn_objects_only

This flag indicates whether only dynamic objects are present.

6.3.2.9.2 b_isf

This flag indicates whether one or more ISF objects are present.

6.3.2.9.3 isf_config

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 83. The intermediate spatial format objects are ordered as presented in the table.

Table 83: 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

6.3.2.9.4 b_ch_assign_code

The `b_ch_assign_code` flag indicates whether the channel assignment is defined by `bed_chan_assign_code`.

6.3.2.9.5 bed_chan_assign_code

The `bed_chan_assign_code` describes bed object channel configuration, number of channels and channel order according to Table 84.

Table 84: bed_chan_assign_code

bed_chan_assign_code	channels	channel configuration description	number of channels
0b000	L/R	2.0.0	2
0b001	L/R/C	3.0.0	3
0b010	L/R/C/LFE/Ls/Rs	5.1.0	6
0b011	L/R/C/LFE/Ls/Rs/Tl/Tr	5.1.2	8
0b100	L/R/C/LFE/Ls/Rs/Tfl/Tfr/Tbl/Tbr	5.1.4	10
0b101	L/R/C/LFE/Ls/Rs/Lb/Rb	7.1.0	8
0b110	L/R/C/LFE/Ls/Rs/Lb/Rb/Tl/Tr	7.1.2	10
0b111	L/R/C/LFE/Ls/Rs/Lb/Rb/Tfl/Tfr/Tbl/Tbr	7.1.4	12

6.3.2.9.6 b_chan_assign_mask

The `b_chan_assign_mask` flag indicates whether the channel assignment is defined by a channel assignment mask.

6.3.2.9.7 b_nonstd_bed_channel_assignment

The `b_nonstd_channel_assignment` flag indicates whether the channel assignment is defined by `nonstd_bed_channel_assignment_mask`.

6.3.2.9.8 nonstd_bed_channel_assignment_mask

The `nonstd_bed_channel_assignment_mask` is a bitmask that describes the channel assignment whenever the channel assignment is non-standard, i.e. when some channels do not appear as part of channel pairs. For example, when the R channel is present but the corresponding L channel is not. Each bit in the field represents a different individual channel present in the stream according to Table 85. When bit = 1 the channel exists; when bit = 0 there is no assignment. Channels follow in the order they are presented in the table.

Table 85: nonstd_bed_channel_assignment_mask

nonstd_bed_channel_assignment_mask	nonstd_bed_channel_assignment	channel
0b00000000000000001	0	L
0b00000000000000010	1	R
0b00000000000000100	2	C
0b00000000000001000	n/a	LFE
0b00000000000010000	4	Ls
0b00000000000100000	5	Rs
0b00000000001000000	6	Lb
0b00000000010000000	7	Rb
0b00000000100000000	8	Tfl
0b00000001000000000	9	Tfr
0b00000010000000000	10	Tl
0b00000100000000000	11	Tr
0b00001000000000000	12	Tbl
0b00010000000000000	13	Tbr
0b00100000000000000	14	Lw
0b01000000000000000	15	Rw
0b10000000000000000	n/a	LFE2

6.3.2.9.9 std_bed_channel_assignment_mask

The presence or absence of bed object channels can be flagged in the field `std_bed_channel_assignment_mask`. Each bit in the field represents a different channel or channel pair present in the bitstream according to Table 86. If the corresponding bit is set to 1 then the channel(s) exist(s); if the corresponding bit is set to 0 there is no assignment. Channels follow in the order they are presented in the table.

Table 86: std_bed_channel_assignment_mask

std_bed_channel_assignment_mask	channel	number of channels
0b0000000001	L/R	2
0b0000000010	C	1
0b0000000100	LFE	1
0b0000001000	Ls/Rs	2
0b0000010000	Lb/Rb	2
0b0000100000	Tfl/Tfr	2
0b0001000000	Tl/Tr	2
0b0010000000	Tbl/Tbr	2
0b0100000000	Lw/Rw	2
0b1000000000	LFE2	1

6.3.2.9.10 n_bed_signals_minus1

The `n_bed_signals_minus1` element indicates the number of bed signals minus 1. To get the number of bed signals, *`n_bed_signals`*, a value of 1 needs to be added to `n_bed_signals_minus1`.

6.3.2.9.11 nonstd_bed_channel_assignment

The channel assignment is given via the `nonstd_bed_channel_assignment` value according to Table 85.

6.3.2.10 ac4_substream_info_obj - AC-4 substream information for object based substreams

6.3.2.10.1 n_objects_code

The `n_objects_code` element describes the number of fullband signals in the substreams according to Table 87.

Table 87: n_objects_code

n_objects_code	n_objects (Number of objects)
0	0
1	1
2	2
3	3
4	5

6.3.2.10.2 b_dynamic_objects

The `b_dynamic_objects` flag indicates whether the substream contains dynamic objects.

6.3.2.10.3 b_lfe

For `b_dynamic_objects`=1, this flag indicates whether a LFE is present. For `b_dynamic_objects`=0, the presence of a LFE is signalled by `bed_chan_assign_code`, `nonstd_bed_channel_assignment_mask`, or `std_bed_channel_assignment_mask`.

6.3.2.10.4 b_bed_objects

The `b_bed_objects` flag indicates whether the substream contains bed objects.

6.3.2.10.5 b_bed_start

The `b_bed_start` flag indicates whether the substream contains the channel information for the bed according to Table 88. Consecutive substreams are ordered according to their channel assignments. If one LFE exists, it is part of the first substream. If two LFE channels exist, one of them is in the first substream, the other in the second substream.

Table 88: b_bed_start

b_bed_start	Description
0	Substream does not contain bed channel information
1	Substream is either first or singular substream in the bed and contains bed channel information

6.3.2.10.6 b_isf_start

The `b_isf_start` flag indicates whether the substream is the first or a singular ISF substream and contains ISF config information.

6.3.2.10.7 res_bytes

This element specifies the size of the `reserved_data` element in bytes.

6.3.2.10.8 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 bit indicates whether an alternative presentation is present.

6.3.2.11.2 b_pres_ndot

This bit 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 flag 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 flag indicates whether a presentation name is present.

6.3.3.1.2 b_length

The length of the `presentation_name` field is 32 bytes if `b_length` = 0, or transmitted as `name_len` otherwise.

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 like the `mdcompat` element indicates the decoder compatibility for a presentation. See Table 78.

6.3.3.1.7 target_device_category

The `target_device_category` element indicates the device categories of the target as a bit mask according to Table 89.

Table 89: target_device_category

target_device_category (incl. tdc_extension)	Description
0b1xxxx	stereo speaker (1D)
0bx1xxx	5.1 (2D)
0bxx1xx	Including height (3D)
0bxxx1x	Portable
0bxxxx11xxx, 0bxxxx1x1xx, 0bxxxx1xx1x, 0bxxxx1xxx1	future extensions

6.3.3.1.8 tdc_extension

The tdc_extension element is used as an extension of the target_device_category element. See Table 89.

6.3.3.1.9 b_ducking_depth_present

This flag 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 90.

Table 90: 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 flag 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.

$$\text{target\_corr\_gain} = (15 - \text{loud\_corr\_target})/2\,[\text{dB}_2]$$

A value of 31 is reserved, and if present indicates a gain of 0 dB.

6.3.3.1.13 n_substreams_in_presentation

This helper variable indicates the number of audio substreams (incl. e.g. 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 flag indicates whether 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 > 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 flag 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.

6.3.3.1.18 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 set, the DRC metadata size is increased by `variable_bits`.

6.3.3.1.20 b_more_bits

This bit 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 91 for the immersive and 22.2 channel configurations.

Table 91: nr_drc_channels for higher channel modes

Channel config	*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 91 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 flag indicates whether gain values for the substream groups are present.

6.3.3.1.23 b_keep

This flag indicates whether to use the same substream group gains as in the previous AC-4 frame. If the `b_keep` flag is not set, new substream group gains are present in the bitstream. Until the first reception of an AC-4 frame with `b_keep` = 0, 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 92. The substream group gain is applied to all substreams in the substream group *sg*.

Table 92: 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 flag 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 Table 93.

Table 93: Pseudocode

```
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;
}
```

superset() is a function which 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 79), *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 94.

Table 94: ch_mode_core

Substream properties	Core channel mode	*ch_mode_core*
In ac4_substream_group_info: b_channel_coded = 0 and b_ajoc = 1, in ac4_substream_info_ajoc: b_static_dmx = 1 and b_lfe = 0	5.0	3
In ac4_substream_group_info: b_channel_coded = 0 and b_ajoc = 1, in ac4_substream_info_ajoc: b_static_dmx = 1 and b_lfe = 1	5.1	4
In ac4_substream_group_info: b_channel_coded = 1, in ac4_substream_info_chan: ch_mode in [11,13]	5.0.2 core	5
In ac4_substream_group_info: b_channel_coded = 1, 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 Table 95 describes the determination of *pres_ch_mode_core*, which shall be done after the determination of *pres_ch_mode*.

Table 95: Pseudocode

```
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;
}
```

superset() is a function which 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.

6.3.3.1.30 pres_top_channel_pairs

This helper variable shall be initialized according to Table 96.

Table 96: 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 helper flag indicates whether a LFE is present in the presentation. It is set if at least one substream in the presentation contains a LFE. If pres_ch_mode $\geq$ 0, the presence of a LFE is determined by the function channel_mode_contains_Lfe(). If pres_ch_mode is -1, the presence of a LFE is determined by the condition that pres_ch_mode_core is 4 or 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

The b_oamd_common_data_present flag indicates whether oamd_common_data is present in the corresponding oamd_substream.

6.3.3.2.3 b_oamd_timing_present

The b_oamd_timing_present flag 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

The b_some_signals_inactive flag 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 dmx_active_signals_mask element signals which of the signals present in var_channel_element contain coded silence.

NOTE: The bit mask is ordered as follows: MSB is the first signal in bitstream order, LSB is the last signal in bitstream order.

6.3.4.3 b_dmx_timing

The b_dmx_timing flag indicates whether an individual OAMD timing data instance is present for core decoding mode.

6.3.4.4 b_oamd_extension_present

The b_oamd_extension_present flag 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

The b_umx_timing flag indicates whether an individual OAMD timing data instance is present for full decoding mode.

6.3.4.7 b_derive_timing_from_dmx

The b_derive_timing_from_dmx flag indicates whether the OAMD timing data instance 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 first one bit shall be read, if this bit is zero another two bits shall be read. The bitcode indicates the *immersive_codec_mode* for the immersive_channel_element as shown in Table 97.

Table 97: 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 which is used in the bitstream syntax of the immersive_channel_element and can be derived from the *immersive_codec_mode* as shown in Table 98.

Table 98: 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 99.

Table 99: 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 100.

Table 100: 22.2 codec mode

22_2_codec_mode	22.2 codec mode
0 = SIMPLE	Simple
1 = ASPX	A-SPX

6.3.5.5 var_codec_mode

This bit indicates the codec mode of the `var_channel_element()` as shown in Table 101.

Table 101: var codec mode

var_codec_mode	var codec mode
0 = SIMPLE	Simple
1 = ASPX	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 102.

Table 102: 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 103.

Table 103: 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 104.

Table 104: ajoc_sparse_select

ajoc_sparse_select	Meaning
0	Upmix from all
1	Upmix from some - read sparse mask values

6.3.6.2.6 ajoc_sparse_mask_dry[o][ch]

Sparse mask values for the dry matrix elements. This bit is indicating if the dry matrix element should be included which means the given input *ch* is factored into the reconstruction of the corresponding output *o*.

6.3.6.2.7 ajoc_sparse_mask_wet[o][d]

Sparse mask values for the wet matrix elements. This bit is indicating if the wet matrix element should be included which means the given decorrelator output *d* is factored into the reconstruction of the corresponding 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 an reconstruction matrix element from old to new values.

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, from old to new values of an reconstruction matrix element 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 105.

Table 105: 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 Table 106 describes how to determine the correct Huffman table for decoding the Huffman code words.

Table 106: Pseudocode

```
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

    // These 12 Huffman codebooks are given in clause A.1.1
    // and are named according to the schema outlined above.

    return ajoc_hcb;
}
```

6.3.6.6 ajoc_dmx_de_data

6.3.6.6.1 b_dmx_de_cfg

This flag indicates whether a dialogue enhancement configuration is present.

6.3.6.6.2 b_keep_dmx_de_coeffs

This flag indicates whether to use the same dialogue downmix coefficients as in the previous AC-4 frame. If the b_keep_dmx_de_coeffs flag is not set, new dialogue downmix coefficients are present in the bitstream.

6.3.6.6.3 de_main_dlg_mask

The de_main_dlg_mask element is a bitmask indicating which objects represent the main dialogue that should be boosted when dialogue enhancement is enabled. A binary one denotes a DE enabled object. The function *dlg_obj()*, as specified in Table 107, shows how to derive the helper elements *num_dlg_obj* and *dlg_idx[]* from the de_main_dlg_mask 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.

Table 107: Pseudocode

```
dlg_obj()
{
  num_dlg_obj = 0;
  for (obj = 0; obj < num_umx_signals; obj++)
  {
    if (de_main_dlg_mask & (1 << (num_umx_signals - obj - 1)))
    {
       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 108.

Table 108: 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

6.3.7 Advanced Joint Channel Coding (A-JCC)

6.3.7.1 ajcc_data

6.3.7.1.1 b_no_dt

The b_no_dt flag indicates whether the present parameters are delta-frequency coded, or if some of the present parameters could also be delta-time coded, as shown in Table 109.

Table 109: b_no_dt

b_no_dt	Description
0	present parameters are delta-frequency or delta-time coded
1	all parameters are delta-frequency coded

6.3.7.1.2 ajcc_num_param_bands_id

The `ajcc_num_param_bands_id` element indicates the number of parameter bands as index to Table 110.

Table 110: 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` indicates the channel configuration of the Advanced Joint Channel coded core, as shown in Table 111.

Table 111: 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 in case of *`b_5fronts`* = 0.

6.3.7.1.4 ajcc_qm_f

The `ajcc_qm_f` indicates the quantization mode for the parameter which are related to the front channels, as shown in Table 112.

Table 112: 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 parameter which are related to the back channels, as shown in Table 113.

Table 113: 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 114.

Table 114: 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 115.

Table 115: 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 116.

Table 116: 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

This element indicates the value `ajcc_num_param_sets` as shown in Table 117. `ajcc_num_param_sets` describes how many A-JCC parameter sets per frame are transmitted in the bitstream.

Table 117: 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 Table 118 describes how to determine the correct Huffman table for decoding the Huffman code words.

Table 118: Pseudocode

```
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>;
        // the line above expands using the inputs data_type, quant_mode and hcb_type
        // These 12 Huffman codebooks are given in TS 103 190-1 [1], clause A.3
        // and are named according to the schema outlined above.
    }
    else {
        ajcc_hcb = AJCC_HCB_<data_type>_<quant_mode>_<hcb_type>;
        // These 12 Huffman codebooks are given in clause A.1.2.
    }
    return ajcc_hcb;
}
```

6.3.8 Metadata

6.3.8.1 basic_metadata - Basic metadata

6.3.8.1.1 b_substream_loudness_info

This flag 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 ¼ 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 flag indicates whether additional substream loudness information is present in a `further_loudness_info()` element.

6.3.8.1.4 dialnorm_bits

The `dialnorm_bits` element signals 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 music and effects (M+E) substream and the dialogue substream. When mixing music and effects (M+E), dialogue and associate substreams, the dialnorm of either the dialogue or the associate 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 LoRo.

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 (M+E) 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 (M+E) substream and the dialogue substream.

6.3.8.2 further_loudness_info - Additional loudness information

6.3.8.2.1 b_rtllcomp

This flag indicates whether realtime loudness correction data is present.

6.3.8.2.2 rtll_comp

This field indicates the realtime 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 bit 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 a Low-Frequency Effects channel.

Table 119: Pseudocode

```
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.

Table 120: Pseudocode

```
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.

Table 121: Pseudocode

```
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 Surround and a Right Surround channel.

Table 122: Pseudocode

```
channel_mode_contains_LsRs()
{
    if (ch_mode >= 3 && ch_mode <= 15) {
         return TRUE;
    }
    return FALSE;
}
```

6.3.8.4.5 channel_mode_contains_LrsRrs()

This function returns true if the channel mode contains a Left Rear Surround and a Right Rear Surround channel.

Table 123: Pseudocode

```
channel_mode_contains_LrsRrs()
{
    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.

Table 124: Pseudocode

```
channel_mode_contains_LwRw()
{
    if (ch_mode == 7 || ch_mode == 8 || (ch_mode >= 13 && ch_mode <= 15)) {
         return TRUE;
    }
    return FALSE;
}
```

6.3.8.4.7 channel_mode_contains_VhlVhr()

This function returns true if the channel mode contains a Left Vertical Height and a Right Vertical Height channel.

Table 125: Pseudocode

```
channel_mode_contains_VhlVhr()
{
    if (ch_mode == 9 || ch_mode == 10) {
         return TRUE;
    }
    return FALSE;
}
```

6.3.8.5 pan_signal_selector

This 2-bit field shall be evaluated if all of the following conditions are met:

- `channel_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

If all of the above conditions are met and the value of `pan_signal_selector` is larger than 0, decoding of `3_0_channel_element` and processing (as described in ETSI TS 103 190-1 [1], clause 5.3.3.3) can be simplified as follows:

- partial decode of `three_channel_data` as indicated in Table 126
- omission of the processing as described in ETSI TS 103 190-1 [1], clause 5.3.3.3
- panning of the two decoded signals using `pan_dialog[0]` and `pan_dialog[1]` as described in clause 5.10.2.3

Table 126: pan_signal_selector

pan_signal_selector	semantics
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

Object audio metadata (OAMD) comprise parameters that describe 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, which are common to all objects in the corresponding substream group.

6.3.9.2.2 b_default_screen_size_ratio

The `b_default_screen_size_ratio` flag 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, in the mastering room, of the width of the screen to the distance between the L and R speakers. 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

The `b_bed_object_chan_distribute` flag indicates whether the channel distribution of bed objects is activated. If activated, one bed object can be distributed over multiple channels.

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* value is a helper variable which 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 127.

Table 127: 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 128.

Table 128: 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. `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 129.

Table 129: 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

The `b_use_ramp_table` flag indicates whether the *`ramp_duration`* value is indicated by the `ramp_duration_table` element or by the `ramp_duration` element according to Table 130.

Table 130: b_use_ramp_table

b_use_ramp_table	**Description**
0	*`ramp_duration`* in audio samples is indicated by `ramp_duration`.
1	*`ramp_duration`* in audio samples is indicated by `ramp_duration_table`.

6.3.9.3.11 ramp_duration_table

The `ramp_duration_table` element indicates the *`ramp_duration`* value as shown in Table 131.

Table 131: 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 according substream.

6.3.9.4.2 b_ducking_disabled

The b_ducking_disabled flag indicates whether the automatic ducking is disabled for the according object.

6.3.9.4.3 object_sound_category

The object_sound_category element defines the object sound category according to Table 132.

Table 132: 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

The b_keep flag indicates whether the values of the last object property update are still valid.

6.3.9.4.6 b_common_data

The b_common_data flag indicates whether the alternative properties set has common parameters for all present objects.

6.3.9.4.7 b_alt_gain

The b_alt_gain flag indicates whether the alt_gain element is present in the bitstream. If the alt_gain element is not present, the gain data contained in the object_basic_info element shall be used.

6.3.9.4.8 alt_gain

The alt_gain element indicates the alternative gain value, to be used as *object_gain*, as shown in Table 133.

Table 133: alt_gain

alt_gain	object_gain [dB]
0-62	$14 - \text{alt\_gain}$
63	$-\infty$

6.3.9.4.9 b_alt_position

The b_alt_position flag indicates whether the alternative properties set contains position data.

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:

$$\text{object\_position\_X} = \text{alt\_pos3D\_X}/62$$

NOTE: Clause 6.3.9.8.4.1 describes room anchored position data.

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:

$$\text{object\_position\_Y} = \text{alt\_pos3D\_Y}/62$$

NOTE: Clause 6.3.9.8.4.1 describes room anchored position data.

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:

$$\text{object\_position\_Z} = \text{sign} \times \text{alt\_pos3D\_Z}/15$$

NOTE: Clause 6.3.9.8.4.1 describes room anchored position data.

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 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: How to handle the updates in terms of timing is indicated via oamd_timing_data described in clause 6.3.9.3.

6.3.9.6.2 b_object_not_active

The `b_object_not_active` flag 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` is a helper variable that 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 134.

Table 134: 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

The `b_basic_info_reuse` flag 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` is a helper variable that indicates whether `object_render_info` is present, or default values shall be used for the according properties, or the values shall be determined from the previous `obj_info_block`, or if only some fields shall be reused from the previous `obj_info_block`. The possible values for `object_render_info_status` are shown in Table 135.

Table 135: 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` = 0 • `object_width` = 0 • `object_screen_factor` = 0 • `b_object_snap` = 0
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

The `b_render_info_reuse` flag indicates whether object properties shall be reused from the `obj_render_info` of the previous `obj_info_block`.

6.3.9.6.7 b_render_info_partial_reuse

The `b_render_info_partial_reuse` flag 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

The `b_add_table_data` flag 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

The `b_default_basic_info_md` flag indicates whether the object's basic info metadata shall be set to default values:

- *`object_gain`* defaults to 0 dB
- *`object_priority`* defaults to 1

NOTE: These default values differ from the default values for *`object_gain`* and *`object_priority`* defined in Table 134, 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 136 and indicates the basic info metadata coding.

Table 136: 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 137 and indicates the non-default value for the object gain.

Table 137: object_gain_code

object_gain_code	*object_gain* [dB]
0b0	specified by `object_gain_value`
0b10	$-\infty$
0b11	set to gain of previous object

6.3.9.7.5 object_gain_value

The `object_gain_value` indicates the value for the object gain as shown in Table 138.

Table 138: 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 bitstream 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 contains multiple object properties, i.e. the position properties.

6.3.9.8.2 obj_render_info_mask

The obj_render_info_mask is a bitmask, which describes which properties are present in the corresponding object_render_info. Each bit, when set to 1, indicates the presence of properties of a specific group as specified in Table 139. If the bit is set to 0, the properties of the corresponding group are not present and the properties of the previous obj_info_block shall be reused. If object_render_info_status is set to ALL_NEW, obj_render_info_mask is not present in the bitstream.

Table 139: obj_render_info_mask

obj_render_info_mask	Description
0b001	Group of position properties present
0b010	Group of zone properties present
0b100	Group of other properties present

6.3.9.8.3 b_diff_pos_coding

The b_diff_pos_coding flag 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 consists of x, y and z coordinates which are 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 value is coded differentially. 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 12. 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.

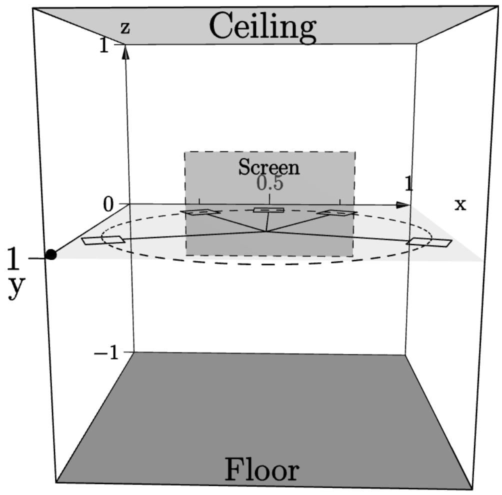

Figure 12: Object position coordinate system

- 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 *object_position_X_prev*. The current value is given by:

$$\text{object\_position\_X} = \text{object\_position\_X\_prev} + \text{diff\_pos3D\_X}/62$$

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:

$$\text{object\_position\_Y} = \text{object\_position\_Y\_prev} + \text{diff\_pos3D\_Y}/62$$

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:

$$\text{object\_position\_Z} = \text{object\_position\_Z\_prev} + \text{diff\_pos3D\_Z}/15$$

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:

$$\text{object\_position\_X} = \text{pos3D\_X}/62$$

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:

$$\text{object\_position\_Y} = \text{pos3D\_Y}/62$$

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:

$$\text{object\_position\_Z} = \text{sign} \times \text{pos3D\_Z}/15$$

6.3.9.8.5 b_grouped_zone_defaults

The `b_grouped_zone_defaults` flag indicates whether the properties of the zone group shall be set to default values. Otherwise, `group_zone_mask` and `zone_mask` are present in the bitstream.

6.3.9.8.6 group_zone_mask

The `group_zone_mask` is a bitmask used for assignment of properties of the zone group as shown in Table 140.

Table 140: group_zone_mask

group_zone_mask	Description
0b001	`zone_mask` is present
0b010	`b_enable_elevation` is 0
0b100	`b_object_snap` is 1

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 141.

Table 141: 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 - 7	Reserved

6.3.9.8.8 b_enable_elevation

The `b_enable_elevation` flag is implicitly signalled via `group_zone_mask` and indicates whether rendering to height (top) speakers is enabled. By default `b_enable_elevation` should be 1.

6.3.9.8.9 b_object_snap

The b_object_snap flag is implicitly signalled via group_zone_mask and 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 0.

6.3.9.8.10 b_grouped_other_defaults

The b_grouped_other_defaults flag 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 which indicates which properties of the other properties group are present in the bitstream as shown in Table 142.

Table 142: 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 element is one bit which indicates the mode of the object spreading according to Table 143. Increasing the object width may increase the number of speakers used to playback a particular object.

Table 143: object_width_mode

object_width_mode	Description
0	1-dimensional object spreading (radial)
1	3-dimensional object spreading

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*, applied to the X and Z dimensions for objects which 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, the object_screen_factor value shall default to 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 144.

Table 144: 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

The b_obj_at_infinity flag 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 145.

Table 145: 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 the signalling of the *object_divergence* according to Table 146.

Table 146: object_div_mode

object_div_mode	Description
0b00	Normal left/right divergence is used with a value given by object_div_table.
0b01	*object_divergence* value (and type) is reused from the previous obj_info_block.
0b10	Normal left/right divergence is used with a value given by object_div_code.
0b11	Reserved

6.3.9.8.22 object_div_table

The object_div_table element indicates the *object_divergence* (*div*) according to Table 147. An object at position (x,y,z) and $\mathrm{div} > 0$ is equivalent to two virtual objects at positions $\left(x - \frac{\mathrm{div}}{2}, y, z\right)$ and $\left(x + \frac{\mathrm{div}}{2}, y, z\right)$ with half the energy each.

Table 147: 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 ($\mathrm{div} = 0$) to the L and R speakers ($\mathrm{div} = 1$) and the effective maximum azimuth angle is +/- 45 degrees.

6.3.9.8.23 object_div_code

The object_div_code indicates the *object_divergence* (*div*) according to Table 148. An object at position (x,y,z) and $\mathrm{div} > 0$ is equivalent to two virtual objects at positions $\left(x - \frac{\mathrm{div}}{2}, y, z\right)$ and $\left(x + \frac{\mathrm{div}}{2}, y, z\right)$ with half the energy each.

Table 148: 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 L and R speakers (div $= 1$) and the effective maximum azimuth angle is +/- 45 degrees.

6.3.10 Presentation data

6.3.10.1 loud_corr - Loudness correction

6.3.10.1.1 b_obj_loud_corr

This flag indicates whether loudness correction data for objects is present.

6.3.10.1.2 b_corr_for_immersive_out

This flag indicates whether loudness correction data for immersive output channel configurations is present.

6.3.10.1.3 b_loro_loud_comp

This flag indicates whether LoRo downmix loudness correction data is present.

6.3.10.1.4 b_ltrt_loud_comp

This flag indicates whether LtRt downmix loudness correction data is present.

6.3.10.1.5 b_loud_comp

This flag 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 ∈ {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, and if present 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 149.

Table 149: bs_ch_config

bs_ch_config	Description
0	5/4/4 or 4/4/4
1	3/4/4 or 2/4/4
2	3/2/4 or 2/4/4
3	5/4/2 or 4/4/2
4	3/4/2 or 2/4/2
5	3/2/2 or 2/2/2

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 150.

Table 150: out_ch_config

out_ch_config	Description
0	3/2/0
1	3/2/2
2	3/2/4
3	3/4/0
4	3/4/2
5, 6, 7	unused

6.3.10.2.5 b_stereo_dmx_coeff

This flag indicates whether stereo downmix coefficients are present.

6.3.10.3 Custom downmix parameter

6.3.10.3.1 gain_f1_code

The `gain_f1_code` element indicates a *gain_f1* value according to Table 151.

Table 151: 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 `gain_f2_code`, `gain_b_code`, `gain_t1_code`, `gain_t2a_code`, `gain_t2b_code`, `gain_t2c_code`, `gain_t2d_code`, `gain_t2e_code` or `gain_t2f_code` elements indicate a corresponding *gain_f2*, *gain_b*, *gain_t1*, *gain_t2a*, *gain_t2b*, *gain_t2c*, *gain_t2d*, *gain_t2e* or *gain_t2f* value according to Table 152.

Table 152: 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	-7,5
6	-9,0
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 flag 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 flag 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 flag 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 flag 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 flag indicates whether the Tl and Tr signals are mixed into the front channels L and R.

6.3.10.3.9 b_top_to_side

This flag indicates whether the Tl and Tr signals are mixed into the side channels Ls and Rs.

6.3.10.3.10 Default custom downmix parameter

In case that 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 153.

Table 153: Default custom downmix parameter

Custom downmix parameter	Default value
b_put_screen_to_c	0
b_top_front_to_front	0
b_top_front_to_side	1
b_top_back_to_front	0
b_top_back_to_side	1
b_top_to_front	0
b_top_to_side	1
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 tracks into speaker feeds.

Table A.25 and Table A.26 list the identifiers that the coefficient matrix appears under in file ts_103190_tables_part2.c contained in archive ts_10319002v010101p0.zip 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 describes AC-4 channel layouts together with their contained speaker locations and speaker group indices.

Table A.27: Speaker layouts and speaker indices

Speaker index	Speaker name	Channel layouts									Speaker group index
		5.X	**7.Xa**	**7.Xb**	**7.Xc**	**7.X.2**	**7.X.4**	**9.X.2**	**9.X.4**	**22.2**	
0	L	Y	Y	Y	Y	Y	Y	Y	Y	Y	0
1	R	Y	Y	Y	Y	Y	Y	Y	Y	Y	
2	C	Y	Y	Y	Y	S	S	S	S	Y	1
3	Ls	Y	Y	Y	Y	Y	Y	Y	Y	Y	2
4	Rs	Y	Y	Y	Y	Y	Y	Y	Y	Y	
5	Lb			Y		S	S	S	S	Y	3
6	Rb			Y		S	S	S	S	Y	
7	Tfl						S		S	Y	4
8	Tfr						S		S	Y	
9	Tbl						S		S	Y	5
10	Tbr						S		S	Y	
11	LFE	X	X	X	X	X	X	X	X	Y	6
12	TL					S		S			7
13	TR					S		S			
14	Tsl									Y	8
15	Tsr									Y	
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		16
25	Rscr							Y	Y		
26	Lw		Y							Y	17
27	Rw		Y							Y	
28	Vhl				Y						18
29	Vhr				Y						
NOTE: Y = Speaker present in layout; S = Speaker present in layout, but can be signalled as silent; X = Speaker present in layout only if channel configuration includes LFE.											

Table A.28 through Table A.42 describe the possible channel configurations with their channel names. All references to ITU channel names are with regard to Recommendation ITU-R BS.2051-0 [4].

Table A.28: 2.0 Channel configuration

Channel name	Abbreviation
Left	L
Right	R

Table A.29: 5.1 Channel configuration (3/2/0; ITU 0+5+0)

Channel name	Abbreviation
Left	L
Right	R
Centre	C
Left Surround	Ls
Right Surround	Rs
Low-Frequency Effects	LFE

Table A.30: 7.1 Channel configuration (3/4/0)

Channel name	Abbreviation
Left	L
Right	R
Centre	C
Left Surround	Ls
Right Surround	Rs
Left Rear Surround	Lrs
Right Rear Surround	Rrs
Low-Frequency Effects	LFE

Table A.31: 5/4/4 Immersive Audio channel configuration (9.1.4; based on ITU 4+9+0)

Channel name	Abbreviation
Left	L
Right	R
Centre	C
Left Screen	Lscr
Right Screen	Rscr
Left Surround	Ls
Right Surround	Rs
Left Back	Lb
Right Back	Rb
Top Front Left	Tfl
Top Front Right	Tfr
Top Back Left	Tbl
Top Back Right	Tbr
Low-Frequency Effects	LFE

Table A.32: 5/4/2 Immersive Audio channel configuration (9.1.2; based on ITU 4+9+0)

Channel name	Abbreviation
Left	L
Right	R
Centre	C
Left Screen	Lscr
Right Screen	Rscr
Left Surround	Ls
Right Surround	Rs
Left Back	Lb
Right Back	Rb
Top Left	Tl
Top Right	Tr
Low-Frequency Effects	LFE

Table A.33: 5/4/0 Immersive Audio channel configuration (9.1.0; based on ITU 4+9+0)

Channel name	Abbreviation
Left	L
Right	R
Centre	C
Left Screen	Lscr
Right Screen	Rscr
Left Surround	Ls
Right Surround	Rs
Left Back	Lb
Right Back	Rb
Low-Frequency Effects	LFE

Table A.34: 5/2/4 Immersive Audio channel configuration (7.1.4; based on ITU 4+9+0)

Channel name	Abbreviation
Left	L
Right	R
Centre	C
Left Screen	Lscr
Right Screen	Rscr
Left Surround	Ls
Right Surround	Rs
Top Front Left	Tfl
Top Front Right	Tfr
Top Back Left	Tbl
Top Back Right	Tbr
Low-Frequency Effects	LFE

Table A.35: 5/2/2 Immersive Audio channel configuration (7.1.2; based on ITU 4+9+0)

Channel name	Abbreviation
Left	L
Right	R
Centre	C
Left Screen	Lscr
Right Screen	Rscr
Left Surround	Ls
Right Surround	Rs
Top Left	Tl
Top Right	Tr
Low-Frequency Effects	LFE

Table A.36: 5/2/0 Immersive Audio channel configuration (7.1.0; based on ITU 4+9+0)

Channel name	Abbreviation
Left	L
Right	R
Centre	C
Left Screen	Lscr
Right Screen	Rscr
Left Surround	Ls
Right Surround	Rs
Low-Frequency Effects	LFE

Table A.37: 3/4/4 Immersive Audio channel configuration (7.1.4, based on ITU 3+7+0)

Channel name	Abbreviation
Left	L
Right	R
Centre	C
Left Surround	Ls
Right Surround	Rs
Left Back	Lb
Right Back	Rb
Top Front Left	Tfl
Top Front Right	Tfr
Top Back Left	Tbl
Top Back Right	Tbr
Low-Frequency Effects	LFE

Table A.38: 3/4/2 Immersive Audio channel configuration (7.1.2, based on ITU 3+7+0)

Channel name	Abbreviation
Left	L
Right	R
Centre	C
Left Surround	Ls
Right Surround	Rs
Left Back	Lb
Right Back	Rb
Top Left	Tl
Top Right	Tr
Low-Frequency Effects	LFE

Table A.39: 3/4/0 Immersive Audio channel configuration (7.1.0, based on ITU 3+7+0)

Channel name	Abbreviation
Left	L
Right	R
Centre	C
Left Surround	Ls
Right Surround	Rs
Left Back	Lb
Right Back	Rb
Low-Frequency Effects	LFE

Table A.40: 3/2/4 Immersive Audio channel configuration (5.1.4, based on ITU 3+7+0)

Channel name	Abbreviation
Left	L
Right	R
Centre	C
Left Surround	Ls
Right Surround	Rs
Top Front Left	Tfl
Top Front Right	Tfr
Top Back Left	Tbl
Top Back Right	Tbr
Low-Frequency Effects	LFE

Table A.41: 3/2/2 Immersive Audio channel configuration (5.1.2; based on ITU 3+7+0)

Channel name	Abbreviation
Left	L
Right	R
Centre	C
Left Surround	Ls
Right Surround	Rs
Top Left	Tl
Top Right	Tr
Low-Frequency Effects	LFE

Table A.42: 22.2 Immersive Audio channel configuration (ITU 9+10+3)

Channel name	Abbreviation	ITU R BS.2051 name
Left	L	FLc
Right	R	FRc
Centre	C	FC
Low-Frequency Effects	LFE	LFE1
Left Surround	Ls	SiL
Right Surround	Rs	SiR
Left Back	Lb	BL
Right Back	Rb	BR
Top Front Left	Tfl	TpFL
Top Front Right	Tfr	TpFR
Top Back Left	Tbl	TpBL
Top Back Right	Tbr	TpBR
Top Side Left	Tsl	TpSiL
Top Side Right	Tsr	TpSiR
Top Front Centre	Tfc	TpFC
Top Back Centre	Tbc	TpBC
Top Centre	Tc	TpC
Low-Frequency Effects 2	LFE2	LFE2
Bottom Front Left	Bfl	BtFL
Bottom Front Right	Bfr	BtFR
Bottom Front Centre	Bfc	BtFC
Back Centre	Cb	BC
Left Wide	Lw	FL
Right Wide	Rw	FR

A.4 Speaker Zones

Table A.43: Speaker zones

Speaker		ITU configuration				
ITU Label	**α [°]**	**A (0+2+0)**	**B (0+5+0)**	**F (3+7+0)**	**G (4+9+0)**	**H (9+10+3)**
M+000	0		Sc	Sc	Sc	Sc
M+022	22.5					
M+SC	(note 1)				Sc	
M+030	30	Sc, Su	Sc	Sc	Sc	Sc
M+045	45					
M+060	60					Sc
M+090	90			Si, Su	Si, Su	Si, Su
M+110	110		B, Su			
M+135	135			B, Su	B, Su	B, Su
M+180	180					B, Su
NOTE 1: M+SC is located at the right and left edges of the display. NOTE 2: B = back zone, Si = Side zone, Su = surround zone, Sc = Screen zone. The centre-back zone is a union of the back zone with the centre speaker.						

Table A.43 specifies an assignment for the horizontal 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 only the right speaker is listed; the left speaker belongs to the same zone as the corresponding right speaker.

From the listed configuration, further configurations can be derived. ITU configuration C through E are equivalent to B since only height speakers are added. A "7.1" configuration can be treated as configuration F less the height speakers. Layouts without centre speaker can be derived by substituting the centre speaker by L and R.

The stereo configuration effectively defeats zone constraints; the back and side zones are empty, while all other zones contain the two only speakers.

Annex B (informative): AC-4 bitrate calculation

For some systems, knowledge of the approximate audio bitrate is required.

For CBR streams (signalled by a `wait_frames` value of 0) the bitrate 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 bitrate.)

For ABR streams, the following steps outline an algorithm that provides a high precision estimate of the bitrate.

1) Define an *admissible sequence* of length L as a sequence of l consecutive frames where all of the following conditions are fulfiled for $1 \leq i < L$:

   - $\texttt{frame\_rate\_index}_i \equiv$ constant (the framerate 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 $\texttt{p}$ 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 $\text{framerate}_{\text{ac4}}$ 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 $\text{N}' = m + \text{wait\_frames}_0 - \text{wait\_frames}_m \geq 3$.

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: $\text{S}_{\text{Tot}} = 8 \times \sum_{i=1}^{m} \text{S}_i$ [bits].

8) Calculate $\text{br}_{\text{min}} = \frac{\text{S}_{\text{Tot}}}{\text{N}'+1} \times \text{framerate}_{\text{ac4}}$[bps] and $\text{br}_{\text{max}} = \frac{\text{S}_{\text{Tot}}}{\text{N}'-1} \times \text{framerate}_{\text{ac4}}$[bps].

9) Calculate $K_1 = \left\lfloor log_2\left(\frac{\text{br}_{\text{min}}}{1000[\text{bps}]}\right)\right\rfloor$ and $K_2 = K_1 + 1$.

10) Calculate four bit-rate estimates: $\text{br}_i^- = F^- \times 2^{K_i}[\text{kbps}], i \in \{1; 2\}$ and $\text{br}_i^+ = F^+ \times 2^{K_i}[\text{kbps}], i \in \{1; 2\}$.

11) Find the index opt for which $\text{br}_{\text{opt}}^- < \text{br}_{\text{max}} \wedge \text{br}_{\text{opt}}^+ \geq \text{br}_{\text{min}}$.

12) Calculate $R_{\text{min}} = max\left(\text{br}_{\text{opt}}^-, \text{br}_{\text{min}}\right)[\text{kbps}]$ and $R_{\text{max}} = min\left(\text{br}_{\text{opt}}^+, \text{br}_{\text{max}}\right)[\text{kbps}]$.

13) Determine the framing overhead bitrate B in kbps. If the `ac4_syncframe` format is used, then $B = \text{OH}_{\text{syncF}} \times \text{framerate}_{\text{ac4}} \times \frac{8}{1000}$[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 $[\text{R}_{\text{min}} + B, \text{R}_{\text{max}} + B]$[kbps].

Annex C (normative): AC-4 Sync Frame

C.1 Introduction

The AC-4 sync frame is an optional bitstream layer for encapsulating AC-4 raw frames. It can be used when the transmission system is not frame based, i.e. when it does not include information about the framing. In this case the AC-4 sync frame offers a simple way to reconstruct the framing at the receiver side by a high level parsing of the bitstream.

The `sync_word` can be either 0xAC40 or 0xAC41. If the `sync_word` is 0xAC41, a `crc_word` is also transmitted.

The `crc_word` is a 16 bit CRC based on the `frame_size` element and the `raw_ac4_frame` element (the `sync_word` element is not part of the CRC calculation). The following generator polynomial is used to generate each of the 16-bit CRC words: $x^{16} + x^{15} + x^2 + 1$. The polynomial is also commonly known as IBM-CRC-16.

C.2 ac4_syncframe

Syntax	No of bits
ac4_syncframe()	
{	
sync_word;	16
frame_size();	
raw_ac4_frame();	
if (sync_word == 0xAC41) {	
crc_word;	16
}	
}	

C.3 frame_size

Syntax	No of bits
frame_size()	
{	
frame_size;	16
if (frame_size == 0xffff) {	
frame_size;	24
}	
}	

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 PES packetisation 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)
- Input buffer size for decoders (clause D.2.4)

D.2 Constraints 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 Annex C.

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 PES packet payload. This means that 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 PMT descriptor loop following the `ES_info_length` field. The signalling may use an `registration_descriptor` and/or an AC-4 specific descriptor and can be defined by application standards. It is recommended that the last bytes of the descriptor are 0x41 0x43 0x2D 0x34 (`"AC-4"`) to provide a fallback signalling option for receivers which do not natively understand the respective descriptor.
- 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 and the 3GPP file format.

This annex additionally covers:

- the steps required to properly packetize an AC-4 bitstream for multiplexing and storage in an MPEG-DASH-compliant ISO base media file format file;
- the steps required to demultiplex an AC-4 bitstream from an MPEG DASH-compliant ISO base media file format file.

E.2 AC-4 Track definition

In the terminology of the ISO Base Media File Format (ISOBMFF) 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.4.1.

The value of the `timescale` parameter in the Media Header Box, 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 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. 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	240 000	8 008
			48 000	1 601, 1 602
	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		

base_samp_freq [kHz]	frame_rate_index	*frame_rate* [fps]	Media time scale [1/sec]	sample_delta [units of media time scale]
44,1	0...12	reserved		
	13	(21,53)	44 100	2 048
	14, 15	reserved		
NOTE: For 29,97 fps (frame_rate_index = 3), there are two possible choices for the media time scale. For a media time scale of 48 000, the sample_delta value is non-constant and changes between the two specified values.				

E.3 AC-4 Sample definition

For the purpose of carrying AC-4 in ISOBMFF, 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`.

E.4 AC4SampleEntry Box

E.4.1 AC4SampleEntry Box

The box type of the AC4SampleEntry Box shall be 'ac-4'.

The AC4SampleEntry Box is defined by clause E.4.1.

Syntax	No of bits
AC4SampleEntry()	
{	
BoxHeader.Size;	32
BoxHeader.Type;	32
Reserved[6];	8
DataReferenceIndex;	16
Reserved[2];	32
ChannelCount;	16
SampleSize;	16
Reserved;	32
SamplingFrequency;	16
Reserved;	16
Ac4SpecificBox();	
}	

The layout of the `AC4SampleEntry` box is identical to that of `AudioSampleEntry` defined in ISO/IEC 14496-12 [3] (including the reserved fields and their values), except that `AC4SampleEntry` ends with a box containing AC-4 bitstream information called `AC4SpecificBox`. The `AC4SpecificBox` field structure for AC-4 is defined in clause E.5.1.

E.4.2 BoxHeader.Size

This field indicates the size of the box according to the `sampleEntry` definition, as specified by the ISO Base Media File Format (ISOBMFF) specification [3].

E.4.3 BoxHeader.Type

This field shall be set to the value 'ac-4'.

E.4.4 DataReferenceIndex

This field shall be set according to the `sampleEntry` definition as specified by the ISO Base Media File Format (ISOBMFF) specification [3].

E.4.5 ChannelCount

This field should be set to the total number of audio output channels of the default presentation of that track, if not defined differently by an application standard. The value of this field shall be ignored on decoding.

E.4.6 SampleSize

This field shall be set to a value of '16'.

E.4.7 SamplingFrequency

The value of this field shall be ignored on decoding.

E.5 AC4SpecificBox

E.5.1 AC4SpecificBox

The AC4SpecificBox is defined clause E.5.1.

Syntax	No of bits
AC4SpecificBox()	
{	
BoxHeader.Size;	32
BoxHeader.Type;	32
ac4_dsi_v1();	
}	

E.5.2 BoxHeader.Size

This field indicates the size of the box as specified by the ISO Base Media File Format (ISOBMFF) specification [3].

E.5.3 BoxHeader.Type

This field shall contain the value 'dac4'.

E.6 ac4_dsi_v1

E.6.1 ac4_dsi_v1

This element is specified in clause E.6.1.

The information from the ac4_dsi_v1 shall not be used to configure the AC-4 decoder. The AC-4 decoder shall obtain its configuration only from the ac4_toc that is part of every sample.

All elements in the ac4_dsi_v1 are valid across every sample referenced by the sample entry containing this ac4_dsi_v1.

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

Syntax	No of bits

```
      }
    }
  }
  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 = presentation_v0_dsi();
    }
    else {
      if (presentation_version == 1) {
        presentation_bytes = presentation_v1_dsi();
      }
      else {
        presentation_bytes = 0;
      }
    }
    skip_bytes = pres_bytes - presentation_bytes;
    skip_area; ....................................................... skip_bytes*8
  }
}
```

NOTE: All entries in the `ac4_dsi_v1()` reflect the content of all samples referenced by the `Ac4SampleEntry` containing the DSI.

E.6.2 ac4_dsi_version

This field indicates the version of the DSI. For a DSI that conforms to the present document, the `ac4_dsi_version` field shall be set to '001'. Decoders conforming to the present document shall discontinue parsing and skip the rest of the box in case the `ac4_dsi_version` field is set to a value larger than 1.

E.6.3 bitstream_version

This field shall contain the bitstream version as described in clause 6.3.2.1.1. Its value shall be the same as read from the `ac4_toc`.

E.6.4 fs_index

This field shall contain the sampling frequency index as described in ETSI TS 103 190-1 [1], clause 4.3.3.2.5. Its value shall be the same as read from the `ac4_toc`.

E.6.5 frame_rate_index

This field shall contain the frame rate index as described in ETSI TS 103 190-1 [1], clause 4.3.3.2.6. Its value shall be the same as read from the `ac4_toc`.

E.6.6 n_presentations

This field shall contain the number of presentations contained in the corresponding AC-4 frame. Its value shall be the same as read from the `ac4_toc`.

E.6.7 short_program_id

This field shall contain the short program ID. Its value shall be the same as read from the `ac4_toc`.

E.6.8 program_uuid

This field shall contain the program identification. Its value shall be the same as read from the `ac4_toc`.

E.6.9 presentation_version

This field shall contain the presentation version as described in ETSI TS 103 190-1 [1], clause 4.3.3.4. Its value shall be the same as the value read from the respective `ac4_presentation_info` or `ac4_presentation_v2_info`.

E.6.10 pres_bytes

This field shall contain the length of the following per-presentation information section.

E.7 ac4_bitrate_dsi

E.7.1 ac4_bitrate_dsi

This clause provides information on the bitrate of an AC-4 stream or individual presentations of an AC-4 stream.

Syntax	No of bits
`ac4_bitrate_dsi()`	
`{`	
bit_rate_mode;	2
bit_rate;	32
bit_rate_precision;	32
`}`	

E.7.2 bit_rate_mode

This field describes the bitrate control according to Table E.2.

Table E.2: bit_rate_mode values

bit_rate_mode	semantics
0	bit rate mode not specified
1	constant bit rate
2	average bit rate
3	variable bit rate

E.7.3 bit_rate

This unsigned 32-bit integer number shall contain the bit rate in bits/second. The value 0 means that the bit rate is unknown.

E.7.4 bit_rate_precision

This unsigned 32-bit integer number shall contain the bit rate precision in bits/second. A value of `bit_rate_precision` of *X* means that the bit rate is in the range [bit rate - *X*, bit rate + *X*]. The value 0xFFFFFFFF means that the bit rate precision is unknown.

E.8 ac4_presentation_v0_dsi

E.8.1 ac4_presentation_v0_dsi

Syntax	No of bits
ac4_presentation_v0_dsi()	
{	
presentation_config;	5
if (presentation_config == 6) {	
b_add_emdf_substreams = 1;	
}	
else {	
mdcompat;	3
b_presentation_group_index;	1
if (b_presentation_group_index) {	
presentation_group_index;	5
}	
dsi_frame_rate_multiply_info;	2
presentation_emdf_version;	5
presentation_key_id;	10
presentation_channel_mask;	24
if (b_single_substream == 1) {	
ac4_substream_dsi();	
}	
else {	
b_hsf_ext;	1
if (presentation_config in [0, 1, 2]) {	
ac4_substream_dsi();	
ac4_substream_dsi();	
}	
if (presentation_config in [3, 4]) {	
ac4_substream_dsi();	
ac4_substream_dsi();	
ac4_substream_dsi();	
}	
if (presentation_config == 5) {	
ac4_substream_dsi();	
}	
if (presentation_config > 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
}	
}	
byte_align;	0…7
}	

NOTE: The number of bits in `byte_align` pads the number of bits, counted from the start of `ac4_presentation_v0_dsi`, to an integer number of bytes.

E.8.2 presentation_config

If the `b_single_substream` element described in ETSI TS 103 190-1 [1], clause 4.3.3.3.1 is set to 0, this field shall contain the value of `presentation config` read from the respective `ac4_presentation_info` as described in ETSI TS 103 190-1 [1], clause 4.3.3.3.4. If the `b_single_substream` element is set to 1, the presentation_config element value shall be set to 0x1F.

E.8.3 mdcompat

This field contains the decoder compatibility indication as described in ETSI TS 103 190-1 [1], clause 4.3.3.3.8. Its value shall be the same as the respective value read from the respective `ac4_presentation_info`.

E.8.4 b_presentation_group_index

This bit indicates that the containing presentation belongs to a group of presentations. Its value shall be the same as the respective value from the respective `ac4_presentation_info`.

E.8.5 presentation_group_index

This field shall contain a presentation group index. Its value shall be the same as the respective value read from the respective `ac4_presentation_info`.

E.8.6 dsi_frame_rate_multiply_info

This field shall signal the `frame_rate_multiply_info` as described in ETSI TS 103 190-1 [1], clause 4.3.3.5. Its value shall correspond to the value read from the respective `ac4_presentation_info` as shown in Table E.3.

Table E.3: Determining dsi_frame_rate_multiply_info

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

E.8.7 presentation_emdf_version

This field shall contain the EMDF syntax version as described in ETSI TS 103 190-1 [1], clause 4.3.3.6.1. Its value shall be the same as the respective `emdf_version` value read from the `emdf_info` field in the respective `ac4_presentation_info`.

E.8.8 presentation_key_id

This field shall contain the authentication ID as described in ETSI TS 103 190-1 [1], clause 4.3.3.6.2. Its value shall be the same as the respective `key_id` value read from the `emdf_info` field in the respective `ac4_presentation_info`.

E.8.9 presentation_channel_mask

This bit mask shall indicate the presence of channels in the presentation. Bit 23 (the most significant bit) shall be set to false. Bits 18...0 indicate the presence of speaker groups identified by the speaker group index described in Table A.27. Bits 22...19 are reserved.

E.8.10 b_hsf_ext

This bit shall indicate the availability of spectral data for high sampling frequencies as described in ETSI TS 103 190-1 [1], clause 4.3.3.3.3. Its value shall be the same as the respective value read from the respective `ac4_presentation_info`.

E.8.11 n_skip_bytes

This field indicates a number of subsequent bytes to skip.

E.8.12 skip_data

This field holds additional data and is reserved for future use.

E.8.13 b_pre_virtualized

This bit indicates pre-rendering as described in ETSI TS 103 190-1 [1], clause 4.3.3.3.5. Its value shall be the same as the respective value read from the respective `ac4_presentation_info`.

E.8.14 b_add_emdf_substreams

This bit indicates presence of additional EMDF containers as described in ETSI TS 103 190-1 [1], clause 4.3.3.3.6. Its value shall be the same as the respective value read from the respective `ac4_presentation_info`.

E.8.15 n_add_emdf_substreams

This field indicates the number of additional EMDF containers as described in ETSI TS 103 190-1 [1], clause 4.3.3.3.7. Its value shall be the same as the respective value read from the respective `ac4_presentation_info`.

E.8.16 substream_emdf_version

This field shall contain the EMDF syntax version as described in ETSI TS 103 190-1 [1], clause 4.3.3.6.1. Its value shall be the same as the respective `emdf_version` value read from the `emdf_info` field in the respective `n_add_emdf_substreams()` loop in the respective `ac4_presentation_info`.

E.8.17 substream_key_id

This field shall contain the authentication ID as described in ETSI TS 103 190-1 [1], clause 4.3.3.6.2. Its value shall be the same as the respective `key_id` value read from the `emdf_info` field in the respective `n_add_emdf_substreams()` loop in the `respective ac4_presentation_info`.

E.8.18 byte_align

These bits are used for the byte alignment of each presentation within the `ac4_dsi element`. Byte alignment is defined relative to the start of the enclosing syntactic element.

E.9 ac4_substream_dsi

E.9.1 ac4_substream_dsi

Syntax	No of bits

```
ac4_substream_dsi()
{
  channel_mode; ..................................................... 5
  dsi_sf_multiplier; ................................................ 2
  b_substream_bitrate_indicator; .................................... 1
  if (b_substream_bitrate_indicator) {
    substream_bitrate_indicator; .................................... 5
  }
  if (ch_mode in [7, 8, 9, 10]) {
    add_ch_base; .................................................... 1
  }
  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.9.2 channel_mode

This field shall contain the channel mode as described in ETSI TS 103 190-1 [1], clause 4.3.3.7.1. Its value shall correspond to the value read from the respective `ac4_substream_info` in the `ac4_presentation_info` and is expressed either through the *ch_mode* parameter from Table 79 (if the `channel_mode` bitfield is not 0b111111111) or through the value "12 + `variable_bits(2)`" (if the `channel_mode` bitfield is 0b111111111).

E.9.3 dsi_sf_multiplier

This field shall signal the `sf_multiplier` as described in ETSI TS 103 190-1 [1], clause 4.3.3.7.3. Its value shall correspond to the value read from the respective `ac4_substream_info` in the `ac4_presentation_info` as shown in Table E.4.

Table E.4: Determining dsi_sf_multiplier

sampling frequency	b_sf_multiplier	sf_multiplier	dsi_sf_multiplier
48 kHz	0	X	00
96 kHz	1	0	01
192 kHz		1	10

E.9.4 b_substream_bitrate_indicator

This bit indicates presence of the bitrate indicator as described in ETSI TS 103 190-1 [1], clause 4.3.3.7.5.

E.9.5 substream_bitrate_indicator

This field shall contain a bitrate indication as described in ETSI TS 103 190-1 [1], clause 4.3.3.7.5. The value shall correspond to the value read from the respective `ac4_substream_info` in the `ac4_presentation_info` and is expressed through the `brate_ind` parameter from ETSI TS 103 190-1 [1], table 89.

E.9.6 add_ch_base

This bit shall contain the Additional Channels Coupling base as described in ETSI TS 103 190-1 [1], clause 4.3.3.7.6. This field is present only if the *ch_mode* value according to ETSI TS 103 190-1 [1], table 87 is in the range [7...10].

E.9.7 b_content_type

This bit indicates the presence of content_type information as described in ETSI TS 103 190-1 [1], clause 4.3.3.7.7.

E.9.8 content_classifier

This field shall contain the content classifier as described in ETSI TS 103 190-1 [1], clause 4.3.3.8.1. The value shall correspond to the value read from the respective content_type field in the ac4_substream_info in the ac4_presentation_info.

E.9.9 b_language_indicator

This bit indicates presence of programme language indication as described in ETSI TS 103 190-1 [1], clause 4.3.3.8.2.

E.9.10 n_language_tag_bytes

This field shall contain the number of subsequent language tags bytes as described in ETSI TS 103 190-1 [1], clause 4.3.3.8.6.

E.9.11 language_tag_bytes

The sequence of langage_tag_bytes shall contain a language tag as described in ETSI TS 103 190-1 [1], clause 4.3.3.8.7. For the respective ac4_substream_info in the respective ac4_presentation_info, these values shall correspond:

- to the values of the respective language_tag_bytes values in the content_type field of the respective ac4_substream_info in the respective ac4_presentation_info, if b_serialized_language_tag is false.
- to the concatenation of language_tag_chunk fields in the content_type field of the respective ac4_substream_info in the respective ac4_presentation_info from consecutive frames, if b_serialized_language_tag is true.

E.10 ac4_presentation_v1_dsi

E.10.1 ac4_presentation_v1_dsi

Syntax	No of bits
ac4_presentation_v1_dsi()	
{	
presentation_config_v1;	5
if (presentation_config_v1 == 0x06) {	
b_add_emdf_substreams = 1;	
}	
else {	
mdcompat;	3
b_presentation_group_index;	1
if (b_presentation_group_index) {	
presentation_group_index;	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 (presentation_channel_mode in [11, 12, 13, 14]) {	
pres_b_4_back_channels_present;	1
pres_top_channel_pairs;	2

```
Syntax                                                                No of bits
      }
      presentation_channel_mask_v1; ...................................... 24
    }
    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 (b_single_substream_group == 1) {
      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
}
```

NOTE: The number of bits in `byte_align` pads the number of bits, counted from the start of `ac4_presentation_v1_dsi`, to an integer number of bytes.

E.10.2 presentation_config_v1

If the `b_single_substream_group` element described in clause 6.2.1.3 is set to 0, this field shall contain the value read from the respective `ac4_presentation_v1_info`. If the `b_single_substream_group` element is set to 1, the `presentation_config_v1` element value shall be set to 0x1F.

E.10.3 mdcompat

This field contains the decoder compatibility indication as described in clause 6.3.2.2.3. Its value shall be the same as the respective value read from the respective `ac4_presentation_v1_info`.

E.10.4 b_presentation_group_index

This bit indicates that the containing presentation belongs to a group of presentations. Its value shall be the same as the respective value from the respective `ac4_presentation_v1_info`.

E.10.5 presentation_group_index

This field shall contain a presentation group index. Its value shall be the same as the respective value read from the respective `ac4_presentation_v1_info`.

E.10.6 dsi_frame_rate_multiply_info

This field shall signal the `frame_rate_multiply_info` as described in ETSI TS 103 190-1 [1], clause 4.3.3.5. Its value shall correspond to the respective value read from the respective `ac4_presentation_v1_info` as shown in Table E.5.

Table E.5: 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

E.10.7 dsi_frame_rate_fractions_info

This field shall signal the `frame_rate_fractions_info` as described in clause 6.3.2.4. Its value shall correspond to the respective value read from the respective `ac4_presentation_v1_info` as shown in Table E.6.

Table E.6: 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	0	X	00
	1	0	01
	1	1	10
5 ,6, 7, 8, 9	0	X	00
	1	X	01
0,1,2,3,4,13	X	X	00

E.10.8 presentation_emdf_version

This field shall contain the EMDF syntax version as described in ETSI TS 103 190-1 [1], clause 4.3.3.6.1. Its value shall be the same as the respective `emdf_version` value read from the `emdf_info` field in the respective `ac4_presentation_v1_info`.

E.10.9 presentation_key_id

This field shall contain the authentication ID as described in ETSI TS 103 190-1 [1], clause 4.3.3.6.2. Its value shall be the same as the respective `key_id` value read from the `emdf_info` field in the respective `ac4_presentation_v1_info`.

E.10.10 b_presentation_channel_coded

This field shall signal whether the presentation is channel coded. This bit shall be set to false if the determination of *pres_ch_mode* according to Table 93 yields the value -1.

E.10.11 dsi_presentation_ch_mode

This field shall signal the presentation channel mode. It shall equal the value of *pres_ch_mode* determined according to Table 93 for the case that this one is different from -1.

E.10.12 pres_b_4_back_channels_present

This field shall signal the presence of non-silent signals in 4 versus 2 back channels. Its value shall equal the value of *b_pres_4_back_channels_present* as described in clause 6.3.3.1.29.

E.10.13 pres_top_channel_pairs

This field shall signal the presence of a non-silent signal pairs in the 4 top channels. Its value shall equal the value of *pres_top_channel_pairs* as described in clause 6.3.3.1.30.

E.10.14 presentation_channel_mask_v1

This bit mask shall indicate the presence of channels in the presentation. Bit 23 (the most significant bit), if set to true, indicates that the presentation is object based and thus channel presence does not apply. In this case, the values of bit 22...0 do not have any meaning. If bit 23 is set to false, then bits 18...0 indicate the presence of speaker groups identified by the speaker group index described in Table A.27. Bits 22...19 are reserved.

E.10.15 b_presentation_core_differs

This field shall be set to true if *pres_ch_mode_core* according to Table 95 has a value that differs from *pres_ch_mode* according to Table 93, and to false otherwise.

E.10.16 b_presentation_core_channel_coded

This field shall signal whether the core decode of a presentation is channel coded. This bit shall set to false if the determination of `pres_ch_mode_core` according to Table 95 yields the value -1.

E.10.17 dsi_presentation_ch_mode_core

This field shall signal the channel mode of the core decode of a presentation. It shall equal the value of *pres_ch_mode_core* determined according to Table 95 for the case that this one is different from -1, subtracted by 3.

Table E.7: Determining dsi_presentation_ch_mode_core

Presentation core channel mode	pres_ch_mode_core	dsi_presentation_ch_mode_core
5.0	3	0
5.1	4	1
5.0.2 core	5	2
5.1.2 core	6	3

E.10.18 b_presentation_filter

This field shall signal whether presentation filter data is available for the presentation. Its value shall be the same as the value of b_presentation_filter in the respective ac4_presentation_v1_info.

E.10.19 b_enable_presentation

This field shall signal whether a presentation is enabled for playback. Its value shall be the same as the value of b_enable_presentation in the respective ac4_presentation_v1_info.

E.10.20 n_filter_bytes

This field shall contain the length of the following data field for filter data. Encoders according to the present document shall write the value 0.

E.10.21 filter_data

This field shall contain filter data. Decoders according to the present document shall parse the data but ignore its contents.

E.10.22 b_multi_pid

This field shall signal whether the presentation has the multi-PID property, i.e. might assume that the substreams of some substream groups might not be stored in the bitstream. Its value shall be the same as the respective value read from the b_multi_pid field in the respective ac4_presentation_v1_info.

E.10.23 n_substream_groups_minus2

This field shall contain the number of substream groups minus 2 for presentation_config_v1 = 5. The indicated number of substream groups shall be the same as the respective *n_substream_groups* value for presentation_config = 5 in the respective ac4_presentation_v1_info.

E.10.24 b_pre_virtualized

This bit indicates pre-rendering as described in ETSI TS 103 190-1 [1], clause 4.3.3.3.5. Its value shall be the same as the respective value read from the respective ac4_presentation_v1_info.

E.10.25 b_add_emdf_substreams

This bit indicates presence of additional EMDF containers as described in ETSI TS 103 190-1 [1], clause 4.3.3.3.6. Its value shall be the same as the respective value read from the respective ac4_presentation_v1_info.

E.10.26 substream_emdf_version

This field shall contain the EMDF syntax version as described in ETSI TS 103 190-1 [1], clause 4.3.3.6.1. Its value shall be the same as the respective emdf_version value read from the emdf_info field in the respective n_add_emdf_substreams() loop in the respective ac4_presentation_v1_info.

E.10.27 substream_key_id

This field shall contain the authentication ID as described in ETSI TS 103 190-1 [1], clause 4.3.3.6.2. Its value shall be the same as the respective key_id value read from the emdf_info field in the respective n_add_emdf_substreams() loop in the respective ac4_presentation_v1_info.

E.10.28 b_presentation_bitrate_info

This bit indicates the presence of an ac4_bitrate_dsi element.

E.11 ac4_substream_group_dsi

E.11.1 ac4_substream_group_dsi

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) {	
dsi_substream_channel_mask;	24
}	
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
}	
objects_assignment_mask;	4
}	
}	
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 b_substreams_present

This bit shall indicate that the substreams referenced in the substream group are stored inside the track. Its value shall be equal to the value of `b_substreams_present` in the respective `ac4_substream_group_info()`.

E.11.3 b_hsf_ext

This bit shall indicate the availability of spectral data for high sampling frequencies as described in ETSI TS 103 190-1 [1], clause 4.3.3.3.3. Its value shall be the same as the respective value read from the respective `ac4_substream_group_info()`.

E.11.4 b_channel_coded

This bit shall indicate that the substreams referenced in the substream group are channel coded. Its value shall be equal to the value of `b_channel_coded` in the respective `ac4_substream_group_info()`.

E.11.5 n_substreams

This field shall contain the number of audio substreams contained in the substream group. It shall be equal to the value of `n_lf_substreams` from the respective `ac4_substream_group_info()`.

E.11.6 dsi_sf_multiplier

This field shall signal the `sf_multiplier` as described in ETSI TS 103 190-1 [1], clause 4.3.3.7.3. Its value shall correspond to the respective value read from the respective `ac4_substream_info_obj` or `ac4_substream_info_ajoc` or `ac4_substream_info_chan` in the respective `ac4_substream_group_info` of the respective `ac4_presentation_v1_info` according to Table E.4.

E.11.7 dsi_substream_channel_mask

This bit mask shall indicate the presence of channels in the presentation. Bit 23 (the most significant bit) shall be set to false. Bits 18...0 indicate the presence of speaker groups identified by the speaker group index described in Table A.27. Bits 22...19 are reserved.

E.11.8 b_ajoc

This bit shall indicate that the substream is coded using the A-JOC coding tool. Its value shall be equal to the value of `b_ajoc` in the respective `ac4_substream_group_info()`.

E.11.9 b_static_dmx

This bit shall indicate that the A-JOC coded substream has a static downmix. Its value shall be equal to the value of `b_static_dmx` in the respective `ac4_substream_info_ajoc()` in `ac4_substream_group_info()`.

E.11.10 n_dmx_objects_minus1

This field shall contain the number of downmix objects of an A-JOC coded substream. Its value shall be equal to the value of `n_fullband_dmx_signals_minus1` in the respective `ac4_substream_info_ajoc()` in `ac4_substream_group_info()`.

E.11.11 n_umx_objects_minus1

This field shall contain the number of upmix objects of an A-JOC coded substream. Its value shall be equal to the value of `n_fullband_upmix_signals_minus1` in the respective `ac4_substream_info_ajoc()` in `ac4_substream_group_info()`.

E.11.12 objects_assignment_mask

This bit mask shall indicate the type of objects of an object coded substream. If the substream is A-JOC coded, then the bit mask shall indicate the type of objects of the A-JOC upmix. Their values shall correspond to the object types of that substream defined in `bed_dyn_obj_assignment()` or `ac4_substream_info_obj()` according to Table E.8.

Table E.8: Object type assignment

objects_assignment_mask	object types present
0b1000	bed objects present
0b0100	dynamic objects present
0b0010	ISF objects present
0b0001	reserved

E.11.13 b_content_type

This bit indicates the presence of `content_type` information as described in ETSI TS 103 190-1 [1], clause 4.3.3.7.7.

E.11.14 content_classifier

This field shall contain the content classifier as described in ETSI TS 103 190-1 [1], clause 4.3.3.8.1. The value shall correspond to the value read from the respective `content_type` field in the `ac4_substream_group_info` in the `ac4_presentation_v1_info`.

E.11.15 b_language_indicator

This bit indicates presence of programme language indication as described in ETSI TS 103 190-1 [1], clause 4.3.3.8.2.

E.11.16 n_language_tag_bytes

This field shall contain the number of subsequent language tags bytes as described in ETSI TS 103 190-1 [1], clause 4.3.3.8.6.

E.11.17 language_tag_bytes

The sequence of `langage_tag_bytes` shall contain a language tag as described in ETSI TS 103 190-1 [1], clause 4.3.3.8.7. For the respective `ac4_substream_group_info` in the respective `ac4_presentation_info`, these values shall correspond:

- to the values of the respective `language_tag_bytes` values in the `content_type` field of the respective `ac4_substream_group_info` in the respective `ac4_presentation_info`, if `b_serialized_language_tag` is false;
- to the concatenation of `language_tag_chunk` fields in the `content_type` field of the respective `ac4_substream_info` in the respective `ac4_presentation_info` from consecutive frames, if `b_serialized_language_tag` is true.

E.12 alternative_info

E.12.1 alternative_info

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 name_len

This field shall contain the length of the following `presentation_name` string.

E.12.3 presentation_name

This field shall contain the name of the presentation.

E.12.4 n_targets

This field shall contain the value of `n_targets_minus1` + 1, with `n_targets_minus1` as defined in clause 6.3.3.1.5.

E.12.5 target_md_compat

This field shall contain the value of `target_level` as defined in clause 6.3.3.1.6.

E.12.6 target_device_category

This field shall contain the value of the respective field defined in clause 6.3.3.1.7.

Annex F (informative): Decoder Interface for Object Audio

The decoder makes data available for an object audio renderer (not specified in the present document).

As specified in clause 5.9, the decoder decodes the synchronized object essence, and object metadata, from the bitstream. The following metadata are available for the object audio renderer:

Position

The decoder provides position data, specified by room anchored coordinates. Room anchored objects are typically used for off-screen effects.

NOTE 1: Coordinates are specified in the normalized room coordinate system shown in Figure 12.

Bed

The decoder can qualify objects as bed objects, which can be interpreted as anchoring the objects directly 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.

NOTE 2: Bed objects are signalled by assigning an object audio metadata parameter specifying the associated speaker to each channel audio track as specified in clause 6.3.2.9.9 or 6.3.2.9.10.

Screen Anchoring

The decoder can qualify object position as relative to the screen. This data can be used to preserve the collocation of audio and visual events in the playback environment.

NOTE 3: 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.

Gain

The decoder provides a gain value to apply modification prior to other processing. This is a convenience functionality that might enable more efficient implementations.

Size

The decoder provides data to modify the apparent spatial extent of the object. Using this data, large objects can be efficiently represented.

NOTE 4: This control does not change the total object loudness.

Priority

The decoder provides an indication of object priority. Object priorities can support device and playback environment adaptable rendering.

EXAMPLE 1: 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.

Zone constraints

The decoder provide data to constrain the set of speakers available for the rendering process. This control can be used by content creators to provide more aesthetically pleasing results when rendering into varying loudspeaker setups.

EXAMPLE 2: An object moving from the screen to the back of the room moves through the middle of the room. Without constraints, all speakers might contribute to the rendering. With appropriate constraints, only front and back speakers will be used, improving the perceived trajectory.

Divergence

The decoder provides object divergence control data. 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. (In contrast, size is generally used to spread the energy into a volume)

NOTE 5: This control does not change the total object loudness.

Snap

The decoder provides a data indicating that an object is intended to be rendered on one loudspeaker only (typically, the nearest available). This allows rendering with maximal timbral fidelity and spatial localisation.

Interpolation

The decoder provides data to control the slope of smoothing processes if implemented in the renderer. 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 6: 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.1 shows which control data is provided by the Decoder Interface for Object Audio.

Table F.1: Control data of Decoder Interface for Object Audio

Control	Available control data	Description specified in
Position	`pos3D_{X,Y,Z}` or `diff_pos3D_{X,Y,Z}`	Clause 6.3.9.8.4
Bed	`nonstd_bed_channel_assignment` or `std_bed_channel_assignment`	Clause 6.3.2.9
Screen Anchoring	`master_screen_size_ratio`, `object_screen_factor`, `object_depth_factor`	Clause 6.3.9.2.2, clause 6.3.9.2.3, clause 6.3.9.8.17, clause 6.3.9.8.18
Gain	`object_gain_code` or `object_gain_value`	Clause 6.3.9.7.4 or clause 6.3.9.7.5
Size	`object_width_{X,Y,Z}`	Clause 6.3.9.8.12, clause 6.3.9.8.13, clause 6.3.9.8.14, clause 6.3.9.8.15, clause 6.3.9.8.16
Priority	`object_priority`	Clause 6.3.9.7.6
Zone constraints	`zone_mask`, `b_enable_elevation`	Clause 6.3.9.8.7, clause 6.3.9.8.8
Divergence	`object_div_table`, `object_div_code`, `object_div_mode`	Clause 6.3.9.8.21, clause 6.3.9.8.22, clause 6.3.9.8.23
Snap	`b_object_snap`	Clause 6.3.9.8.9
Interpolation	`ramp_duration`	Clause 6.3.9.3.8

Annex G (informative): Bibliography

ICSA 2014 050: "Evaluation of panning algorithms for theatrical applications", Nicolas Tsingos, Charles Q. Robinson, Daniel P. Darcy and Poppy A.C. Crum.

NOTE: Available at http://www.vdtshop.de/ICSA2014050.

History

Document history		
V1.1.1	April 2014	Publication as ETSI TS 103 190
V1.1.1	September 2015	Publication