INTERNATIONAL STANDARD

ISO/IEC 23008-3

Second edition
2019-02

Information technology — High efficiency coding and media delivery in heterogeneous environments —

Part 3: 3D audio

Technologies de l'information — Codage à haute efficacité et livraison des medias dans des environnements hétérogènes —

Partie 3: Audio 3D

Reference number
ISO/IEC 23008-3:2019(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This second edition cancels and replaces the first edition (ISO/IEC 23008-3:2015), which has been technically revised. It also incorporates ISO/IEC 23008-3:2015/Amd.1:2016, ISO/IEC 23008-3:2015/Amd.2:2016, ISO/IEC 23008-3:2015/Amd.3:2017 and ISO/IEC 23008-3:2015/Amd.4:2016.

The main changes compared to the previous edition are as follows:

— unreadable equations have been corrected;

— profiles have been defined;

— transport of MPEG-H 3D audio in MPEG-4 ISO Base Media File Format has been defined;

— coding efficiency, especially for low bitrate coding modes, has been improved (for scene-based as well as for object-based and for multichannel-based content);

— descriptive metadata has been added;

— MHAS description has been updated;

— usage of MPEG-H 3D audio in broadcasting applications has been greatly improved;

— a tool for Advanced Loudness Control has been added;

— a layered coding mode for coding of scene-based content has been added;

— carriage of systems metadata has been defined.

A list of all parts in the ISO/IEC 23008 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

3D sound systems are able to realize a significantly enhanced sound experience relative to current widespread 5.1 channel audio programs and playback systems. These systems demand high quality audio coding and error-free transmission in order to keep the timbre, sound localization and sound envelopment of the original audio program. Presentation over headphones with suitable spatialization are also considered.

This document provides means for all scenarios where there is a need to compress a multi-channel audio program (e.g. 22.2 channel program) and to render it to the native target number of loudspeakers. In order to reach a wide market, a 3D audio program is able to be downmixed to a lower hierarchy of loudspeakers, for example 10.1 or 8.1 channels. In addition, all scenarios support a level of random access to facilitate broadcast break-in, and "trick modes" such as fast forward when playing from stored media.

This document focuses on applications such as audio for home theatres where the audio presentation is immersive, involving many loudspeakers (e.g. from 10 to more than 20) surrounding the listener and placed below, at and above ear-level. Moreover, applications as personal TV, TV for smartphones and multi-channel audio-only programs are envisioned. These require that 3D audio encoding/decoding systems are able to operate at low bitrates appropriate for efficient transmission over a cellular channel. At the same time, the sense of envelopment and accurate sonic localization even for systems having a tablet-sized visual displays with loudspeakers built into the device and headphone listening are maintained.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of patents. ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are registered with ISO and IEC. Information may be obtained from:

Electronics and Telecommunications Research Institute (ETRI)	218 Gajeong-ro, Yuseong-gu, Daejeon, 34129, KOREA
Koninklijke Philips N.V.	High Tech Campus 5, 5656AE Eindhoven, THE NETHERLANDS
Thomson Licensing	Suite 303, 4 Research Way, Princeton, NJ 08540, USA
Wilus Inc.	48 Mabang-ro, Seocho-gu, Seoul, 137-894, KOREA
Fraunhofer Gesellschaft zur Foerderung der angewandten Forschung e.V.	Am Wolfsmantel 33, 90158 Erlangen, GERMANY
Qualcomm Incorporated	5775 Morehouse Drive, San Diego, CA 92021, USA
Dolby Laboratories Licensing Corporation	100 Potrero Avenue, San Francisco, CA 94103-4938, USA
Dolby International AB	999 Brannan Street, San Francisco, CA 94103-4938, USA

Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 3: 3D audio

1 Scope

This document specifies technology that supports the efficient transmission of immersive audio signals and flexible rendering for the playback of immersive audio in a wide variety of listening scenarios. These include home theatre setups with 3D loudspeaker configurations, 22.2 loudspeaker systems, automotive entertainment systems and playback over headphones connected to a tablet or smartphone.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 13818-1, *Information technology — Generic coding of moving pictures and associated audio information — Part 1: Systems*

ISO/IEC 14496-3:2009, *Information technology — Coding of audio-visual objects — Part 3: Audio*

ISO/IEC 14496-11, *Information technology — Coding of audio-visual objects — Part 11: Scene description and application engine*

ISO/IEC 23001-8, *Information technology — MPEG systems technologies — Part 8: Coding-independent code-points*[1]

ISO/IEC 23003-1:2007, *Information technology — MPEG audio technologies — Part 1: MPEG Surround*

ISO/IEC 23003-2, *Information technology — MPEG audio technologies — Part 2: Spatial Audio Object Coding (SAOC)*

ISO/IEC 23003-3:2012, *Information technology — MPEG audio technologies — Part 3: Unified speech and audio coding*

ISO/IEC 23003-4:2015, *Information technology — MPEG audio technologies — Part 4: Dynamic range control*

IETF RFC 4122, July 2005, *A Universally Unique IDentifier (UUID) URN Namespace*

[1] ISO/IEC 23001-8 has been superseded by ISO/IEC 23091 (all parts).

3 Terms, definitions, symbols, abbreviated terms and mnemonics

3.1 Terms, definitions, symbols and abbreviated terms

For the purposes of this document, the terms, definitions, symbols and abbreviations in ISO/IEC 14496-3:2009, 1.3 and 1.4 and in ISO/IEC 23003-3:2012, 3.1 apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.2 Mnemonics

The following mnemonics are defined to describe the different data types used in the coded bitstream payload.

bslbf	Bit string, left bit first, where “left” is the order in which bit strings are written in ISO/IEC 14496 (all parts). Bit strings are written as a string of 1s and 0s within single quote marks, for example '1000 0001'. Blanks within a bit string are for ease of reading and have no significance.
uimsbf	Unsigned integer, most significant bit first.
vlclbf	Variable length code, left bit first, where “left” refers to the order in which the variable length codes are written.
tcimsbf	Two’s complement integer, most significant (sign) bit first.

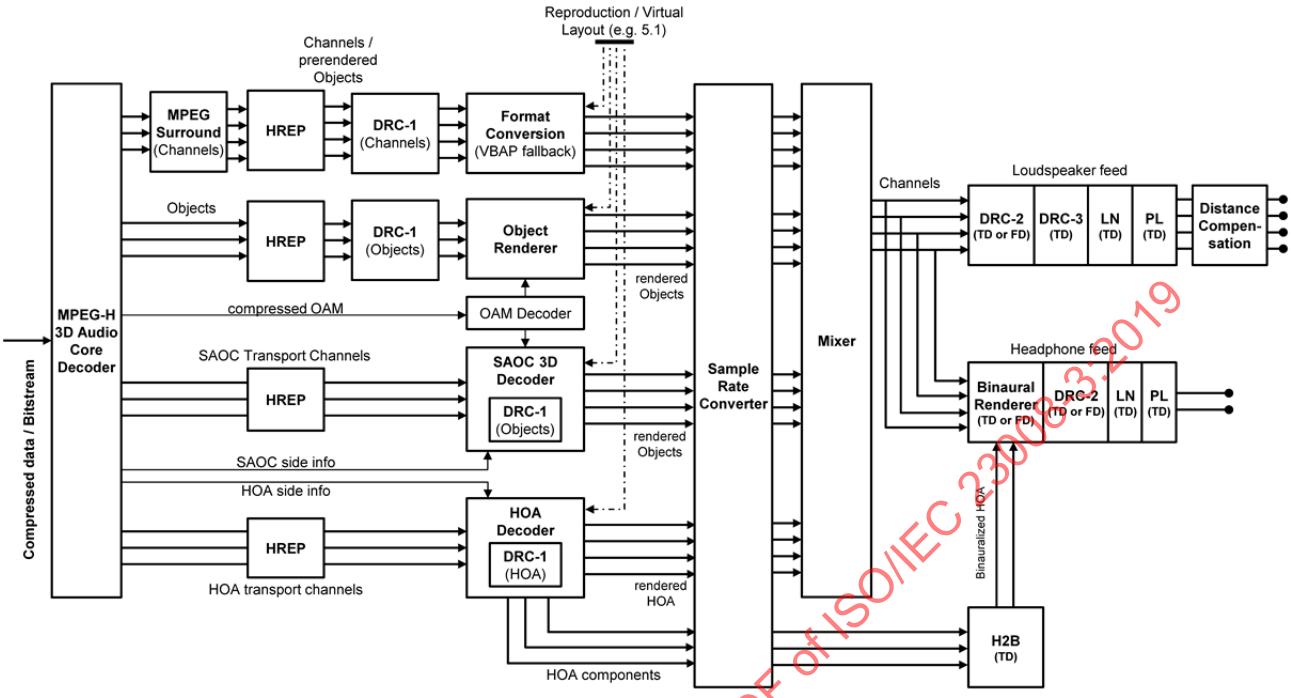
4 Technical overview

4.1 Decoder block diagram

The 3D audio codec system consists of an MPEG-H 3D audio core codec for coding of channel, object and higher order ambisonics (HOA) signals. The core codec is based on the MPEG-D USAC codec. To increase the efficiency for coding a large amount of objects, MPEG SAOC technology has been adopted. Several types of renderers perform the tasks of rendering objects to channels, rendering channels to a different loudspeaker setup, rendering HOA signals to the loudspeaker setup or rendering virtual loudspeaker channels or HOA components to headphones.

When object signals are explicitly transmitted or parametrically encoded using SAOC, the corresponding object metadata information is compressed and multiplexed into the 3D audio bitstream.

Figure 1 shows the different algorithmic blocks of the 3D audio system.



Key

DRC	dynamic range control	HREP	high resolution envelope processing
SAOC	spatial audio object coding	OAM	object audio metadata
HOA	higher order ambisonics	H2B	HOA to binaural
LN	loudness normalization	PL	peak limiter

Figure 1 — 3D audio decoder

4.2 Overview over the codec building blocks

The MPEG-H 3D audio core codec for loudspeaker-channel signals, discrete object signals, object downmix signals and pre-rendered signals is based on MPEG-D USAC technology. It handles the coding of the multitude of signals by creating channel- and object-mapping information based on the geometric and semantic information of the input's channel and object assignment. This mapping information describes how input channels and objects are mapped to channel elements (CPEs, SCEs, LFEs) and the corresponding information is transmitted to the decoder.

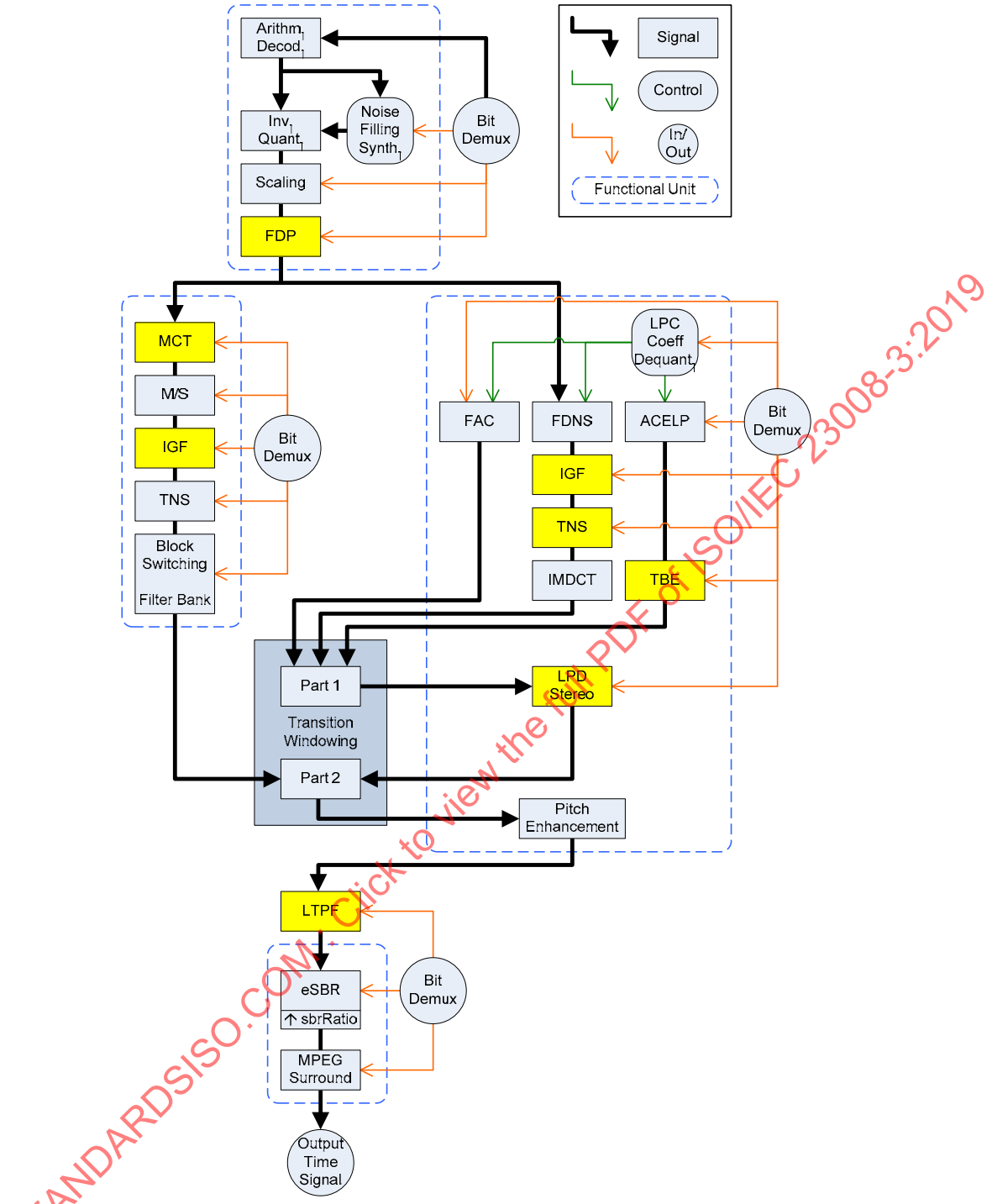


Figure 2 — Simplified typical MPEG-H core decoder configuration

Figure 2 shows a simplified block diagram of the typical MPEG-H core decoder building blocks. The major differences compared to MPEG-D USAC technology are highlighted in yellow. The highlighted tools are described in detail in Clause 5.

The coding of objects is possible in different ways, depending on the rate/distortion requirements and the interactivity requirements for the renderer. The following object coding variants are possible.

— Prerendered objects: Object signals are pre-rendered and mixed to multi-channel or HOA signals before encoding, as appropriate. The subsequent coding chain then operates on multi-channel or HOA signals.

— Discrete object waveforms: Objects are supplied as monophonic waveforms to the encoder. The encoder uses single channel elements SCEs to transmit the objects in addition to the channel signals. The decoded objects are rendered and mixed at the receiver side. Compressed object metadata information is transmitted to the receiver/renderer alongside.

— Parametric object waveforms: Object properties and their relation to each other are described by means of SAOC parameters. The downmix of the object signals is coded with the MPEG-H 3D audio core codec. The parametric information is transmitted alongside. The number of downmix channels is chosen depending on the number of objects and the overall data rate. Compressed object metadata information is transmitted to the SAOC renderer.

The SAOC encoder and decoder for object signals are based on MPEG SAOC technology. The system is capable of recreating, modifying and rendering a number of audio objects based on a smaller number of transmitted channels and additional parametric data (OLDs, IOCs, DMGs).

The SAOC decoder reconstructs the object/channel signals from the decoded SAOC transport channels and parametric information, and generates the output audio scene based on the reproduction layout, the decompressed object metadata information and optionally on the user interaction information.

The object metadata codec efficiently codes the associated metadata that specifies the geometrical position and volume of each object in 3D space by quantization of the object properties in time and space. The compressed object metadata is transmitted to the receiver as side information.

The object renderer utilizes the compressed object metadata to generate object waveforms according to the given reproduction format. Each object is rendered to certain output channels according to its metadata. The output of this block results from the sum of the partial results.

The loudspeaker renderer converts between the transmitted channel configuration and the desired reproduction format. It is thus called 'format converter'. In case of conversions to lower numbers of output channels it creates downmixes. The system automatically generates optimized downmix matrices for the given combination of input and output formats and applies these matrices in a downmix process. The format converter allows for standard loudspeaker configurations as well as for random configurations with non-standard loudspeaker positions.

The higher order ambisonics (HOA) decoder/renderer reconstructs the HOA coefficient signals based on the HOA transport channels decoded by the 3D audio core decoder and the HOA specific side information. The coding principle is based on a separate transmission of so-called predominant sounds and ambient sound scene components. Subsequently the HOA renderer generates the loudspeaker channel feeds based on the reproduction layout.

If two or more groups of channel based content, discrete/parametric objects or HOA based content are decoded, the corresponding waveforms are delay-aligned and sample-wise added by the mixer before providing the resulting waveforms (or before feeding them to a postprocessor module such as the binaural renderer, DRC-2, DRC-3, the peak limiter PL, or the loudspeaker distance compensation).

The sample rate converter block converts between the core decoder sampling rate and the decoder output sampling rate. It enables a constant output sampling rate in case of varying core coding sampling rates and allows for resampling of audio scenes, which may be coded in multiple sub-streams with different core sampling rates, to a common output sampling rate.

Note that in case DRC-2/3 gains need to be applied in the post-processing context (see Figure 2), the DRC-2/3 gains should be resampled similar to the audio samples.

The MPEG surround decoder takes the downmix signals coming from the MPEG-H 3D audio decoder and performs the guided MPEG surround upmix using the MPEG surround side information to reproduce the multichannel signal for the transmitted loudspeaker layout.

The binaural renderer module produces a binaural downmix of the multichannel audio material, such that each input channel is represented by a virtual sound source. The processing is conducted frame-wise in the QMF domain or in the time domain. The binauralization is based on measured binaural room impulse responses (BRIRs).

Virtual layout information fed from an application shall be consistent with the corresponding BRIR set provided (expressed with MeasurementSetup, see Table 249), in the sense that the set of positions corresponding to the virtual layout is a subset of the set of positions corresponding to the BRIRs.

4.3 Efficient combination of decoder processing blocks in the time domain and QMF domain

There are numerous processing blocks in the MPEG-H 3D audio decoding, rendering and processing framework. In general these blocks can be classified into different classes:

1) Block operates in the time domain (**TD**)
2) Block operates in frequency domain (**FD**) – also called QMF domain
3) Block is "**neutral**" and can operate in FD or TD.
   Signal-processing-wise the same operation is carried out in either FD or TD. Operation in a different domain may cause no difference or only a small but perceptually negligible difference to the output signal.

Table 1 lists all blocks with their corresponding processing domain. For the sake of clarification, the functional blocks are grouped into semantically differentiable contexts, which follow the MPEG-H 3D audio signal processing as shown in Figure 3.

Table 1 — MPEG-H 3D audio functional blocks and internal processing domain

Processing context	Functional block	Processing domain	Delay samples [$1/f_{s,core}$] or [$1/f_{s,out}$]	Contribution to maximum delay high profile samples [$1/f_{s,out}$]	Contribution to maximum delay low complexity profile samples [$1/f_{s,out}$]
Audio core		**TD,** Core frame length = 1024	0		
	SBR / MPS212	**FD,** Core frame length = 2048 or 4096	0		
	SBR/MPS212, stereoConfigIndex==3	**FD,** Core frame length = 2048 or 4096	384	**384 × RSR_{max}**	
	MPEG-H 3D audio core coder	**FD** or **TD**			
	MPEG surround	**FD**	640	**640 × RSR_{max}**	
	HREP	**TD,** Core frame length = 1024	64		
	HREP, QMF-synthesis and QMF-analysis pair and alignment to 64 sample grid	**FD TD FD**	64 + 257 + 320 + 63	**(64 + 257 + 320 + 63) × RSR_{max}**	
	QMF-analysis	**TD, FD**	320	**320 × RSR_{max}**	
	QMF-synthesis	**FD, TD**	257	**257 × RSR_{max}**	
	Hybrid filter/SBR	**FD**	384	**384 × RSR_{max}**	
Rendering	DRC-1	if multiband: **TD (STFT)** or **FD** else: **neutral**	0		
	Format converter, core frame length = 1024	**FD**	3072	**3072 × RSR_{max}**	
	Format converter, core frame length = 2048 or 4096	**FD**	2048		
	STFT format converter	**TD (STFT)**	256		**256 × RSR_{max}**
	Object renderer	**Neutral**	0		
	SAOC 3D decoder, bsDoubleFrameLengthFlag=0	**FD**	0		
	SAOC 3D decoder, bsDoubleFrameLengthFlag=1 Core frame length = 1024	**FD**	1024		
	SAOC 3D decoder, bsDoubleFrameLengthFlag=1 Core frame length = 2048	**FD**	2048		
	HOA decoder	**TD, (FD)** [c]	0 | 577		
	HOA decoder with multiband DRC-1	**TD, FD or TD (STFT), TD**	**577 | 256**		
	Sample rate converter	**FD, TD**	256	**256**	**256**
Mixing	Mixer	**FD, TD**	0		

Processing context	**Functional block**	**Processing domain**	**Delay** samples $[1/f_{s,core}]$ or $[1/f_{s,out}]$	**Contribution to maximum delay high profile** samples $[1/f_{s,out}]$	**Contribution to maximum delay low complexity profile** samples $[1/f_{s,out}]$
Maximum accumulated delay to mixer:				**5761 × RSR_{max} + 256**	**256 × RSR_{max} + 256**
Post-processing	QMF-analysis	**TD, FD**	320	**320**	**320**
	QMF-synthesis	**FD, TD**	257	**257**	**257**
	DRC-2	if multiband: **FD** else: **neutral**	0		
	FD binauralizer	**FD**	0		
	TD binauralizer	**TD**	0		
End of chain	DRC-3 (only singleband)	**TD**	0		
	Loudness normalization	**TD**	0		
	Peak limiter [a]	**TD**	240	**240**	**240**
	LS distance compensation	**TD**	0		
***Maximum accumulated delay* to decoder output [a, b]:**				5761 × RSR_{max} + 1073 **= 18356**	256 × RSR_{max} + 1073 **= 1841**

[a] The given values are valid for $f_{s,out}$ =48 kHz. The peak limiter shall introduce a delay of 5ms, i.e. the limiter delay in samples shall be determined as floor($f_{s,out}$ · 5/kHz). The maximum accumulated delay to the decoder output has to be adapted accordingly for output sampling rates the differ from $f_{s,out}$ = 48 kHz.

[b] The maximum resampling factor RSR_{max} of the sampling rate converter is 3.

[c] HOA Decoder: FD in case subband directional prediction or PAR is used (only high profile), low complexity profile only operates in TD.

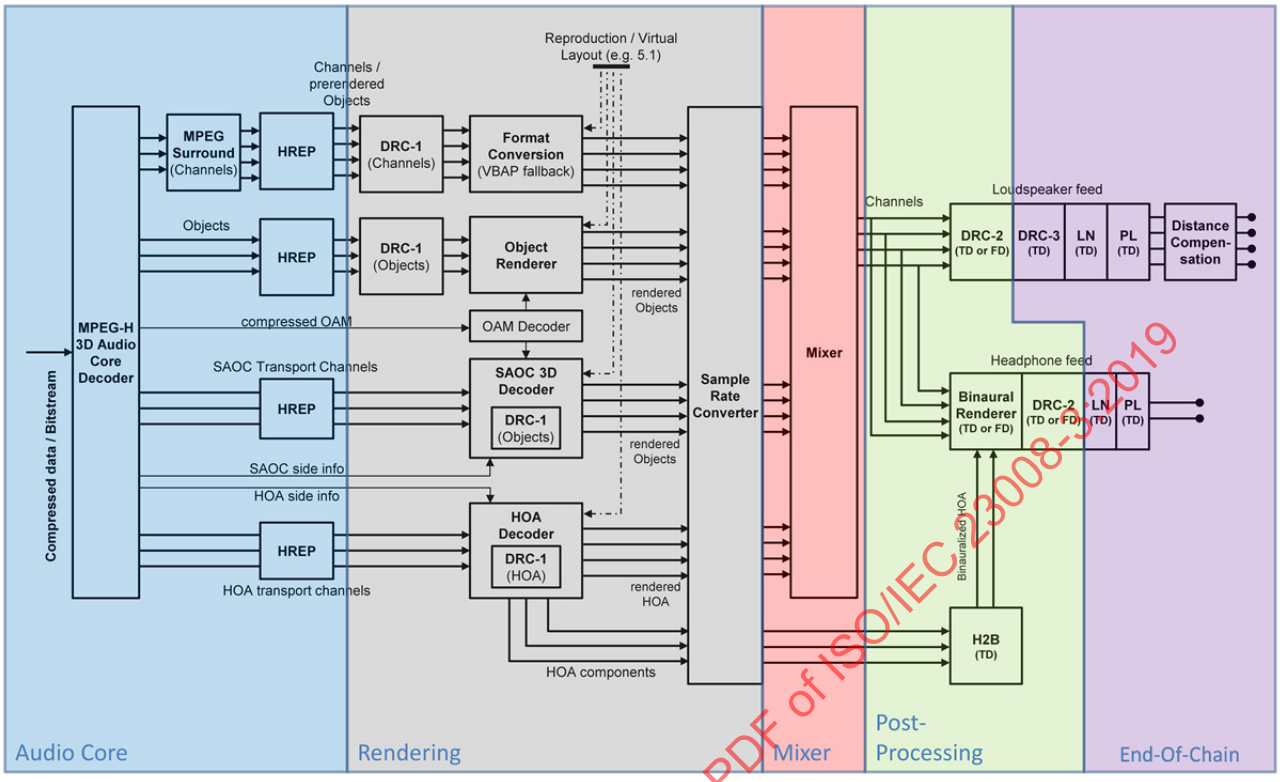


Figure 3 — MPEG-H 3D audio decoder overview with signal processing context

Operating blocks that are directly connected in the signal chain and which operate in the *same* domain can interface in that common domain.

Operating blocks that are directly connected in the signal chain and which operate in *different* domains require a transformation block which turns one signal representation into another, i.e. a QMF analysis (TD to FD) or a QMF synthesis (FD to TD). This transform causes additional delay and hence needs to be taken into account when determining the overall signal processing delay.

The following subclause describes formal generic rules which determine how the various functional blocks can be connected.

4.4 Rule set for determining processing domains

4.4.1 Audio core codec processing domain

The incoming bitstream configuration determines which core decoder tools are active and whether parts of the core codec operate in QMF domain / FD.

a) If last decoding stage is in QMF domain (SBR and/or MPS212 active)
(typical for mid to low bit rate coding)
   1) formal audio core codec processing domain is **FD**

b) If last decoding stage is **not** in QMF domain (neither SBR nor MPS212 active)
(typical for high rate coding)
   1) formal audio core codec processing domain is **TD**

4.4.2 Mixing

— The mixer may operate in **TD** or in **FD**

4.4.3 DRC-1 Operation domains (DRC in rendering context)

- If multiband DRC-1 data is present in the bitstream for a DRC-1 module,
  - the corresponding DRC-1 processing block should operate in **FD** or **TD (STFT)**

4.4.4 Audio core codec interface domain to rendering

- If the audio core codec processing domain is **TD**
  - If the rendering block requires operation in **FD**
    - apply QMF analysis and output **FD** for this rendering block
  - else
    - output **TD**
- If the audio core codec processing domain is **FD**
  - If the subsequent rendering block requires operation in **TD**
    - apply QMF synthesis and output **TD** for this rendering block
  - else
    - output **FD**

4.4.5 Rendering context

Rendering paths are considered individually:

- If the rendering block operates in **FD** and mixer operates in **TD**
  - apply QMF synthesis after renderer and output **TD**
- If the rendering block operates in **TD** and mixer operates in **FD**
  - apply QMF analysis after renderer and output **FD**
- else
  - keep processing domain

In case the core decoder operates in the time domain (TD), i.e. if the core decoder does not apply QMF domain processing like SBR or MPS-212, the low-complexity STFT domain downmix as specified in clauses 24 and 25 shall be applied for format conversion if format conversion is requested.

4.4.6 Post-processing context

- If mixer operates **TD**
  - If **any** block in post-processing context requires operation in **FD**
    - convert to **FD** immediately after mixer;
    - after the **FD** operation convert to **TD**, i.e. prior to end-of-chain context
  - else
    - stay in **TD**
- If Mixer operates in **FD**
  - If **no** block in post-processing context requires operation in **FD**
    - convert to **TD** prior to DRC-2

— else

— convert to **TD** prior to end-of-chain context

4.4.7 End-of-chain context

All blocks shall operate in **TD**.

4.5 Sample rate converter

The sample rate converter performs sample rate conversion between the core coder sampling rate and the output sampling rate of the decoder. The signal delay introduced by the sample rate converter shall not exceed 256 samples, measured in samples at the output sampling rate of the decoder. The sample rate converter shall be able to perform sampling rate conversions with resampling ratios (defined as decoder output sampling rate divided by core decoder output sampling rate) of 3/2, 2, and 3. The sample rate converter block shall further be able to pass through the signal(s) if the core decoder sampling rate matches the decoder output sampling rate.

4.6 Decoder delay

If a constant decoder delay application is signalled in the bitstream (**receiverDelayCompensation**==1, as defined in subclause 5.2.2) then the decoding and rendering delay from the IMDCT output of the core decoder to the mixing block of the MPEG-H 3D audio decoder shall be kept constant by introducing delay lines where required. Similarly, the overall delay from the IMDCT output to the decoder output shall be kept constant by introducing delay lines where required. The constant delay values that shall be fulfilled are determined by the ***maximum accumulated delay*** numbers in Table 1.

Enforcing a constant delay from the IMDCT output to the mixer implies that all signals at the IMDCT output shall be aligned. Accordingly, the decoder shall compensate for different delays that may occur in the processing of the IMDCT output signals, e.g. different SBR/MPS processing delays for SCEs and CPEs in case of stereoConfigIndex==3.

Further, all side information utilized in the rendering blocks as well as for DRC processing shall be sent aligned to the IMDCT output waveforms independent of the value of receiverDelayCompensation. The rendering and DRC side information shall be applied delayed in the rendering/DRC blocks. The side information delays are determined by the delays the waveforms encounter in the processing pipeline from the IMDCT output until reaching the corresponding rendering/DRC blocks. The following kinds of side information shall be aligned to the IMDCT output waveforms and shall be applied delayed as defined above:

— OAM object metadata;

— DRC side information;

— SAOC side information;

— HOA side information.

In case a bitstream contains multiple signal groups with different signalGroupTypes, the bitstream element receiverDelayCompensation shall be set to 1.

4.7 Contribution mode of MPEG-H 3D audio

The contribution mode of MPEG-H 3D audio specifies a generic transport mechanism for audio signals with accompanying metadata and it is designed to be unaware of the signal type and of the content and structure of the associated metadata.

A value of **speakerLayoutType** == 3 as defined in Table 66 in the signalling of the **referenceLayout** in the **mpegh3daConfig()** indicates that MPEG-H 3D audio shall operate in contribution mode. In contribution mode the rendering context shall operate in a pass-through mode, i.e. the format converter shall apply an identity matrix to the signal. contribution mode bitstreams shall have the following additional restrictions:

— bsNumSignalGroups == 0 (one single signal group);

— SignalGroupType == 0 (channel signal group);

— differsFromReferenceLayout[0] == 0 (no audioChannelLayout in the signal group);

— core coder delay lines for compensation of SBR, MPS212, shall be applied;

— Content of extension elements shall not be processed in the decoder but shall be made available to an external framework.

4.8 MPEG-H 3D audio profiles and levels

4.8.1 General

This subclause defines profiles and their levels for MPEG-H 3D audio.

Complexity units are defined to give an approximation of the decoder complexity in terms of processing power required for the decoding process. The approximated processing power is given in processor complexity units (PCU), specified in millions operations per second (MOPS).

4.8.2 Profiles

The following audio profiles are defined:

1) The main profile of MPEG-H 3D audio provides a complete set of features for low-bitrate and high-quality coding, and rendering for all playback scenarios.

   NOTE The definition of the main profile, its associated bitstream syntax, semantics, and decoding process description was captured in ISO/IEC 23008-3:2015 (the first edition of this document).

2) The high profile of MPEG-H 3D audio provides a complete set of features for low-bitrate and high-quality coding, and rendering for all playback scenarios. The high profile is a superset of the low-complexity profile.

3) The low complexity profile provides features for broadcasting and streaming with reduced decoder complexity.

Table 2 — Summary of the location of and normative reference to the definitions of MPEG-H 3D audio profiles

Tool/Module		Defined in ISO/IEC	Sub-clause	USAC 23003-3	MPEG-H 3D audio High profile	MPEG-H 3D audio Low-complexity profile
block switching		14496-3	4.6.11	**X**	X	X
window shapes	AAC based	14496-3	4.6.11	**X**	X	X
	additional windows	23003-3	6.2.9.3	**X**	X	X
filter bank	AAC based	14496-3	4.6.11	**X**	X	X
	additional USAC	23003-3	7.9	**X**	X	X
TNS		14496-3	4.6.9	**X**	X	X
intensity		14496-3	4.6.8.2			
coupling		14496-3	4.6.8.3			
perceptual noise synthesis	PNS	14496-3	4.6.13			
	noise filling	23003-3	7.2	**X**	X	X
MS	basic mid/side coding	14496-3	4.6.8.1	**X**	X	X
	MDCT based complex prediction	23003-3	7.7.2	**X**	X	X
quantization	non-uniform	14496-3	4.6.1	**X**	X	X
	uniform	23003-3	7.1	**X**	X	X
entropy coding (spectral coef.)	Huffman coding	14496-3	4.6.3			
	context adaptive arithmetic coding	23003-3	7.4	**X**	X	X
SBR	base	14496-3	4.6.18	**X**	X	
	enhanced	23003-3	7.5	**X**	X	
parametric stereo extension	parametric stereo	14496-3	8.6.4 / 8.A			
	MPEG surround 2-1-2 (incl. residual coding)	23003-3	6.2.13	**X**	X	
	quad channel element	23008-3	5.5		X	
ACELP		23003-3	7.14	**X**	X	X
frequency domain noise shaping	scale factor based	14496-3	4.6.2	**X**	X	X
	LPC based	23003-3		**X**	X	X
intelligent gap filling	IGF for FD	23008-3			X	X
improved LPD coding	IGF for TCX and TBE in ACELP	23008-3			X	X
	LPD stereo	23008-3			X	X
predictors for FD and TCX	frequency-domain prediction and time-domain post-filtering	23008-3			X	X
discrete multi-channel coding	MCT	23008-3			X	X
format converter	generic downmix	23008-3	10, 24		X	X [d]

Tool/Module		Defined in ISO/IEC	Sub-clause	USAC 23003-3	MPEG-H 3D audio High profile	MPEG-H 3D audio Low-complexity profile
immersive rendering	immersive rendering within format converter	23008-3	11, 25		X	X [d]
static metadata	metadata audio elements (MAE) and audio scene information (ASI) decoder and renderer	23008-3	15		X	X
dynamic object metadata	object audio metadata (OAM) decoder and renderer	23008-3	7, 8		X	X
MPS	MPEG surround extension	23003-1	10		X	
SAOC-3D	decoder and renderer	23008-3	9		X	
HOA	decoder and renderer	23008-3	12		X	X [e]
	near field compensation	23008-3			X	X [a]
	subband directional prediction	23008-3			X	
	parametric ambiance replication (PAR)	23008-3			X	
	phase-based decorrelation	23008-3			X	
binaural	FD-binaural, TD-binaural	23008-3	13		X	X [b]
	HOA2Binaural H2B	23008-3			X	X [b]
DRC	DRC-1	23003-4			X	X [c]
	DRC-2 (single band)	23003-4			X	X
	DRC-2 (multi band)	23003-4				
	DRC-3 (single band)	23003-4			X	X
sample rate converter		23008-3			X	X
peak limiter	unguided clipping prevention	23008-3 23003-4	D		X	X
loudness	loudness metadata and handling	23003-4	6		X	X
	loudness compensation	23008-3			X	X
MHAS	MPEG-H 3D audio stream	23008-3	14		X	X
	truncation message and CRC packet type, ASI packet type	23008-3			X	X
file format	carriage of MPEG-H 3D audio in ISO base media file format	23008-3			[f]	

Tool/Module		Defined in ISO/IEC	Sub-clause	USAC 23003-3	MPEG-H 3D audio High profile	MPEG-H 3D audio Low-complexity profile
interfaces and processing	interfaces and processing for interaction data and local setup info	23008-3	17,18		X	X
carriage of system data	carriage of system data for the interaction with system engine	23008-3			X	X
TCC	tonal component coding	23008-3			X	
IC	internal channel	23008-3			X	
HREP	high resolution envelope processing	23008-3			X	

[a] Restrictions apply dependent on the levels.
[b] Implementation of binaural rendering is only mandated if headphone reproduction is supported.
[c] Multi-band DRC-1 shall be applied in the STFT domain of the TD format converter.
[d] The TD format converter downmix shall be applied for downmixing.
[e] In order to achieve target complexity for the LC profile at a given level, study Annex G.
[f] File format encapsulation is independent of the profile that is used for the bitstream. A profile level indicator is part of the file format specification (see subclause 20.4).

4.8.2.1 Levels of the low complexity profile

Table 3 — Levels and their corresponding restrictions for the low complexity profile

Level	Max. sampling rate	Max. no. of core ch. in compressed data stream	Max. no. of decoder processed core ch.	Max. no. of loud speaker output ch.	Example of max. loud speaker configuration	Max. no. of decoded objects	Example of a max. config C+O	Max. HOA order	Example of max. HOA order + O
1	48000	10	5	2	2.0	5	2 ch. + 3 static obj. [a]	2	2nd order + 3 static obj. [a]
2	48000	18	9	8	7.1	9	6 ch. + 3 static obj. [a]	4	4th order + 3 static obj. [a]
3	48000	32	16	12	11.1	16	12 ch. + 4 obj.	6	6th order + 4 obj.
4	48000	56	28	24	22.2	28	24 ch. + 4 obj.	6	6th order + 4 obj.
5	96000	56	28	24	22.2	28	24 ch. + 4 obj.	6	6th order + 4 obj.

[a] In this context "static objects" are understood as channel-based signals without accompanying OAM data which are not also associated to a channel bed.

— The use of switch groups determines the subset of core channels from the core channels in the bitstream that shall be decoded.

— If the mae_AudioSceneInfo() contains switch groups (mae_numSwitchGroups>0), then the elementLengthPresent flag shall be 1.

— The number of channels of the signalled referenceLayout shall not exceed the maximum number of loudspeaker output channels as defined in the levels Table 3.

Table 4 — Approximated worst case processing power (PCU) of decoder modules and the whole decoder for the different levels of the low complexity profile given in MOPS

Level	Core LC	Format converter	Object renderer	HOA [a]	Objects only renderer	DRC	Limiter	Binaural [b]	Worst case PCU
1	33	3	0	3	9	6	4	7	58
2	59	10	0	17	16	18	5	19	118
3	106	36	7	36	29	24	6	27	206
4	186	113	7	93	50	30	9	46	392
5	373	226	14	186	50	34	19	92	758

[a] The complexity numbers for the HOA spatial decoding and rendering are based on the low complexity combined HOA spatial decoding and rendering described in Annex G.

[b] The complexity numbers for binaural processing are calculated on the basis of BRIR filters of 1 second length measured in a BS.1116 compliant room.

4.8.2.2 Restrictions for the low complexity profile and levels

In the low complexity profile the core decoder, format converter, object renderer, HOA renderer and DRC and peak limiter operate in the time domain, MDCT-domain or STFT-domain.

The following restrictions apply for HOA renderer and decoder.

Table 5 — Restrictions for the HOA spatial decoding and rendering according to the level of the low complexity profile

Restriction applies to	Maximum allowed value depending on Mpegh3daProfileLevelIndication				
	Lvl 1	Lvl 2	Lvl 3	Lvl 4	Lvl 5
HOA order (max)	2	4	6	6	6
Number of predominant sounds (max)	3	5	7	8	8
Number of directional signals used in prediction (max): MaxNoOfDirSigsForPrediction	2	3	no restrictions apply	no restrictions apply	no restrictions apply
The near field compensation (NFC) processing may be applied to HOA content of an order which is smaller or equal to:	N/A (NFC not allowed)	1	2	3	3

NFC may be employed in not more than one signal group of type SignalGroupTypeHOA.

The following restrictions apply to MPEG-D DRC (ISO/IEC 23003-4) when employed as part of MPEG-H 3D audio.

— drcFrameSizePresent and timeDeltaMinPresent shall be set to 0.

— gainInterpolationType shall be set to 1.

— dependsOnDrcSetPresent shall be set to 0 for drcInstructionsUniDrc() with downmixId == 0.

— HOA signal groups shall be restricted to one drcChannelGroup and DRC gains shall be applied to the HOA core channels (HOATransportChannels).

— The values of bsSequenceIndex within drcInstructionsUniDrc() shall be unique in simultaneously applied DRC sets except for bsSequenceIndex == 0.

— Multiband DRC shall be restricted to drcInstructionsUniDrc() with downmixId == 0. If the bitstream should contain multiband DRC, the number of multiband DRC core channels shall be restricted as follows:

(numAudioChannels +
numAudioObjects + numAudioObjectsMB +
numHOATransportChannels + numHOATransportChannelsMB)
≤ (numCoreChannelsMax(Lvl) – dependsOnDrcSetPresentFlag – 1),

where

— numAudioChannels, numAudioObjects and numHOATransportChannels are the number of C, O and HOA core channels as specified in Table 14;

— numAudioObjectsMB and numHOATransportChannelsMB are the number of O and HOA core channels out of numAudioObjects and numHOATransportChannels that contain multiband DRC;

— numCoreChannelsMax is the maximum number of decoder processed core channels depending on the Mpegh3daProfileLevelIndication field as defined in Table 3;

— dependsOnDrcSetPresentFlag is set to one if the bitstream contains any configuration with dependsOnDrcSetPresent==1 (otherwise zero).

— nNodes shall be restricted to a maximum value of 32, where nNodes is the number of encoded gain values in the current DRC frame.

— loudnessInfoSetPresent within mpegh3daUniDrcConfig() shall be set to 0.

— nDrcChannelGroups shall be restricted to 1 for drcInstructionsUniDrc() with downmixId != 0.

Table 6 — Restrictions applying to DRC processing according to the levels of the low complexity profile

Restriction applies to	Maximum allowed value depending on Mpegh3daProfileLevelIndication				
	Lvl 1	Lvl 2	Lvl 3	Lvl 4	Lvl 5
nDrcChannelGroupsTotal [a]	5	9	16	28	28
drcCoefficientsUniDrcCount	4	4	4	4	4
bandCount [b]	2	4	4	4	4
sequenceCountTotal [c]	24	28	32	48	63
drcInstructionsUniDrcCount	16	16	32	32	32

[a] Maximum allowed number of simultaneously active DRC channel groups in all applied DRC sets.
[b] Maximum allowed number of DRC bands for multiband DRC.
[c] Sum of all nDrcBands in drcGainSequence() structures plus number of sequences with gainCodingProfile=3.

The following tool specific restrictions apply:

— If the independent noise filling (INF) of the intelligent gap filling (IGF) is activated (i.e. if igfUseEnf==1), then the complex prediction tool shall be restricted to real-only prediction, i.e. complex_coef shall be 0.

— If stereo filling is activated (i.e. if stereo_filling==1), then the complex prediction tool shall be restricted to real-only prediction, i.e. complex_coef shall be 0.

— The independent noise filling of the intelligent gap filling shall not be employed in cases where igfBgn corresponds to an audio frequency higher than 8 kHz.

— The LPD mode shall only be employed at 3D audio core coder sampling rates (as defined in Table 12) ≤32 000 Hz.

EXAMPLE For a 48 kHz input signal, the encoder resamples the signal to a 32 kHz core coder sampling rate and the LPD decoder operates at this lower sampling rate. After the core decoding the signal is resampled to 48 kHz.

— The multi-channel coding tool (MCT) shall not employ more stereo boxes than specified in Table **7**.

Table 7 — Restrictions applying to MCT processing according to the levels of the low complexity profile

Restriction applies to	Maximum allowed value depending on Mpegh3daProfileLevelIndication				
	Lvl 1	Lvl 2	Lvl 3	Lvl 4	Lvl 5
Number of stereo boxes in MCT	5	9	16	28	28

The following restrictions apply to coding of audio objects and the associated OAM data.

Table 8 — Restrictions applying to object processing according to the levels of the low complexity profile

Restriction applies to	Maximum allowed value n depending on Mpegh3daProfileLevelIndication				
	Lvl 1	Lvl 2	Lvl 3	Lvl 4	Lvl 5
(number of objects without divergence) + 3·(number of objects with divergence > 0) ≤ n	5	9	16	28	28

— Efficient object metadata decoding is not permitted, i.e. lowDelayMetadataCoding shall be 1.

— Furthermore the OAM frame length shall comply to:

OAMFrameLength = outputFrameLength / *n*,
with *n* being a positive integer in the range of {1,...,4}

— Objects shall not employ divergence and spread at the same time.

— If an object is defined with a spatial extent (spread $\alpha > 0{,}0°$ for uniform spread, spread_width $\alpha_{width} > 0{,}0°$ for non-uniform spread) it shall have a divergence value equal to zero.

— If an object is defined with a divergence value > 0, it shall not have a spatial extent (spread α shall be equal to 0,0° for uniform spread, spread_width α_{width} shall be equal to 0,0° for non-uniform spread).

The following restrictions apply to binaural rendering:

The value of bsBinauralDataFormatID in BinauralRendering() should be set to 1 (if the FD Binaural renderer is implemented) or to 2 (if the TD Binaural renderer is implemented). The value of bsBinauralDataFormatID can be set to 0 if the parameterization of binaural room impulse responses according to subclauses 13.2.3 or 13.3.3 is implemented.

The number of BRIR sets is restricted to a maximum number of 3.

In case of H2B filters, the number of BRIR filter pairs to be provided shall correspond to 'Maximum H2B filter order' column in Table 9. In the other cases, the following applies.

The number of BRIR pairs in each BRIR set shall correspond to the number indicated in the relevant level-dependent row of Table 9. The measured BRIR positions shall correspond to all nominal geometric positions corresponding to the list of LoudspeakerGeometry indices in Table 9. The correspondence between LoudspeakerGeometry index and nominal geometric position is defined in ISO/IEC 23001-8. Thereby, it is ensured that one BRIR pair is available for each possible regular input channel configuration that can be used within the indicated level.

An input channel configuration is regular if it is defined by means of an ISO/IEC 23001-8 ChannelConfiguration or a list of ISO/IEC 23001-8 LoudspeakerGeometry (CICPspeakerIdx).

If binaural rendering is activated, the measured BRIR positions shall be passed to the mpegh3daLocalSetupInformation(). Thus, all renderer stages are set to the target layout that is equal to the transmitted channel configuration. As one BRIR is available per regular input channel, the format converter can be passed through in case regular input channel positions are used.

Table 9 — The binaural restrictions for the low complexity profile

Level	Number of BRIR pairs	Maximum H2B filter order	BRIR positions by means of loudspeaker position abbreviation	BRIR positions by means of loudspeakergeometry according to ISO/IEC 23001-8
1	3	1	L, R, C	0, 1, 2
2	10	2	L, R, C, LS, Rs, Lc, Rc, Lsr, Rsr, Cs	0, 1, 2, 4, 5, 8, 9, 10, 15, 16
3	21	3	L, R, C, Ls, Rs, Lc, Rc, Lsr, Rsr, Cs, Lss, Rss, Lv, Rc, Cv, Lvr, Rvr, Cvr, Rs, Lvs, Rvs	0, 1, 2, 4, 5, 8, 9, 10, 13, 14,15, 16, 17, 18, 19, 20, 21, 22, 25, 30, 31
4	28	5	L, R, C, Ls, Rs, Lc, Rc, Lsr, Rsr, Cs, Lss, Rss, Lv, Rc, Cv, Lvr, Rvr, Cvr, Lvss, Rvss, Ts, Lb, Rb, Cb, Lvs, Rvs, Lbs, Rbs,	0, 1, 2, 4, 5, 8, 9, 10, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 30, 31, 37, 38
5	28	6	L, R, C, Ls, Rs, Lc, Rc, Lsr, Rsr, Cs, Lss, Rss, Lv, Rc, Cv, Lvr, Rvr, Cvr, Lvss, Rvss, Ts, Lb, Rb, Cb, Lvs, Rvs, Lbs, Rbs,	0, 1, 2, 4, 5, 8, 9, 10, 13, 14,15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 30, 31, 37, 38

The following additional parameter values restrictions apply.

— The value of kMax in FdBinauralRendererParam() shall be equal to or less than 48 (bands).

— The value of kConv in FdBinauralRendererParam() shall be equal to 32.

— The values of rt60[k] in SfrBrirParam() shall be less than or equal to 1,0 (sec).

— The average of the values of nFilter[k] shall be less than or equal to 64.

— The values of nFilter[k] in VoffBrirParam() should be less than or equal to 256.

The following coding tools, modules, or features shall not be employed:

— Time warped filterbank;

— 768 sample outputFrameLength, i.e. coreSbrFrameLengthIndex shall not be 0.

The following text describes restrictions dependent on the length of the arithmetic coder codeword, arith_data(). For this text the following definitions apply.

F_{sOut}	core coder sampling rate as indicated by means of usacSamplingFrequencyIndex or usacSamplingFrequency in mpegh3daConfig()
F_{sMax}	maximum allowed sampling rate of a given level in this profile
N_{chMax}	maximum number of decoder processed core channels of a given level in this profile according to Table 3.
N_{chLtpf}	number of core coder channels in which the long term post filter (LTPF) is applied
N_{chInf}	number of core coder channels in which the independent noise filling (INF) is applied
$Nbits_{arith\_data()}(ch)$	number of bits used for arithmetic coding of spectral data, arith_data(), for core coder channel ch for a given frame

$Nbits_{arith\_all} = \sum_{\text{all channels}} Nbits_{\text{arith\_data()}}(ch)$, i.e. the sum of all bits used for the arithmetic coding of spectral data of all core coder channel

— In any given audio frame $Nbits_{arith\_all}$ shall comply with the following restriction:

$$Nbits_{arith\_all} < \frac{(3072 \cdot N_{chMax} - 2048 \cdot N_{chLtpf} - 2048 \cdot N_{chInf}) \cdot F_{sMax}}{F_{sOut}}$$

The following restrictions apply to the AudioPreRoll() extension.

— Decoders conforming to this profile shall support the full decoding and correct handling of the AudioPreRoll() extension.

— The number of pre-roll frames, numPreRollFrames, in an AudioPreRoll() extension payload shall not exceed 1 (one).

— In access units that are embedded as pre-roll in an AudioPreRoll() extension the usacExtElementPresent field for extensions of type ID_EXT_ELE_AUDIOPREROLL shall be 0.

The following restrictions apply to the employed sampling rate and the resampler block.

— The sampling rate that is signalled by means of usacSamplingFrequencyIndex as defined in ISO/IEC 23003-3:2012, Table 67 or usacSamplingFrequency shall be one of the values in the first column of Table 10.

— Depending on the above mentioned sampling rate and the profile level the resampler may employ one of the resampling ratios indicated in Table 10.

Table 10 — Allowed sampling rates and resampling ratios

Allowed sampling rate	Allowed resampling ratio depending on Mpegh3daProfileLevelIndication				
	Lvl 1	Lvl 2	Lvl 3	Lvl 4	Lvl 5
96 000	N/A	N/A	N/A	N/A	1
88 200	N/A	N/A	N/A	N/A	1
64 000	N/A	N/A	N/A	N/A	1.5
58 800	N/A	N/A	N/A	N/A	1.5
48 000	1	1	1	1	1 or 2
44 100	1	1	1	1	1 or 2
32 000	1.5	1.5	1.5	1.5	1.5 or 3
29 400	1.5	1.5	1.5	1.5	1.5 or 3
24 000	2	2	2	2	2
22 050	2	2	2	2	2
16 000	3	3	3	3	3
14 700	3	3	3	3	3

The following restrictions apply to the coding of the audio scene information structure.

Table 11 — ProfileLevel dependent restrictions to selected fields of the mae_AudioSceneInfo()

Restriction applies to	Allowed maximum value depending on Mpegh3daProfileLevelIndication				
	Lvl 1	Lvl 2	Lvl 3	Lvl 4	Lvl 5
mae_numGroups	5	9	16	28	28
mae_numSwitchGroups	2	4	8	14	14

Restriction applies to	Allowed maximum value depending on Mpegh3daProfileLevelIndication				
	Lvl 1	Lvl 2	Lvl 3	Lvl 4	Lvl 5
mae_numGroupPresets	4	4	8	16	31
(mae_bsGroupPresetNumConditions +1)	5	9	16	16	16
mae_numDownmixIdGroupPresetExtensions per mae_groupPresetID	4	4	8	16	31
(mae_bsNumDescLanguages +1)	4	4	4	8	16
(mae_bsDescriptionDataLength +1)	256	256	256	256	256

— If mae_numSwitchGroups > 0, then elementLengthPresent shall be set to 1.

The following restriction applies to the usage of the contribution mode.

— For end user devices the contribution mode of MPEG-H 3D audio as defined in subclause 4.7 shall not be supported.

4.8.2.3 Levels of the main profile

ISO/IEC 23008-3:2015 (the first edition of this document) captured the definition of the main profile, its associated bitstream syntax, semantics, and decoding process description.

4.8.2.4 Levels of the high profile

Currently blank — Placeholder for high profile.

5 MPEG-H 3D audio core decoder

5.1 Definitions

5.1.1 Joint stereo

The MDCT domain-based joint stereo coding tool with the possibility of complex stereo prediction is as defined in ISO/IEC 23003-3:2012, subclause 7.7.

5.1.2 MPEG surround based stereo (MPS 212)

The MPEG surround 2-1-2 based stereo tool working in QMF domain is as defined in ISO/IEC 23003-3:2012, subclause 7.11, with the possibility of using residual coding (unified stereo) as specified in ISO/IEC 23003-3:2012, B.21.

5.2 Syntax

5.2.1 General

The bitstream syntax is based on ISO/IEC 23003-3:2012, Clause 5.

Modifications and amendments to the existing bitstream syntax are listed below.

In environments that require byte alignment, MPEG-H 3D audio configuration elements or payload elements that are not an integer number of bytes in length are padded at the end to achieve an integer byte count.

5.2.2 Decoder configuration

5.2.2.1 General configuration syntax

Table 12 — Syntax of mpegh3daConfig()

Syntax	No. of bits	Mnemonic
mpegh3daConfig()		
{		
mpegh3daProfileLevelIndication	**8**	**uimsbf**
usacSamplingFrequencyIndex;	**5**	**bslbf**
if (usacSamplingFrequencyIndex == 0x1f) {		
usacSamplingFrequency;	**24**	**uimsbf**
}		
coreSbrFrameLengthIndex;	**3**	**uimsbf**
cfg_reserved;	**1**	**uimsbf**
receiverDelayCompensation;	**1**	**uimsbf**
referenceLayout = SpeakerConfig3d();		
FrameworkConfig3d();		
mpegh3daDecoderConfig();		
if (**usacConfigExtensionPresent**) {	**1**	**uimsbf**
mpegh3daConfigExtension();		
}		
}		

Table 13 — Syntax of FrameworkConfig3d()

Syntax	No. of bits	Mnemonic
FrameworkConfig3d()		
{		
Signals3d();		
}		

Table 14 — Syntax of Signals3d()

Syntax	No. of bits	Mnemonic
Signals3d()		
{		
numAudioChannels = 0;		
numAudioObjects = 0;		
numSAOCTransportChannels = 0;		
numHOATransportChannels = 0;		
bsNumSignalGroups;	**5**	**uimsbf**
for (grp = 0; grp < bsNumSignalGroups + 1 ; grp++) {		
signal_groupID[grp] = grp;		
differsFromReferenceLayout[grp] = 0;		
signalGroupType[grp]**;**	**3**	**bslbf**
bsNumberOfSignals[grp] = escapedValue(5, 8, 16);		
if (SignalGroupType[grp] == SignalGroupTypeChannels) {		
numAudioChannels += bsNumberOfSignals[grp] + 1;		

Syntax	No. of bits	Mnemonic
differsFromReferenceLayout[grp];	**1**	**bslbf**
if(differsFromReferenceLayout[grp]) {		
audioChannelLayout[grp] = SpeakerConfig3d();		
}		
else {		
audioChannelLayout[grp] = referenceLayout;		
}		
}		
if (SignalGroupType[grp] == SignalGroupTypeObject) {		
numAudioObjects += bsNumberOfSignals[grp] + 1;		
}		
if (SignalGroupType[grp] == SignalGroupTypeSAOC) {		
numSAOCTransportChannels += bsNumberOfSignals[grp] + 1;		
saocDmxLayoutPresent;	**1**	**bslbf**
if (saocDmxLayoutPresent == 1) {		
saocDmxChannelLayout = SpeakerConfig3d();		
}		
}		
if (SignalGroupType[grp] == SignalGroupTypeHOA) {		
numHOATransportChannels += bsNumberOfSignals[grp] + 1;		
}		
}		
}		

5.2.2.2 Loudspeaker configuration syntax

Table 15 — Syntax of SpeakerConfig3d()

Syntax	No. of bits	Mnemonic
SpeakerConfig3d()		
{		
speakerLayoutType;	**2**	**uimsbf**
if (speakerLayoutType == 0) {		
CICPspeakerLayoutIdx;	**6**	**uimsbf**
}		
else {		
numSpeakers = escapedValue(5, 8, 16) + 1;		
if (speakerLayoutType == 1) {		
for (i = 0; i < numSpeakers; i++) {		
CICPspeakerIdx;	**7**	**uimsbf**
}		
}		
if (speakerLayoutType == 2) {		
mpegh3daFlexibleSpeakerConfig(numSpeakers);		
}		
}		
}		

Table 16 — Syntax of mpegh3daFlexibleSpeakerConfig()

Syntax	No. of bits	Mnemonic
mpegh3daFlexibleSpeakerConfig(numSpeakers) { **angularPrecision**;	**1**	**uimsbf**

for (i = 0; i < numSpeakers; i++) {
  mpegh3daSpeakerDescription();
  if ((AzimuthAngle != 0°) && (AzimuthAngle != 180°)) {
		[a]

alsoAddSymmetricPair;	**1**	**uimsbf**

if (alsoAddSymmetricPair) {
  (*also add the loudspeaker with the opposite AzimuthDirection*);
  i++;
  }
  }
  }
}		

[a] The value of AzimuthAngle can be calculated using Table 17.

Table 17 — Syntax of mpegh3daSpeakerDescription()

Syntax	No. of bits	Mnemonic
mpegh3daSpeakerDescription() { **isCICPspeakerIdx**	**1**	**uimsbf**
if (isCICPspeakerIdx) { **CICPspeakerIdx**;	**7**	**uimsbf**

}
  else {
  ElevationClass;
	2	**uimsbf**

if (ElevationClass == 3) { **ElevationAngleIdx**;	**5, 7**	**uimsbf**
if (ElevationAngle != 0°) {		[a]
ElevationDirection;	**1**	**uimsbf**

}
  }
  AzimuthAngleIdx;
	6, 8	**uimsbf**

if ((AzimuthAngle != 0°) && (AzimuthAngle != 180°)) {		[a]
AzimuthDirection;	**1**	**uimsbf**
} **isLFE;**	**1**	**uimsbf**
} }		

[a] The number of bits for ElevationAngleIdx and AzimuthAngleIdx depends on the value of angularPrecision according to Table 67. The value of ElevationAngle and AzimuthAngle can be derived from Table 69 and Table 71, respectively. In case isCICPspeakerIdx is one, ElevationAngle and AzimuthAngle shall be signalled by means of a LoudspeakerGeometry as defined in ISO/IEC 23001-8.

5.2.2.3 Core decoder configuration

Table 18 — Syntax of mpegh3daDecoderConfig()

Syntax	No. of bits	Mnemonic

mpegh3daDecoderConfig()
{
numElements = escapedValue(4,8,16) + 1;
elementLengthPresent
 	
for (elemIdx=0; elemIdx<numElements; ++elemIdx) {
usacElementType[elemIdx**]**
switch (usacElementType[elemIdx]) {
case ID_USAC_SCE:
mpegh3daSingleChannelElementConfig(sbrRatioIndex);
break;
case ID_USAC_CPE:
mpegh3daChannelPairElementConfig(sbrRatioIndex);
break;
case ID_USAC_LFE:
mpegh3daLfeElementConfig();
break;
case ID_USAC_EXT:
mpegh3daExtElementConfig();
break;
}
}
	1
	
	2
		uimsbf

NOTE mpegh3daSingleChannelElementConfig(), mpegh3daChannelPairElementConfig(), mpegh3daLfeElementConfig() and mpegh3daExtElementConfig() signalled at position elemIdx refer to the corresponding elements in mpegh3daFrame() at the respective position elemIdx.		

Table 19 — Syntax of mpegh3daSingleChannelElementConfig()

Syntax	No. of bits	Mnemonic

mpegh3daSingleChannelElementConfig(sbrRatioIndex)
{
mpegh3daCoreConfig();
if (sbrRatioIndex > 0) {
SbrConfig();
}
}		

Table 20 — Syntax of mpegh3daChannelPairElementConfig()

Syntax	No. of bits	Mnemonic

mpegh3daChannelPairElementConfig(sbrRatioIndex)
{
mpegh3daCoreConfig();
if (enhancedNoiseFilling) {
igfIndependentTiling;
}
if (sbrRatioIndex > 0) {
SbrConfig();
stereoConfigIndex;
} else {
stereoConfigIndex = 0;
	1
	
	2
		bslbf
		
		uimsbf

Syntax	No. of bits	Mnemonic
}		
if (stereoConfigIndex > 0) {		
Mps212Config(stereoConfigIndex);		
}		
qceIndex;	**2**	**Uimsbf**
if (qceIndex > 0) {		
shiftIndex0;	**1**	**uimsbf**
if(shiftIndex0 > 0) {		
shiftChannel0;	**nBits** [a]	
}		
}		
shiftIndex1;	**1**	**uimsbf**
if (shiftIndex1 > 0) {		
shiftChannel1;	**nBits** [a]	
}		
if (sbrRatioIndex == 0 && qceIndex == 0) {		
lpdStereoIndex;	**1**	**bslbf**
} else {		
lpdStereoIndex = 0		
}		
}		

[a] **nBits** = floor(log2(numAudioChannels + numAudioObjects + numHOATransportChannels + numSAOCTransportChannels - 1)) + 1.

Table 21 — Syntax of mpegh3daCoreConfig()

Syntax	No. of bits	Mnemonic
mpegh3daCoreConfig()		
{		
tw_mdct;	**1**	**bslbf**
fullbandLpd;	**1**	**bslbf**
noiseFilling;	**1**	**bslbf**
if **(enhancedNoiseFilling) {**	**1**	**bslbf**
igfUseEnf;	**1**	**bslbf**
igfUseHighRes;	**1**	**bslbf**
igfUseWhitening;	**1**	**bslbf**
igfAfterTnsSynth;	**1**	**bslbf**
igfStartIndex;	**5**	**uimsbf**
igfStopIndex;	**4**	**uimsbf**
}		
}		

Table 22 — Syntax of mpegh3daLfeElementConfig()

Syntax	No. of bits	Mnemonic
mpegh3daLfeElementConfig()		
{		
tw_mdct = 0;		
fullbandLpd = 0;		
noiseFilling = 0;		
enhancedNoiseFilling = 0;		
}		

Table 23 — Syntax of mpegh3daExtElementConfig()

Syntax	No. of bits	Mnemonic
mpegh3daExtElementConfig()		
{		
usacExtElementType = escapedValue(4, 8, 16);		
usacExtElementConfigLength = escapedValue(4, 8, 16);		
if (usacExtElementDefaultLengthPresent) {	**1**	**uimsbf**
usacExtElementDefaultLength = escapedValue(8, 16, 0) + 1;		
} else {		
usacExtElementDefaultLength = 0;		
}		
usacExtElementPayloadFrag;	**1**	**uimsbf**
switch (usacExtElementType) {		
case ID_EXT_ELE_FILL:		
/* No configuration element */		
break;		
case ID_EXT_ELE_MPEGS:		
SpatialSpecificConfig();		
break;		
case ID_EXT_ELE_SAOC:		
SAOCSpecificConfig();		
break;		
case ID_EXT_ELE_AUDIOPREROLL:		
/* No configuration element */		
break;		
case ID_EXT_ELE_UNI_DRC:		
mpegh3daUniDrcConfig();		
break;		
case ID_EXT_ELE_OBJ_METADATA:		
ObjectMetadataConfig();		
break;		
case ID_EXT_ELE_SAOC_3D:		
SAOC3DSpecificConfig();		
break;		
case ID_EXT_ELE_HOA:		
HOAConfig();		
break;		
case ID_EXT_ELE_FMT_CNVRTR		
/* No configuration element */		
break;		
case ID_EXT_ELE_MCT:		
MCTConfig();		
break;		
case ID_EXT_ELE_TCC:		
TccConfig();		
break;		
case ID_EXT_ELE_HOA_ENH_LAYER:		
HOAEnhConfig();		
break;		

Syntax	No. of bits	Mnemonic
case ID_EXT_ELE_HREP:		
HREPConfig(current_signal_group);		
break;		
case ID_EXT_ELE_ENHANCED_OBJ_METADATA:		
EnhancedObjectMetadataConfig();		
break;		
default:	[a]	
while (usacExtElementConfigLength--) {		
tmp;	**8**	**uimsbf**
}		
break;		
}		
}		

[a] The default entry for the usacExtElementType is used for unknown extElementTypes so that legacy decoders can cope with future extensions.

Table 24 — Syntax of mpegh3daConfigExtension()

Syntax	No. of bits	Mnemonic
mpegh3daConfigExtension()		
{		
numConfigExtensions = escapedValue(2,4,8) + 1;		
for (confExtIdx=0; confExtIdx<numConfigExtensions; confExtIdx++) {		
usacConfigExtType[confExtIdx] = escapedValue(4,8,16);		
usacConfigExtLength[confExtIdx] = escapedValue(4,8,16);		
switch (usacConfigExtType[confExtIdx]) {		
case ID_CONFIG_EXT_FILL:		
while (usacConfigExtLength[confExtIdx]--) {		
fill_byte[i]; /* should be '10100101' */	**8**	**uimsbf**
}		
break;		
case ID_CONFIG_EXT_DOWNMIX:		
downmixConfig();		
break;		
case ID_CONFIG_EXT_LOUDNESS_INFO:		
mpegh3daLoudnessInfoSet();		
break;		
case ID_CONFIG_EXT_AUDIOSCENE_INFO:		
mae_AudioSceneInfo();		
break;		
case ID_CONFIG_EXT_HOA_MATRIX:		
HoaRenderingMatrixSet();		
break;		
case ID_CONFIG_EXT_ICG:		
ICGConfig();		
break;		
case ID_CONFIG_EXT_SIG_GROUP_INFO:		
SignalGroupInformation();		
break;		
default:		
while (usacConfigExtLength[confExtIdx]--) {		

Syntax	No. of bits	Mnemonic
tmp;	**8**	**uimsbf**
}		
break;		
}		
}		
}		

Table 25 — References to USAC configuration syntactic elements

Syntax of	shall be as defined in
escapedValue()	ISO/IEC 23003-3
SbrConfig()	ISO/IEC 23003-3
Mps212Config()	ISO/IEC 23003-3
SAOCSpecificConfig()	ISO/IEC 23003-2
SpatialSpecificConfig()	ISO/IEC 23003-1

5.2.2.4 Syntax of downmix matrix elements

Table 26 — Syntax of downmixConfig()

Syntax	No. of bits	Mnemonic
downmixConfig ()		
{		
downmixConfigType;	**2**	**uimsbf**
if (downmixConfigType == 0 || downmixConfigType == 2) {		
passiveDownmixFlag	**1**	**uimsbf**
if (passiveDownmixFlag ==0) {		
phaseAlignStrength	**3**	**uimsbf**
}		
immersiveDownmixFlag	**1**	**uimsbf**
}		
if (downmixConfigType == 1 || downmixConfigType == 2) {		
DownmixMatrixSet()		
}		
}		

Table 27 — Syntax of DownmixMatrixSet()

Syntax	No. of bits	Mnemonic
DownmixMatrixSet()		
{		
downmixIdCount;	**5**	**uimsbf**
for (k=0; k< downmixIdCount; ++k) {		
downmixId;	**7**	**uimsbf**
downmixType;	**2**	**uimsbf**
if (downmixType == 0) {		
CICPspeakerLayoutIdx;	**6**	**uimsbf**
} else if (downmixType == 1) {		
CICPspeakerLayoutIdx;	**6**	**uimsbf**
bsDownmixMatrixCount[k] = escapedValue(1,3,0);	**1..4**	
for (l=0; l< bsDownmixMatrixCount[k]+1; ++l) {		
bsNumAssignedGroupIDs[k][l] = escapedValue(1,4,4);	**1..9**	
for (m=0; m<bsNumAssignedGroupIDs[k][l]+1; ++m) {		
signal_groupID[k][l][m]**;**	**5**	**uimsbf**
}		

Syntax	No. of bits	Mnemonic
DmxMatrixLenBits = escapedValue(8,8,12);	**8..28**	
DownmixMatrix(inputConfig(audioChannelConfig), inputCount(audioChannelConfig), outputConfig(CICPspeakerLayoutIdx), outputCount(CICPspeakerLayoutIdx));	**DmxMatrixLenBits**	
}		
}		
}		
}		

Table 28 — Syntax of DownmixMatrix()

Syntax	No. of bits	Mnemonic
DownmixMatrix(inputConfig, inputCount, outputConfig, outputCount)		
{		
equalizerPresent;	**1**	**uimsbf**
if (equalizerPresent) {		
EqualizerConfig (inputConfig, inputCount);		
}		
precisionLevel;	**2**	**uimsbf**
maxGain = escapedValue(3, 4, 0);		
minGain = (-1) · (escapedValue(4, 5, 0) + 1);		
(compactInputConfig, compactInputCount) = ConvertToCompactConfig(inputConfig, inputCount);		
(compactOutputConfig, compactOutputCount) = ConvertToCompactConfig(outputConfig, outputCount);		
isAllSeparable;	**1**	**uimsbf**
if (!isAllSeparable) {		
for (i = 0; i < compactOutputCount; i++) {		
if (compactOutputConfig[i].pairType == SYMMETRIC) {		
isSeparable[i];	**1**	**uimsbf**
}		
}		
} else {		
for (i = 0; i < compactOutputCount; i++) {		
if (compactOutputConfig[i].pairType == SYMMETRIC) {		
isSeparable[i] = 1;		
}		
}		
}		
isAllSymmetric;	**1**	**uimsbf**
if (!isAllSymmetric) {		
for (i = 0; i < compactOutputCount; i++) {		
isSymmetric[i];	**1**	**uimsbf**
}		
} else {		
for (i = 0; i < compactOutputCount; i++) {		
isSymmetric[i] = 1;		
}		
}		

Syntax	No. of bits	Mnemonic
mixLFEOnlyToLFE;	**1**	**uimsbf**
rawCodingCompactMatrix;	**1**	**uimsbf**
if (rawCodingCompactMatrix) {		
for (i = 0; i < compactInputCount; i++) {		
for (j = 0; j < compactOutputCount; j++) {		
if (!mixLFEOnlyToLFE || (compactInputConfig[i].isLFE == compactOutputConfig[j].isLFE)) {		
compactDownmixMatrix[i][j];	**1**	**uimsbf**
} else {		
compactDownmixMatrix[i][j] = 0;		
}		
}		
}		
} else {		
totalCount = CalculateTotalCount();		
useCompactTemplate;	**1**	**uimsbf**
nBits = 3; if (totalCount >= 256) nBits = 4;		
runLGRParam;	**nBits**	**uimsbf**
count = 0;		
while (count < totalCount) {		
zeroRunLength;	**varies**	**bslbf**
/* limited Golomb-Rice using p = runLGRparam and N = totalCount+1*/		
flatCompactMatrix[count .. count + zeroRunLength] = {0, ..., 0, 1};		
count += zeroRunLength + 1;		
}		
count = 0;		
compactTemplate = FindCompactTemplate(inputConfig, inputCount, outputConfig, outputCount);		
for (i = 0; i < compactInputCount; i++) {		
for (j = 0; j < compactOutputCount; j++) {		
if (mixLFEOnlyToLFE && compactInputConfig[i].isLFE && compactOutputConfig[j].isLFE) {		
compactDownmixMatrix[i][j];	**1**	**uimsbf**
} else if (mixLFEOnlyToLFE && (compactInputConfig[i].isLFE || compactOutputConfig[j].isLFE)) {		
compactDownmixMatrix[i][j] = 0;		
} else {		
compactDownmixMatrix[i][j] = flatCompactMatrix[count++];		
if (useCompactTemplate) {		
compactDownmixMatrix[i][j] ^= compactTemplate[i][j];		
}		
}		
}		
}		
fullForAsymmetricInputs;	**1**	**uimsbf**
rawCodingNonzeros;	**1**	**uimsbf**
if (!rawCodingNonzeros) {		
gainLGRParam;	**3**	**uimsbf**
gainTableSize = generateGainTable(maxGain, minGain, precisionLevel);		
}		

Syntax	No. of bits	Mnemonic

```
  for (i = 0; i < compactInputCount; i++) {
    iType = compactInputConfig[i].pairType;
    for (j = 0; j < compactOutputCount; j++) {
      oType = compactOutputConfig[j].pairType;
      i1 = compactInputConfig[i].originalPosition;
      o1 = compactOutputConfig[j].originalPosition;

      if ((iType != SYMMETRIC) && (oType != SYMMETRIC)) {
        downmixMatrix[i1][o1] = 0.0;
        if (!compactDownmixMatrix[i][j]) continue;

        downmixMatrix[i1][o1] = DecodeGainValue();
      } else if (iType != SYMMETRIC) {
        o2 = compactOutputConfig[j].SymmetricPair.originalPosition;
        downmixMatrix[i1][o1] = 0.0;
        downmixMatrix[i1][o2] = 0.0;
        if (!compactDownmixMatrix[i][j]) continue;

        downmixMatrix[i1][o1] = DecodeGainValue();
        useFull = (iType == ASYMMETRIC) &&
fullForAsymmetricInputs;
        if (isSymmetric[j] && !useFull) {
          downmixMatrix[i1][o2] = downmixMatrix[i1][o1];
        } else {
          downmixMatrix[i1][o2] = DecodeGainValue();
        }
      } else if (oType != SYMMETRIC) {
        i2 = compactInputConfig[i].SymmetricPair.originalPosition;
        downmixMatrix[i1][o1] = 0.0;
        downmixMatrix[i2][o1] = 0.0;
        if (!compactDownmixMatrix[i][j]) continue;

        downmixMatrix[i1][o1] = DecodeGainValue();
        if (isSymmetric[j]) {
          downmixMatrix[i2][o1] = downmixMatrix[i1][o1];
        } else {
          downmixMatrix[i2][o1] = DecodeGainValue();
        }
      } else {
        i2 = compactInputConfig[i].SymmetricPair.originalPosition;
        o2 = compactOutputConfig[j].SymmetricPair.originalPosition;
        downmixMatrix[i1][o1] = 0.0;
        downmixMatrix[i1][o2] = 0.0;
        downmixMatrix[i2][o1] = 0.0;
        downmixMatrix[i2][o2] = 0.0;
        if (!compactDownmixMatrix[i][j]) continue;

        downmixMatrix[i1][o1] = DecodeGainValue();
        if (isSeparable[j] && isSymmetric[j]) {
          downmixMatrix[i2][o2] = downmixMatrix[i1][o1];
        } else if (!isSeparable[j] && isSymmetric[j]) {
          downmixMatrix[i1][o2] = DecodeGainValue();
          downmixMatrix[i2][o1] = downmixMatrix[i1][o2];
          downmixMatrix[i2][o2] = downmixMatrix[i1][o1];
        } else if (isSeparable[j] && !isSymmetric[j]) {
```

Syntax	No. of bits	Mnemonic

downmixMatrix[i2][o2] = DecodeGainValue();
} else {
downmixMatrix[i1][o2] = DecodeGainValue();
downmixMatrix[i2][o1] = DecodeGainValue();
downmixMatrix[i2][o2] = DecodeGainValue();
}
}
}
}
}		

Table 29 — Syntax of DecodeGainValue()

Syntax	No. of bits	Mnemonic

DecodeGainValue()
{
if (rawCodingNonzeros) {
nAlphabet = ((maxGain - minGain) * 2 ^ precisionLevel) + 2;
gainValueIndex = ReadRange(nAlphabet);
gainValue = maxGain - gainValueIndex / 2 ^ precisonLevel;
} else {
gainValueIndex;
/* limited Golomb-Rice using p = gainLGRParam and N = gainTableSize*/
gainValue = gainTable[gainValueIndex];
}
if (gainValue < minGain) gainValue = -infinity;
return gainValue;
}
	varies	**bslbf**

Table 30 — Syntax of ReadRange()

Syntax	No. of bits	Mnemonic

ReadRange(alphabetSize)
{
nBits = floor(log2(alphabetSize));
nUnused = 2 ^ (nBits + 1) - alphabetSize;
range;
	nBits	**uimsbf**

if (range >= nUnused) { **rangeExtra**;	**1**	**uimsbf**

range = range * 2 - nUnused + rangeExtra;
}
return range;
}		

Table 31 — Syntax of EqualizerConfig()

Syntax	No. of bits	Mnemonic
EqualizerConfig(inputConfig, inputCount)		
{		
numEqualizers = escapedValue(3, 5, 0) + 1;		
eqPrecisionLevel;	**2**	**uimsbf**
eqExtendedRange;	**1**	**uimsbf**
for (i = 0; i < numEqualizers; i++) {		
numSections = escapedValue(2, 4, 0) + 1;		
lastCenterFreqP10 = 0;		
lastCenterFreqLd2 = 10;		
maxCenterFreqLd2 = 99;		
for (j = 0; j < numSections; j++) {		
centerFreqP10 = lastCenterFreqP10 + ReadRange(4 - lastCenterFreqP10);		
if (centerFreqP10 > lastCenterFreqP10) { lastCenterFreqLd2 = 10;}		
if (centerFreqP10 == 3) { maxCenterFreqLd2 = 24; }		
centerFreqLd2 = lastCenterFreqLd2 +		
ReadRange(1 + maxCenterFreqLd2 - lastCenterFreqLd2);		
lastCenterFreqP10 = centerFreqP10;		
lastCenterFreqLd2 = centerFreqLd2;		
qFactorIndex;	**5**	**uimsbf**
if (qFactorIndex > 19) {		
qFactorExtra;	**3**	**uimsbf**
}		
cgBits = 4 + eqExtendedRange + eqPrecisionLevel;		
centerGainIndex;	**cgBits**	**uimsbf**
}		
sgBits = 4 + eqExtendedRange + min(eqPrecisionLevel + 1, 3);		
scalingGainIndex;	**sgBits**	**uimsbf**
}		
for (i = 0; i < inputCount; i++) {		
hasEqualizer[i];	**1**	**uimsbf**
if (hasEqualizer[i]) {		
equalizerIndex[i] = ReadRange(numEqualizers);		
} else {		
equalizerIndex[i] = -1;		
}		
}		
}		

5.2.2.5 Syntax of HOA matrix elements

Table 32 — Syntax of HoaRenderingMatrixSet()

Syntax	No. of bits	Mnemonic
HoaRenderingMatrixSet()		
{		
numOfHoaRenderingMatrices;	**5**	**uimsbf**
for (k=0; k< numOfHoaRenderingMatrices; ++k) {		
HoaRenderingMatrixId;	**7**	**uimsbf**

CICPspeakerLayoutIdx;	**6**	**uimsbf**
HoaMatrixLenBits = escapedValue(8,8,12);	**8..28**	
HoaRenderingMatrix(NumOfHoaCoeffs, outputConfig(CICPspeakerLayoutIdx), outputCount(CICPspeakerLayoutIdx));	**HoaMatrixLenBits**	
}		
}		

Table 33 — Syntax of HoaRenderingMatrix()

Syntax	No. of bits	Mnemonic
HoaRenderingMatrix(NumOfHoaCoeffs, outputConfig, outputCount)		
{		
lfeExist = 0;		
hasLfeRendering = 0;		
for (i=0; i< inputCount; ++i)		
isHoaCoefSparse[i] = 0;		
maxHoaOrder = sqrt(NumOfHoaCoeffs)-1;		
precisionLevel	**2**	**uimsbf**
isNormalized	**1**	**uimsbf**
if (**gainLimitPerHoaOrder**) {	**1**	**uimsbf**
for (i = 0; i<(maxHoaOrder+1); ++i) {		
maxGain[i] = - escapedValue(3, 5, 6);		
minGain[i] = -(escapedValue(4, 5, 6) + 1 – maxGain[i]);		
}		
} else {		
maxGain[0] = - escapedValue (3, 5, 6);		
minGain[0] = -(escapedValue (4, 5, 6) + 1- maxGain[0]);		
for (i = 1; i<(maxHoaOrder+1); ++i) {		
maxGain[i] = maxGain[0];		
minGain[i] = minGain[0];		
}		
}		
if (**isFullMatrix**==0) {	**1**	**uimsbf**
nbitsHoaOrder = ceil(log2(maxHoaOrder+1));		
firstSparseOrder	**nbitsHoaOrder**	**uimsbf**
for (i = (firstSparseOrder*firstSparseOrder); i<inputCount; ++i)		
isHoaCoefSparse[i] = 1;		
}		
for (i=0; i< outputCount; ++i){		
if (outputConfig[i].isLFE)		
lfeExist = 1;		
}		
if (lfeExist)		
hasLfeRendering;	**1**	**uimsbf**
numPairs = findSymmetricSpeakers(outputCount, outputConfig, hasLfeRendering);		
for (i=0; i<numPairs; ++i) {		
valueSymmetricPairs[i] = 0;		
signSymmetricPairs[i] = 0;		
}		
zerothOrderAlwaysPositive;	**1**	**uimsbf**
if (**isAllValueSymmetric**) {	**1**	**uimsbf**
for (i=0; i<numPairs; ++i) { valueSymmetricPairs[i] = 1; }		
} else {		

Syntax	No. of bits	Mnemonic
if (**isAnyValueSymmetric**) {	**1**	**uimsbf**
for (i=0; i<numPairs; ++i)		
valueSymmetricPairs[i] = **boolVal;**	**1**	**uimsbf**
if (**isAnySignSymmetric**) {	**1**	**uimsbf**
for (i=0; i<numPairs; ++i) {		
if (0==valueSymmetricPairs[i])		
signSymmetricPairs[i] = **boolVal;**	**1**	**uimsbf**
}		
}		
} else {		
if (**isAllSignSymmetric**) {	**1**	**uimsbf**
for (i=0; i<numPairs; ++i)		
signSymmetricPairs[i] = 1;		
} else {		
if (**isAnySignSymmetric**) {	**1**	**uimsbf**
for (i=0; i<numPairs; ++i)		
signSymmetricPairs[i] = **boolVal;**	**1**	**uimsbf**
}		
}		
}		
hasVerticalCoef;	**1**	**uimsbf**
DecodeHoaMatrixData()		
}		

Table 34 — Syntax of DecodeHoaMatrixData()

Syntax	No. of bits	Mnemonic
DecodeHoaMatrixData()		
{		
j = 0;		
for (i=0; i<outputCount; ++i) {		
isValueSymmetric[i] = 0;		
isSignSymmetric[i] = 0;		
if ((outputConfig[i].pairType == SP_PAIR_SYMMETRIC) &&		
(outputConfig[i].symmetricPair != NULL)) {		
if (0==(outputConfig[i].isLFE && (0==hasLfeRendering))) {		
isValueSymmetric[i] = valueSymmetricPairs[j];		
isSignSymmetric[i] = signSymmetricPairs[j++];		
}		
}		
}		
for (i = 0; i < inputCount; ++i) {		
currentHoaOrder = ceil(sqrt(i+1)-1);		
for (j = outputCount-1; j >= 0; --j) {		
signMatrix[i * outputCount + j] = 1;		
hoaMatrix [i * outputCount + j] = 0.0;		
if ((vertBitmask[i] && hasVerticalCoef) || !vertBitmask[i]) {		
hasValue = 1;		
if (0 == isValueSymmetric[j]) {		
if ((hasLfeRendering && outputConfig[j].isLFE) ||		
(!outputConfig[j].isLFE)) {		
if (isHoaCoefSparse[i]) {		
hasValue;	**1**	**uimsbf**
}		

Syntax	No. of bits	Mnemonic

if (hasValue) {
hoaMatrix [i * outputCount + j] =
DecodeHoaGainValue(currentHoaOrder);
if (0==isSignSymmetric[j]) {
if (hoaMatrix [i * outputCount + j] != 0.0) {
if (currentHoaOrder || !zerothOrderAlwaysPositive) {
signMatrix[i * outputCount + j] =
signVal*2-1;
	1	**uimsbf**

}
}
} else { // isSignSymmetric[i] == 1
pairIdx = outputConfig[j].symmetricPair->originalPosition;
signMatrix[i * outputCount + j] =
symSigns[i] * signMatrix[i * outputCount + pairIdx];
}
} /* has value */
}
}
else { /* isValueSymmetric */
pairIdx = outputConfig[j].symmetricPair->originalPosition;
hoaMatrix [i*outputCount+j] =
hoaMatrix [i*outputCount+pairIdx];
signMatrix[i *outputCount+j] =
symSigns[i] * signMatrix[i*outputCount+pairIdx];
}
}
}
}
for (i = 0; i < inputCount; ++i) {
for (j = 0; j < outputCount; ++j) {
hoaMatrix[i *outputCount+j] *= signMatrix[i*outputCount+j];
hoaMatrix[i *outputCount+j] /= sqrt(2 · ceil(sqrt(i+1)-1) + 1);
}
}
if(isNormalized) {
currentScalar = 0.0;
for (i = 0; i < inputCount; ++i) {
for (j = 0; j < outputCount; ++j) {
if (!outputConfig[j].isLFE)
currentScalar += hoaMatrix[i * outputCount + j] *
hoaMatrix[i * outputCount + j];
}
}
currentScalar = 1.0/sqrt(currentScalar);
for (i = 0; i < inputCount; ++i) {
for (j = 0; j < outputCount; ++j) {
if(!outputConfig[j].isLFE)
hoaMatrix[i * outputCount + j] *= currentScalar;
}
}
}		

Table 35 — Syntax of DecodeHoaGainValue()

Syntax	No. of bits	Mnemonic

DecodeHoaGainValue(order)
{
nAlphabet = ((maxGain[order] – minGain[order]) * 2 ^ precisionLevel) + 2;
gainValueIndex = ReadRange(nAlphabet);
gainValue = maxGain[order] - gainValueIndex / 2 ^ precisonLevel;
if (gainValue < minGain[order]) {
gainValue = 0.0;
} else {
gainValue = 10.0 ^ (gainValue / 20.0);
}
return gainValue;
}		

5.2.2.6 Syntax for internal channel processing configuration

The ICGConfig defines which type of processing is required in the internal channel processing block. For each CPE, one bit flag of **ICGPreAppliedCPE** is read if **ICGPreAppliedPresent** equals one.

Table 36 — Syntax of ICGConfig()

Syntax	No. of bits	Mnemonic

ICGConfig ()
{
if (**ICPresent**) {
	1	**Uimsbf**

for (elemIdx=0, elemCPE=0; elemIdx<numElements; ++elemIdx)
{
if (usacElementType[elemIdx] == ID_USAC_CPE)
{
ICinCPE[elemCPE];
	1	**Uimsbf**

elemCPE++;
}
}
if (**ICGPreAppliedPresent**) {
	1	**Uimsbf**

for (elemIdx=0, elemCPE=0; elemIdx<numElements; ++elemIdx)
{
if (usacElementType[elemIdx] == ID_USAC_CPE)
{
ICGPreAppliedCPE[elemCPE];
	1	**Uimsbf**

elemCPE++;
}
}
}
}
}		

ICPresent This field indicates whether at least one of CPEs has channel assignment capable of internal channel processings.

ICinCPE This field indicates whether each of CPEs shall use internal channel processings based on the paired channel information

ICGPreAppliedPresent This field indicates whether at least one CPE is encoded with the consideration of internal channel gain.

ICGPreAppliedCPE This field indicates whether each CPE is encoded with the consideration of internal channel gain.

5.2.2.7 Syntax of signal group information

Table 37 — Syntax of SignalGroupInformation()

Syntax	No. of bits	Mnemonic
SignalGroupInformation(bsNumSignalGroups)		
{		
for (grp = 0; grp < bsNumSignalGroups + 1 ; grp++) {		
groupPriority[grp];	**3**	**uimsbf**
fixedPosition[grp];	**1**	**uimsbf**
}		
}		

5.2.2.8 Extension element configurations

Table 38 — HREPConfig ()

Syntax	No. of bits	Mnemonic
HREPConfig(current_signal_group)		
{		
signal_type = signalGroupType[current_signal_group];		
signal_count = bsNumberOfSignals[current_signal_group] + 1;		
if (signal_type == SignalGroupTypeChannels) {		
channel_layout = audioChannelLayout[current_signal_group];		
}		
extendedGainRange;	**1**	**uimsbf**
extendedBetaFactorPrecision;	**1**	**uimsbf**
for (sig = 0; sig < signal_count; sig++) {	[a]	
if ((signal_type == SignalGroupTypeChannels) && *isLFEChannel*(channel_layout, sig)) {		
isHREPActive[sig] = 0;		
} else {		
isHREPActive[sig];	**1**	**uimsbf**
}		
if (isHREPActive[sig]) {		
if (sig == 0) {		
lastFFTLine[0];	**4**	**uimsbf**
transitionWidthLines[0];	**4**	**uimsbf**
defaultBetaFactorIdx[0];	**nBitsBeta**	**uimsbf**
} else {	[b]	
if (**useCommonSettings**) {	**1**	**uimsbf**
lastFFTLine[sig] = lastFFTLine[0];		
transitionWidthLines[sig] = transitionWidthLines[0];		
defaultBetaFactorIdx[sig] = defaultBetaFactorIdx[0];		
} else {		
lastFFTLine[sig];	**4**	**uimsbf**
transitionWidthLines[sig];	**4**	**uimsbf**

Syntax	No. of bits	Mnemonic
defaultBetaFactorIdx[sig];	**nBitsBeta**	**uimsbf**
}		
}		
}		
}		
}		

[a] The helper function *isLFEChannel*(channel_layout, sig) returns 1 if the channel on position *sig* in *channel_layout* is a LFE channel or 0 otherwise.
[b] nBitsBeta = 3 + extendedBetaFactorPrecision.

Table 39 — Syntax of MCTConfig()

Syntax [a]	No. of bits	Mnemonic
MCTConfig()		
{		
nMCTChannels = 0;		
for(chan=0;chan < bsNumberOfSignals[grp]+1; chan++) {		
mctChanMask[chan]**;**	**1**	uimsbf
if(mctChanMask[chan] > 0) {		
mctChannelMap[nMCTChannels]=chan;		
nMCTChannels++;		
}		
}		
}		

[a] The corresponding ID_USAC_EXT element shall be prior to any audio element of the corresponding signal group.

Table 40 — Syntax of TccConfig()

Syntax	No. of bits	Mnemonic
TccConfig()		
{		
for(elemIdx=0; elemIdx < numElements; ++elemIdx) {		
tccMode[elemIdx]**;**	**2** [a]	uimsbf
}		
}		

[a] tccMode is set only for SCE and CPE channel elements, so elemIdx should refer only to those elements.

5.2.3 MPEG-H 3D audio core bitstream payloads

5.2.3.1 Payloads for MPEG-H 3D audio core

The bitstream syntax is based on ISO/IEC 23003-3:2012, subclause 5.3.1. Modifications and amendments are listed below.

Table 41 — Syntax of mpegh3daFrame(), top level payload for MPEG-H 3D audio Core

Syntax	No. of bits	Mnemonic
mpegh3daFrame()		
{		

Syntax	No. of bits	Mnemonic
usacIndependencyFlag;	**1**	**uimsbf**
for (elemIdx=0; elemIdx<numElements; ++elemIdx) {		
if ((usacElementType[elemIdx] != ID_USAC_EXT) &&		
(elementLengthPresent == 1)) {		
elementLength	**16**	**uimsbf**
}		
switch (usacElementType[elemIdx]) {		
case ID_USAC_SCE:		
mpegh3daSingleChannelElement(usacIndependencyFlag);	**elementLength**, [a]	
break;		
case ID_USAC_CPE:		
mpegh3daChannelPairElement(usacIndependencyFlag);	**elementLength**, [a]	
break;		
case ID_USAC_LFE:		
mpegh3daLfeElement(usacIndependencyFlag);	**elementLength**, [a]	
break;		
case ID_USAC_EXT:		
mpegh3daExtElement(usacIndependencyFlag);		
break;		
}		
}		
[a] If present, elementLength represents the length of the corresponding element it refers to in number of bits.		

Table 42 — Syntax of mpegh3daSingleChannelElement()

Syntax	No. of bits	Mnemonic
mpegh3daSingleChannelElement(indepFlag)		
{		
mpegh3daCoreCoderData(1, indepFlag);		
if (sbrRatioIndex > 0) {		
UsacSbrData(1, indepFlag);		
}		
}		

Table 43 — Syntax of mpegh3daChannelPairElement()

Syntax	No. of bits	Mnemonic
mpegh3daChannelPairElement(indepFlag)		
{		
if (stereoConfigIndex == 1) {		
nrCoreCoderChannels = 1;		
} else {		
nrCoreCoderChannels = 2;		
}		
mpegh3daCoreCoderData(nrCoreCoderChannels, indepFlag);		
if (sbrRatioIndex > 0) {		
if (stereoConfigIndex == 0 || stereoConfigIndex == 3) {		
nrSbrChannels = 2;		

Syntax	No. of bits	Mnemonic

} else {
nrSbrChannels = 1;
}
UsacSbrData(nrSbrChannels, indepFlag);
}
 	
if (stereoConfigIndex > 0) {
Mps212Data(indepFlag);
}
}		

Table 44 — Syntax of mpegh3daLfeElement()

Syntax	No. of bits	Mnemonic

mpegh3daLfeElement(indepFlag)
{
fd_channel_stream(0, 0, 0, 0, 0, indepFlag);
}		

Table 45 — Syntax of mpegh3daExtElement()

Syntax	No. of bits	Mnemonic
mpegh3daExtElement(indepFlag) {		
usacExtElementPresent	**1**	**uimsbf**
if (usacExtElementPresent==1) {		
usacExtElementUseDefaultLength;	**1**	**uimsbf**
if (usacExtElementUseDefaultLength) {		
usacExtElementPayloadLength = usacExtElementDefaultLength;		
} else {		
usacExtElementPayloadLength;	**8**	**uimsbf**
if (usacExtElementPayloadLength == 255)		
valueAdd	**16**	**uimsbf**
usacExtElementPayloadLength += valueAdd - 2;		
}		
}		
if (usacExtElementPayloadLength>0) {		
if (usacExtElementPayloadFrag) {		
usacExtElementStart;	**1**	**uimsbf**
usacExtElementStop;	**1**	**uimsbf**
} else {		
usacExtElementStart = 1;		
usacExtElementStop = 1;		
}		
for (i=0; i<usacExtElementPayloadLength; i++) {		
usacExtElementSegmentData[i];	**8**	**uimsbf**
}		
}		
}		
}		

Table 46 — Syntax of ics_info()

Syntax	No. of bits	Mnemonic
ics_info()		
{		
window_sequence;	**2**	**uimsbf**
window_shape;	**1**	**uimsbf**
if (window_sequence == EIGHT_SHORT_SEQUENCE) {		
max_sfb;	**4**	**uimsbf**
scale_factor_grouping;	**7**	**uimsbf**
}		
else {		
max_sfb;	**6**	**uimsbf**
}		
}		

5.2.3.2 Subsidiary payloads

Table 47 — Syntax of mpegh3daCoreCoderData()

Syntax	No. of bits	Mnemonic
mpegh3daCoreCoderData(nrChannels, indepFlag)		
{		
for (ch=0; ch < nrChannels; ch++) {		
core_mode[ch]**;**	**1**	**uimsbf**
}		
if (nrChannels == 2) {		
StereoCoreToolInfo(core_mode);		
} else {		
common_ltpf = 0;		
}		
for (ch = 0; ch < nrChannels; ch++) {		
if (core_mode[ch] == 1) {		
if (lpdStereoIndex == 1 && ch == 1 && core_mode[0] == 1) {		
lpd_stereo_stream(indepFlag);		
} else {		
lpd_channel_stream(noiseFilling, fullbandLpd, indepFlag);		
}		
}		
else {		
if ((nrChannels == 1) || (core_mode[0] != core_mode[1])) {		
tns_data_present[ch];	**1**	**uimsbf**
}		
fd_channel_stream(common_window, common_tw,		
tns_data_present[ch], noiseFilling, fullbandLpd, indepFlag);		
}		
}		
}		

Table 48 — Syntax of StereoCoreToolInfo()

Syntax	No. of bits	Mnemonic
StereoCoreToolInfo(core_mode)		
{		
if (core_mode[0] == 0 && core_mode[1] == 0) {		
tns_active;	**1**	**uimsbf**
if (**common_window**) {	**1**	**uimsbf**
ics_info();		
if (**common_max_sfb**) {	**1**	**uimsbf**
max_sfb1 = max_sfb;		
} else {		
if (window_sequence == EIGHT_SHORT_SEQUENCE) {		
max_sfb1;	**4**	**uimsbf**
} else {		
max_sfb1;	**6**	**uimsbf**
}		
}		
max_sfb_ste = max(max_sfb, max_sfb1);		
if (enhancedNoiseFilling && !igfIndependentTiling) {		
max_sfb_ste = min(max_sfb_ste, igf_sfb_start);		
}		
ms_mask_present;	**2**	**uimsbf**
if (ms_mask_present == 1) {		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = 0; sfb < max_sfb_ste; sfb++) {		
ms_used[g][sfb];	**1**	**uimsbf**
}		
}		
}		
if (ms_mask_present == 3) {		
cplx_pred_data();		
} else {		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = 0; sfb < max_sfb_ste; sfb++) {		
alpha_q_re[g][sfb] = 0;		
alpha_q_im[g][sfb] = 0;		
}		
}		
}		
if (enhancedNoiseFilling && !igfIndependentTiling) {		
igf_ms_mask_present;	**2**	**uimsbf**
if (igf_ms_mask_present == 1) {		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = igf_sfb_start; sfb < igf_sfb_stop; sfb += (2 – igfUseHighRes)) {		
ms_used[g][sfb];	**1**	**uimsbf**
if (!igfUseHighRes && (sfb + 1) < igf_sfb_stop) {		
ms_used[g][sfb+1] = ms_used[g][sfb];		
}		
}		
}		
}		
if (igf_ms_mask_present == 3) {		
igf_stereo_pred_data();		
} else {		

Syntax	No. of bits	Mnemonic
for (g = 0; g < num_window_groups; g++) {		
for (sfb = max_sfb_ste; sfb < igf_sfb_stop; sfb++) {		
alpha_q_re[g][sfb] = 0;		
alpha_q_im[g][sfb] = 0;		
}		
}		
}		
}		
}		
if (tw_mdct) {		
if (**common_tw**) {	**1**	**uimsbf**
tw_data();		
}		
}		
if (**common_ltpf**) {	**1**	**uimsbf**
if (**ltpf_data_present**) {	**1**	**uimsbf**
ltpf_pitch_lag_index;	**9**	**uimsbf**
ltpf_gain_index;	**2**	**uimsbf**
}		
}		
if (tns_active) {		
if (common_window) {		
common_tns;	**1**	**uimsbf**
} else {		
common_tns = 0;		
}		
if (!enhancedNoiseFilling || igfAfterTnsSynth) {		
tns_on_lr;	**1**	**uimsbf**
} else {		
tns_on_lr = 1;		
}		
if (common_tns) {		
tns_data();		
tns_data_present[0] = 0;		
tns_data_present[1] = 0;		
} else {		
if (**tns_present_both**) {	**1**	**uimsbf**
tns_data_present[0] = 1;		
tns_data_present[1] = 1;		
} else {		
tns_data_present[1]**;**	**1**	**uimsbf**
tns_data_present[0] = 1 – tns_data_present[1];		
}		
}		
} else {		
common_tns = 0;		
tns_data_present[0] = 0;		
tns_data_present[1] = 0;		
}		
} else {		
common_window = 0;		
common_ltpf = 0;		
common_tw = 0;		
}		
}		

Table 49 — Syntax of lpd_stereo_stream()

Syntax	No. of bits	Mnemonic
lpd_stereo_stream(indepFlag)		
{		
res_mode;	**1**	**uimsbf**
q_mode;	**1**	**uimsbf**
ipd_mode;	**2**	**uimsbf**
pred_mode;	**1**	**uimsbf**
cod_mode;	**2**	**uimsbf**
nbands = band_config(N, res_mode);		
ipd_band_max = max_band[res_mode][ipd_mode];		
cod_band_max = max_band[res_mode][cod_mode];		
cod_L = 2*(band_limits[cod_band_max]-1);		
for (k = ccfl / M; k >= 0; k--) {		
if (q_mode == 0 || k % 2 == 1) {		
for (b = 0; b < nbands; b++) {		
ild_idx[k][b];	**5**	**uimsbf**
}		
for (b = 0; b < ipd_band_max; b++) {		
ipd_idx[k][b];	**3**	**uimsbf**
}		
if (pred_mode == 1) {		
for (b = cod_band_max; b < nbands;b++) {		
pred_gain_idx[k][b];	**3**	**uimsbf**
}		
}		
}		
}		
if (cod_mode >0) {		
cod_gain_idx[k];	**7**	**uimsbf**
for (i = 0; i < cod_L/8; i++) {		
code_book_indices(i, 1, 1);		
}		
}		
}		

Table 50 — Syntax of fd_channel_stream()

Syntax	No. of bits	Mnemonic
fd_channel_stream(common_window, common_tw, tns_data_present, noiseFilling, fullbandLpd, indepFlag)		
{		
global_gain;	**8**	**uimsbf**
if (noiseFilling) {		
noise_level;	**3**	**uimsbf**
noise_offset;	**5**	**uimsbf**
} else {		
noise_level = 0;		
}		
if (!common_window) {		
ics_info();		
}		

Syntax	No. of bits	Mnemonic
if (tw_mdct && !common_tw) {		
tw_data();		
}		
if (!common_ltpf) {		
if (**ltpf_data_present**) {	**1**	**uimsbf**
ltpf_pitch_lag_index;	**9**	**uimsbf**
ltpf_gain_index;	**2**	**uimsbf**
}		
}		
if ((!indepFlag) &&		
(window_sequence != EIGHT_SHORT_SEQUENCE)) {		
if (**fdp_data_present**) {	**1**	**uimsbf**
fdp_spacing_index;	**8**	**uimsbf**
}		
} else {		
fdp_data_present = 0;		
}		
if (indepFlag) {		
prev_aliasing_symmetry;	**1**	**uimsbf**
} else {		
prev_aliasing_symmetry = curr_aliasing_symmetry;		
}		
curr_aliasing_symmetry;	**1**	**uimsbf**
scale_factor_data();		
if (enhancedNoiseFilling) {		
igf_AllZero;	**1**	**uimsbf**
igf_level(igf_AllZero**,** indepFlag**);**	**0...**	NOTE
if (!igf_AllZero) {		
igf_data(indepFlag, 0);		
} else {		
igfCurrTileIdx = {3, 3, 3, 3};		
igfPrevTileIdx = {3, 3, 3, 3};		
igf_PrevWhiteningLevel = {0, 0, 0, 0};		
igf_WhiteningLevel = {0, 0, 0, 0};		
igfApplyTNF = 0;		
}		
if (tns_data_present) {		
tns_data();		
}		
ac_spectral_data(indepFlag);		
if (**fac_data_present**) {	**1**	**uimsbf**
if (fullbandLpd) {		
fac_length = (window_sequence==EIGHT_SHORT_SEQUENCE) ? ccfl/32 : ccfl/16;		
} else {		
fac_length = (window_sequence==EIGHT_SHORT_SEQUENCE) ? ccfl/16 : ccfl/8;		
}		
fac_data(1, fac_length);		
}		
}		
NOTE For details on **igf_level()** see subclause 5.5.5.		

Table 51 — Syntax of igf_stereo_pred_data()

Syntax	No. of bits	Mnemonic
igf_stereo_pred_data(igf_sfb_start, igf_sfb_stop, indepFlag)		
{		
igf_stereo_pred_all;	**1**	**uimsbf**
if (igf_stereo_pred_all == 0) {		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = igf_sfb_start; sfb < igf_sfb_stop; sfb += SFB_PER_PRED_BAND) {		
cplx_pred_used[g][sfb];	**1**	**uimsbf**
if ((sfb + 1) < igf_sfb_stop) {		
cplx_pred_used[g][sfb+1] = cplx_pred_used[g][sfb];		
}		
}		
}		
} else {		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = igf_sfb_start; sfb < igf_sfb_stop; sfb++) {		
cplx_pred_used[g][sfb] = 1;		
}		
}		
}		
igf_pred_dir;	**1**	**uimsbf**
if (indepFlag) {		
igf_delta_code_time = 0;		
} else {		
igf_delta_code_time;	**1**	**uimsbf**
}		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = igf_sfb_start; sfb < igf_sfb_stop; sfb += SFB_PER_PRED_BAND) {		
if (cplx_pred_used[g][sfb]) {		
hcod_sf[dpcm_alpha_q_re[g][sfb]];	**1..19**	**vlclbf**
} else {		
alpha_q_re[g][sfb] = 0;	**[1]**	[2]
}		
alpha_q_im[g][sfb] = 0;		
}		
}		
}		

Table 52 — Syntax of igf_data()

Syntax	No. of bits	Mnemonic
igf_data(indepFlag, core_mode)		
{		
if (!indepFlag) {		
igf_UsePrevTileIdx;	**1**	**uimsbf**
} else {		
igf_UsePrevTileIdx = 0;		
}		
if (igf_UsePrevTileIdx) {		
for (i = 0; i < igfNTiles; i++) {		
igfCurrTileIdx[i] = igfPrevTileIdx[i];		
}		
} else {		

Syntax	No. of bits	Mnemonic
for (i = 0; i < igfNTiles; i++) {		
igfCurrTileIdx[i];	**2**	**uimsbf**
}		
}		
for (i = 0; i < igfNTiles; i++) {		
igfPrevTileIdx[i] = igfCurrTileIdx[i];		
igf_WhiteningLevel[i] = 0;		
}		
for (i = igfNTiles; i < 4; i++) {		
igfCurrTileIdx[i] = 3;		
igfPrevTileIdx[i] = 3;		
igf_WhiteningLevel[i] = 0;		
}		
If (core_mode == 0) {		
igfEnableWht = (window_sequence != EIGHT_SHORT_SEQUENCE);		
} else {		
igfEnableWht = 1;		
}		
if (igfUseWhitening && igfEnableWht) {		
if (indepFlag) {		
igf_UsePrevWhiteningLevel = 0;		
} else {		
igf_UsePrevWhiteningLevel;	**1**	**uimsbf**
}		
if (igf_UsePrevWhiteningLevel) {		
for (i = 0; i < igfNTiles; i++) {		
igf_WhiteningLevel[i] = igf_PrevWhiteningLevel[i];		
}		
} else {		
if (**igf_WhiteningLevel**[0]) {	**1**	**uimsbf**
igf_WhiteningLevel[0] = 1 + **tmp**;	**1**	**uimsbf**
}		
if (**remainingTilesDifferent**) {	**1**	**uimsbf**
for (i = 1; i < igfNTiles; i++) {		
if (**igf_WhiteningLevel**[i]) {	**1**	**uimsbf**
igf_WhiteningLevel[i] = 1 + **tmp**;	**1**	**uimsbf**
}		
}		
} else {		
for (i = 1; i < igfNTiles; i++) {		
igf_WhiteningLevel[i] = igf_WhiteningLevel[0];		
}		
}		
}		
}		
for (i =0; i < 4; i++) {		
igf_PrevWhiteningLevel[i] = igf_WhiteningLevel[i];		
}		
if (igfUseEnf) {		
if (window_sequence != EIGHT_SHORT_SEQUENCE) {		
igfApplyTNF;	**1**	**uimsbf**
}		
}		
}		

Table 53 — Syntax of lpd_channel_stream()

Syntax	No. of bits	Mnemonic
lpd_channel_stream(noiseFilling, fullbandLpd, indepFlag)		
{		
if (fullbandLpd) {		
tns_data_present;	**1**	**uimsbf**
if (noiseFilling || tns_data_present) {		
window_shape;	**1**	**uimsbf**
max_sfb;	**4**	**uimsbf**
} else {		
window_shape = max_sfb = 0;		
}		
} else {		
tns_data_present = 0;		
}		
acelp_core_mode;	**3**	**uimsbf**
if (fullbandLpd) {		a
lpd_mode;	**3**	**uimsbf**
bpf_control_info = 1;		
} else {		
lpd_mode;	**5**	**uimsbf**
bpf_control_info;	**1**	**uimsbf**
}		
core_mode_last;	**1**	**uimsbf**
fac_data_present;	**1**	**uimsbf**
first_lpd_flag = !core_mode_last;		
first_tcx_flag = TRUE;		
k = 0;		
if (first_lpd_flag) {		
last_lpd_mode = -1;		b
}		
nbDiv = (fullbandLpd == 1) ? 2 : 4;		
while (k < nbDiv) {		
if (k == 0) {		
if ((core_mode_last == 1) && (fac_data_present == 1)) {		
fac_data(0, ccfl/8);		
}		
} else {		
if ((last_lpd_mode == 0 && mod[k] > 0) ||		
(last_lpd_mode > 0 && mod[k] == 0)) {		
fac_data(0, ccfl/8);		
}		
}		
if (mod[k] == 0) {		
acelp_coding(acelp_core_mode, fullbandLpd);		
last_lpd_mode = 0;		
k += 1;		
} else {		
window_sequence = EIGHT_SHORT_SEQUENCE;		c

Syntax	No. of bits	Mnemonic

tcx_coding(lg(mod[k]), first_tcx_flag, tns_data_present, noiseFilling,
enhancedNoiseFilling, indepFlag);
last_lpd_mode = mod[k];
k += (1 << (mod[k]-1));
first_tcx_flag = FALSE;
}
}
 	
lpc_data(first_lpd_flag);
 	
if ((core_mode_last == 0) && (fac_data_present == 1)) {
short_fac_flag;
	1	**uimsbf**

if (fullbandLpd) {
fac_length = short_fac_flag ? ccfl/32 : ccfl/16;
} else {
fac_length = short_fac_flag ? ccfl/16 : ccfl/8;
}
fac_data(1, fac_length);
}
}		

[a] **lpd_mode** defines the contents of the array mod[] as described in ISO/IEC 23003-3:2012, Table 89 or, if fullbandLpd is equal to 1, in Table 86.

[b] first_lpd_flag is defined in ISO/IEC 23003-3, subclause 6.2.10.2.

[c] The number of spectral coefficients, lg, depends on mod[k] according to ISO/IEC 23003-3:2012, Table 148 or, if fullbandLpd is equal to 1, in Table 88.

Table 54 — Syntax of acelp_coding()

Syntax	No. of bits	Mnemonic
acelp_coding(acelp_core_mode, fullbandLpd) { **mean_energy;**	**2**	**uimsbf**
nb_subfr = coreCoderFrameLength / 256;		[a]

for (sfr = 0; sfr < nb_subfr; sfr++) {
if ((sfr == 0) || ((nb_subfr == 4) && (sfr == 2))) {
acb_index[sfr];
	9	**uimsbf**

} else { **acb_index**[sfr];	**6**	**uimsbf**
} **ltp_filtering_flag**[sfr];	**1**	**bmsbf**
switch (acelp_core_mode) { case 0 **icb_index**[sfr]; break;	**20**	**uimsbf**
case 1 **icb_index**[sfr]; break;	**28**	**uimsbf**
case 2 **icb_index**[sfr]; break;	**36**	**uimsbf**

Syntax	No. of bits	Mnemonic
case 3		
icb_index[sfr];	**44**	**uimsbf**
break;		
case 4		
icb_index[sfr];	**52**	**uimsbf**
break;		
case 5		
icb_index[sfr];	**64**	**uimsbf**
break;		
case 6		
icb_index[sfr];	**12**	**uimsbf**
break;		
case 7		
icb_index[sfr];	**16**	**uimsbf**
break;		
}		
gains[sfr]**;**	**7**	**uimsbf**
}		
if (fullbandLpd) {		
tbe_data();		
}		
}		

[a] coreCoderFrameLength designates the core frame length in samples and is equal to either 1024 or 768.

Table 55 — Syntax of tcx_coding()

Syntax	No. of bits	Mnemonic
tcx_coding(lg, first_tcx_flag, tns_data_present, noiseFilling, enhancedNoiseFilling, indepFlag)		
{		
if (noiseFilling) {		
noise_factor;	**3**	**uimsbf**
} else {		
noise_factor = 8;		
}		
global_gain;	**7**	**uimsbf**
if (lg == ccfl) {		
if (**ltpf_data_present**) {	**1**	**uimsbf**
ltpf_pitch_lag_index;	**9**	**uimsbf**
ltpf_gain_index;	**2**	**uimsbf**
}		
} else {		
ltpf_data_present = 0;		
}		
if ((indepFlag == 0) && (lg == ccfl)) {		
if (**fdp_data_present**) {	**1**	**uimsbf**
fdp_spacing_index;	**8**	**uimsbf**
}		
} else {		
fdp_data_present = 0;		
}		

Syntax	No. of bits	Mnemonic
if (enhancedNoiseFilling) {		
num_windows = 1;		
igf_AllZero;	**1**	**uimsbf**
igf_level(igf_AllZero, indepFlag);	**0...**	[a]
if (!igf_AllZero) {		
igf_data(indepFlag, 1);		
} else {		
igfCurrTileIdx = {3, 3, 3, 3};		
igfPrevTileIdx = {3, 3, 3, 3};		
igf_PrevWhiteningLevel = {0, 0, 0, 0};		
igf_WhiteningLevel = {0, 0, 0, 0};		
igfApplyTNF = 0;		
}		
}		
if (tns_data_present) {		
num_windows = 1;		
tns_data();		
}		
if (first_tcx_flag) {		
if (indepFlag) {		
arith_reset_flag = 1;		
} else {		
arith_reset_flag;	**1**	**uimsbf**
}		
} else {		
arith_reset_flag = 0;		
}		
arith_data(lg, arith_reset_flag);		
}		

[a] For details on **igf_level()** see subclause 5.5.5.

Table 56 — Syntax of tbe_data()

Syntax	No. of bits	Mnemonic
tbe_data()		
{		
tbe_heMode;	**1**	**uimsbf**
idxFrameGain;	**5**	**uimsbf**
idxSubGains;	**5**	**uimsbf**
lsf_idx[0];	**7**	**uimsbf**
lsf_idx[0];	**7**	**uimsbf**
if (tbe_heMode == 0) {		
tbe_hrConfig;	**1**	**uimsbf**
tbe_nlConfig;	**1**	**uimsbf**
idxMixConfig;	**2**	**uimsbf**
if (tbe_hrConfig == 1) {		
idxShbFrGain;	**6**	**uimsbf**
idxResSubGains;	**5**	**uimsbf**
} else {		

Syntax	No. of bits	Mnemonic
idxShbExcResp[0];	**7**	**uimsbf**
idxShbExcResp[0];	**4**	**uimsbf**
}		
} else {		
tbe_nlConfig = 1;		
}		
}		

Table 57 — References to USAC syntactic elements

Syntax of	Defined in
ics_info()	ISO/IEC 23003-3:2012, 5.3, Table 22
tw_data()	ISO/IEC 23003-3:2012, 5.3, Table 27
scale_factor_data()	ISO/IEC 23003-3:2012, 5.3, Table 28
tns_data()	ISO/IEC 23003-3:2012, 5.3, Table 29
ac_spectral_data()	ISO/IEC 23003-3:2012, 5.3, Table 30
lpc_data()	ISO/IEC 23003-3:2012, 5.3, Table 32
code_book_indices()	ISO/IEC 23003-3:2012, 5.3, Table 35
arith_data()	ISO/IEC 23003-3:2012, 5.3, Table 38
fac_data()	ISO/IEC 23003-3:2012, 5.3, Table 39
UsacSbrData()	ISO/IEC 23003-3:2012, 5.3, Table 40
Mps212Data()	ISO/IEC 23003-3:2012, 5.3, Table 52

5.2.3.3 Extension element payloads

Table 58 — Syntax of AudioPreRoll()

Syntax	No. of bits	Mnemonic
AudioPreRoll()		
{		
configLen = escapedValue(4,4,8);	**4..16**	
Config()	**8*configLen**	
applyCrossfade;	**1**	**bool**
apr_reserved;	**1**	**bool**
numPreRollFrames = escapedValue(2,4,0);	**2..6**	
for (frameIdx=0; frameIdx < numPreRollFrames; ++frameIdx) {		
auLen = escapedValued(16,16,0)	**16..32**	**uimsbf**
AccessUnit()	**8*auLen**	
}		
}		

Table 59 — Syntax of TccGroupOfSegments()

Syntax	No. of bits	Mnemonic
TccGroupOfSegments()		
{		
if (**tccDataPresent**) {	**1**	**uimsbf**
numSegments;	**3**	**uimsbf**
for (k=0;k< numSegments;k++) {		
isContinued[k];	**1**	**uimsbf**
segLength[k];	**2**	**uimsbf**

Syntax	No. of bits	Mnemonic
amplQuant[k];	**1**	**uimsbf**
amplTransformCoeffDC[k];	**8**	**uimsbf**
j = 0;	[a]	
while (amplTransformIndex[k][j] = huff_dec(**huffWord**))	**1..12**	
{		
if (amplTransformIndex[k][j] == 0) {		
numAmplCoeffs = j;		
break;		
}		
j++;		
}		
for (j=0; j < numAmplCoeffs; j++) {	[b]	
amplTransformCoeffAC[k][j]= huff_dec(**huffWord**);	**1..15**	
amplSgn[k][j];	**1**	**uimsbf**
}		
freqQuant[k];	**1**	**uimsbf**
freqTransformCoeffDC[k];	**11**	**uimsbf**
j = 0;	[a]	
while (freqTransformIndex[k][j] = huff_dec(**huffWord**))	**1..12**	
{		
if (freqTransformIndex[k][j] == 0) {		
numFreqCoeffs = j;		
break;		
}		
j++;		
}		
for (j=0; j < numFreqCoeffs; j++) {	[b]	
freqTransformCoeffAC[k][j] = huff_dec(**huffWord**);	**1..15**	
freqSgn[k][j];	**1**	**uimsbf**
}		
}		
}		
}		

[a] Huffman codes table: Table 119.
[b] Huffman codes table: Table 120.

Table 60 — Syntax of MultichannelCodingBoxRotation()

Syntax	No. of bits	Mnemonic
MultichannelCodingBoxRotation()		
{		
if (keepTree == 0) {	[a]	
channelPairIndex;	**nBits**	
}		
else {		
channelPairIndex=lastChannelPairIndex;		
}		
hasMctMask;	**1**	uimsbf

Syntax	No. of bits	Mnemonic
hasBandwiseAngles;	**1**	uimsbf
windowsPerFrame =1;		
if (hasMctMask || hasBandwiseAngles) {		
isMCTShort;	**1**	uimsbf
numMaskBands;	**5**	uimsbf
if (isMCTShort) {		
numMaskBands = numMaskBands*8;		
windowsPerFrame =8;		
}		
} else {		
numMaskBands = MAX_NUM_MC_BANDS;		
}		
if (hasMctMask) {		
for(j=0;j<numMaskBands;j++) {		
mctMask[j]**;**	**1**	uimsbf
}		
} else {		
for(j=0;j<numMaskBands;j++) {		
mctMask[j] = 1;		
}		
}		
if (indepFlag > 0) {		
mct_delta_time = 0;		
} else {		
mct_delta_time;	**1**	uimsbf
}		
if (hasBandwiseAngles == 0) {		
hcod_angle[**dpcm_beta**[0]];	**1..10**	vlclbf
}		
else {		
for(j=0;j<numMaskBands;j++) {		
if (mctMask[j] ==1) {		
hcod_angle[**dpcm_beta**[j]];	**1..10**	vlclbf
}		
}		
}		
}		

[a] nBits = max(1, floor(log2(nMCTChannels · (nMCTChannels-1)/2 – 1))+1).

Table 61 — Syntax of MultichannelCodingBoxPrediction()

Syntax	No. of bits	Mnemonic
MultichannelCodingBoxPrediction()		
{		
if (keepTree == 0) {	[a]	
channelPairIndex;	**nBits**	uimsbf
}		
else {		

Syntax	No. of bits	Mnemonic
channelPairIndex= lastChannelPairIndex; }		
hasMctMask;	**1**	uimsbf
hasBandwiseCoeff;	**1**	uimsbf
windowsPerFrame =1; if (hasMctMask || hasBandwiseCoeff) {		
isMCTShort;	**1**	uimsbf
numMaskBands;	**5**	uimsbf

if (isMCTShort) {
numMaskBands = numMaskBands*8;
windowsPerFrame =8;
}
} else {
numMaskBands = MAX_NUM_MC_BANDS;
}
if (hasMctMask) {
for(j=0;j<numMaskBands;j++) {		

mctMask[j]**;**	**1**	uimsbf

}
} else {
for(j=0;j<numMaskBands;j++) {
mctMask[j] = 1;
}
}
pred_dir;
if (indepFlag > 0) {
mct_delta_time = 0;
} else {
	1	

mct_delta_time;	**1**	uimsbf
} if (hasBandwiseCoeff == 0) {		
hcod_sf[**dpcm_alpha_q_re**[0]];	**1..19**	vlclbf

}
else {
for(j=0;j<numMaskBands;j++) {
if (mctMask[j] ==1) {		

hcod_sf[**dpcm_alpha_q_re**[j]];	**1..19**	vlclbf
} } } }		

[a] nBits = max(1, floor(log2(nMCTChannels · (nMCTChannels-1)/2 – 1))+1).

Table 62 — Syntax of MultichannelCodingFrame()

Syntax	No. of bits	Mnemonic
MultichannelCodingFrame()		
{		
MCTSignalingType;	**2**	uimsbf
if (indepFlag == 1) {		
keepTree = 0;		
} else {		
keepTree;	**1**	uimsbf
}		
if (keepTree==0) {		
numPairs = escapedValue(5,8,16);		
}		
else {		
numPairs = lastNumPairs;		
}		
MCTStereoFilling = 0;		
if (MCTSignalingType > 1) {		
MCTSignalingType = MCTSignalingType – 2;		
MCTStereoFilling = 1;		
}		
for(pair=0; pair<numPairs;pair++) {		
hasStereoFilling[pair] = 0;		
if (MCTStereoFilling == 1) {		
hasStereoFilling[pair]**;**	**1**	uimsbf
}		
if (MCTSignalingType == 0) { /* tree of stereo prediction boxes */		
MultichannelCodingBoxPrediction();		
}		
if (MCTSignalingType == 1) { /* tree of rotation boxes */		
MultichannelCodingBoxRotation();		
}		
}		
}		

Table 63 — Syntax of HREPFrame()

Syntax	No. of bits	Mnemonic
HREPFrame(outputFrameLength, current_signal_group)		
{		
gain_count = outputFrameLength / 64;		
signal_count = bsNumberOfSignals[current_signal_group] + 1;		
useRawCoding;	**1**	**uimsbf**
if (useRawCoding) {		
for (pos = 0; pos < gain_count; pos++) {		
for (sig = 0; sig < signal_count; sig++) {		
if (isHREPActive[sig] == 0) continue;		
gainIdx[pos][sig];	**nBitsGain** [a]	**uimsbf**
}		
}		
} else {		
HREP_decode_ac_data(gain_count, signal_count);		
}		
for (sig = 0; sig < signal_count; sig++) {		
if (isHREPActive[sig] == 0) continue;		
all_zero = 1; /* all gains are zero for the current channel */		
for (pos = 0; pos < gain_count; pos++) {		
if (gainIdx[pos][sig] != GAIN_INDEX_0dB) {		
all_zero = 0;		
break;		
}		
}		
if (all_zero == 0) {		
useDefaultBetaFactorIdx;	**1**	**uimsbf**
if (useDefaultBetaFactorIdx) {		
betaFactorIdx[sig] = defaultBetaFactorIdx[sig];		
} else {		
betaFactorIdx[sig];	**nBitsBeta**	**uimsbf**
}		
}		
}		
}		
[a] nBitsGain = 3 + extendedGainRange.		

5.3 Data structure

5.3.1 General

The data structure is based on ISO/IEC 23003-3:2012, Clause 6.

Modifications and amendments are listed below.

Signal groups are represented by audio data elements which are configured in the mpegh3daDecoderConfig(). The bitstream element Signals3d() specifies the assignment of audio data elements to signal groups. In case a signal group requires extension payloads, the corresponding

mpegh3daExtElementConfig()s shall immediately precede the configuration elements for the audio data elements which belong to the associated signal group.

5.3.2 General configuration data elements

mpegh3daProfileLevelIndication Indication of MPEG-H 3D audio profile and level according to Table 64

Table 64 — Value of mpegh3daProfileLevelIndication

value	Indication of profile	Indication of level
0x00	reserved for ISO use	
0x01	main profile	L1
0x02	main profile	L2
0x03	main profile	L3
0x04	main profile	L4
0x05	main profile	L5
0x06	high profile	L1
0x07	high profile	L2
0x08	high profile	L3
0x09	high profile	L4
0x0A	high profile	L5
0x0B	low complexity profile	L1
0x0C	low complexity profile	L2
0x0D	low complexity profile	L3
0x0E	low complexity profile	L4
0x0F	low complexity profile	L5
0x10-0xFF	reserved for future profile definition	

cfg_reserved reserved, value shall be ignored.

receiverDelayCompensation This flag forces the decoder to operate in a constant delay. The decoder delay shall be kept constant by employing appropriate delay lines to obtain the delays noted in Table 1.

referenceLayout This structure describes the loudspeaker layout which the content of the present audio stream was originally designed or produced for. In that sense it represents the optimal loudspeaker layout from the content creator's perspective. This layout also acts as default rendering layout in the case where the targetLayout is not known to the decoder. The referenceLayout shall contain a specific and real layout. A ChannelConfiguration value of 0 (“any setup”) of ISO/IEC 23001-8 is not allowed.

audioChannelLayout — This structure describes the loudspeaker layout for a group with SignalGroupType == SignalGroupTypeChannels. If the audioChannelLayout is different from the referenceLayout (differsFromReferenceLayout==1), then the audioChannelLayout shall be smaller than the referenceLayout.

targetLayout — This structure describes the target loudspeaker layout, i.e. the actual loudspeaker constellation which the audio content shall be rendered to. This information will typically come from outside of the decoder. If the targetLayout is unknown, it shall be assumed that targetLayout = referenceLayout.

bsNumSignalGroups — This field defines the number of signal groups that are present in the bitstream.

signal_groupID — This variable implicitly assigns an ID to the signal groups listed in the Signals3d() syntax element. The ID is used to assign signal groups to transmitted downmix matrices in the DownmixMatrixSet() syntax element. Only signal groups of type SignalGroupTypeChannels may be assigned to downmix matrices.

signalGroupType — This field defines the type of a signal group.

Table 65 — Value of signalGroupType

signalGroupType	Value	Meaning
SignalGroupTypeChannels	0	Signal group contains channel signals, i.e. signals that should be played back from one static position, e.g. by a specific loudspeaker
SignalGroupTypeObject	1	Signal group contains object signals, i.e. signals that should be rendered to the reproduction layout
SignalGroupTypeSAOC	2	Signal group contains SAOC signals
SignalGroupTypeHOA	3	Signal group contains HOA signals
reserved	4 - 7	n.a.

differsFromReferenceLayout — This flag indicates whether the layout for which the signal group (of type SignalGroupTypeChannels) is intended differs from the reference layout.

numAudioChannels — Explicit number of audio channels that are conveyed in the present stream by means of SCEs, CPEs, QCEs, and LFEs. The value of this bitstream field shall correspond to the number of channels as signalled with the help of audioChannelLayout. In the element loop of mpegh3daDecoderConfig() these elements shall be located before all object related elements.

numAudioObjects	Explicit number of audio objects channels that are conveyed in the present stream by means of SCEs, CPEs, and QCEs and that are further processed in the object renderer. In the element loop of mpegh3daDecoderConfig() these elements shall be located after all audio channel related elements.
numSAOCTransportChannels	Explicit number of SAOC audio transport channels that are conveyed in the present stream by means of SCEs, CPEs, QCEs, and LFEs and that are further processed in the SAOC 3D Decoder. In the element loop of mpegh3daDecoderConfig() these elements shall be located after all audio channel and audio object related elements.
saocDmxLayoutPresent	This flag indicates if the SAOC audio transport channels contain a meaningful downmix of the input channels and objects. If saocDmxLayoutPresent == 1 the saocDmxChannelLayout is associated with the SAOC audio transport channels. Additionally, if the number of output channels is smaller than the number of SAOC audio transport channels, the SAOC audio transport channels should be further processed as signals of type SignalGroupTypeChannels with the audio channel layout saocDmxChannelLayout and the SAOC payload should be discarded.
saocDmxChannelLayout	This structure describes the loudspeaker layout of the SAOC audio transport channels, if saocDmxLayoutPresent == 1. The number of loudspeakers signalled in saocDmxChannelLayout shall be equal to (bsNumberOfSignals[grp] + 1), if SignalGroupType[grp] == SignalGroupTypeSAOC.
numHOATransportChannels	Explicit number of HOA transport channels that are conveyed in the present stream by means of SCEs, CPEs, QCEs and that are further processed in the HOA Decoder. In the element loop of mpegh3daDecoderConfig() these elements shall be located after all audio channel, audio object and SAOC related elements.

5.3.3 Loudspeaker configuration data elements

speakerLayoutType	This field indicates by which means the loudspeaker layout is conveyed in the bitstream element according to Table 66.

Table 66 — Meaning of speakerLayoutType

Value	Meaning
0	Loudspeaker layout shall be signalled by means of ChannelConfiguration index as defined in ISO/IEC 23001-8.
1	Loudspeaker layout shall be signalled by means of a list of LoudspeakerGeometry indices as defined in ISO/IEC 23001-8.
2	Loudspeaker layout is signalled by means of a list of explicit geometric position information.
3	Contribution Mode: No associated loudspeaker layout; Renderers shall operate as defined in subclause 4.7. This value is only allowed when signaling the referenceLayout.

CICPspeakerLayoutIdx ChannelConfiguration value. Shall be as defined in ISO/IEC 23001-8.

CICPspeakerIdx LoudspeakerGeometry value. Shall be as defined in ISO/IEC 23001-8.

angularPrecision This flag signals the angular precision of loudspeaker geometry information according to Table 67.

Table 67 — Meaning of angularPrecision

Value of angularPrecision	Angular precision degrees (°)	Number of bits used for coding of bitstream field:	
		ElevationAngleIdx	AzimuthAngleIdx
0	5	5	6
1	1	7	8

alsoAddSymmetricPair This flag signals if a symmetric pair on the horizontal plane is directly following.

isCICPspeakerIdx This flag signals whether the loudspeaker position shall be signalled by means of a LoudspeakerGeometry according to ISO/IEC 23001-8.

ElevationClass Indicates loudspeaker elevation by means of a simple middle/ upper/lower layer indication according to Table 68.

Table 68 — Meaning of ElevationClass field

Value	Meaning	ElevationAngle
0	Loudspeaker is located in the middle layer.	0°
1	Loudspeaker is located in the upper layer.	35°
2	Loudspeaker is located in the lower layer.	-15°
3	Loudspeaker position signalled explicitly.	N.A.

ElevationAngleIdx Index for calculating the elevation (i.e. vertical) angle for a given loudspeaker according to Table 69.

Table 69 — Calculation of ElevationAngle from ElevationAngleIdx

ElevationAngleIdx	angularPrecision == 0	angularPrecision == 1
0-18	ElevationAngle = ElevationAngleIdx * 5°	ElevationAngle = ElevationAngleIdx * 1°
19-31	reserved	
32-90	N.A.	
91-127	N.A.	reserved

ElevationDirection This flag signals the direction of increasing elevation according to Table 70.

Table 70 — Meaning of ElevationDirection

Value of ElevationDirection	Direction
0	Upwards
1	Downwards

AzimuthAngleIdx Index for calculating the azimuth (i.e. horizontal) angle for a given loudspeaker according to Table 71.

Table 71 — Calculation AzimuthAngle from AzimuthAngleIdx

AzimuthAngleIdx	angularPrecision == 0	angularPrecision == 1
0-36	AzimuthAngle = AzimuthAngleIdx * 5°	AzimuthAngle = AzimuthAngleIdx * 1°
37-180	N.A.	
181-255	N.A.	reserved

AzimuthDirection This flag signals the direction of increasing azimuth angles according to Table 72.

Table 72 — Meaning of AzimuthDirection

Value of AzimuthDirection	Direction
0	Counter-clockwise
1	clockwise

isLFE This flag signals if the given loudspeaker is an LFE loudspeaker.

5.3.4 Core decoder configuration data elements

qceIndex This index describes whether two subsequent elements of type mpegh3daChannelPairElement() are treated as a quadruple channel element (QCE). The different QCE modes are given in Table 73. The qceIndex shall be the same for the two subsequent elements forming one QCE.

Table 73 — Value of qceIndex

qceIndex	Meaning
0	Stereo CPE
1	QCE without residual
2	QCE with residual
3	reserved

shiftIndex0 This flag signals if the first channel in the element is the next non-assigned channel or if the element is shifted in the channel map relative to the next non-assigned channel.

shiftIndex1 This flag signals if the second channel in element is the next non-assigned channel or if the second channel in the element is shifted in the channel map relative to the next non-assigned channel.

shiftChannel0 Offset by which the first channel in the element is shifted relative to the next non-assigned channel.

shiftChannel1 Offset by which the second channel in the element is shifted relative to the next non-assigned channel.

enhancedNoiseFilling This flag signals the usage of the enhanced noise filling tool.

igfUseEnf This flag signals the usage of IGF envelope noise flattening.

igfUseWhitening This flag signals the usage of IGF spectral whitening.

igfAfterTnsSynth This flag signals that IGF should be applied after TNS synthesis filtering.

igfIndependentTiling This flag signals that IGF is applied in discrete channel mode.

igfStartIndex This flag signals the IGF start index, which is mapped to a scalefactor band index.

igfStopIndex This flag signals the IGF stop index, which is mapped to a scalefactor band index.

igfUseHighRes This flag signals that for every scalefactor band in IGF range a IGF level value is transmitted. If the flag is zero, low resolution is used which implies, that for two scalefactor bands only one IGF level value is transmitted.

igfCurrTileIdx[ ] Vector of length 4 containing the tile index.

igfApplyTNF Flag indicating the application of IGF-TNF filtering.

igf_data() Syntax element which reads IGF tile and whitening side information for each channel ch.

igf_level() Syntax element which reads the IGF level information for each channel ch.

igf_AllZero This flag signals that all levels in IGF range are zero.

igf_UsePrevTileIdx This flag signals that previous values of tile indices should be used.

igf_UsePrevWhiteningLevel This flag signals that previous values of whitening levels should be used.

igf_WhiteningLevel[ ] This values describe which whitening should be used, see Table 74.

igf_ms_mask_present this two bit field indicates that the MS mask is:
00 All zeros.
01 A mask of max_sfb bands of **ms_used** follows this field.
10 All ones.
11 M/S coding is disabled, real stereo prediction is enabled.

igf_pred_dir Indicates the direction of prediction (same as cplx_pred_dir).

Table 74 — Value of igf_WhiteningLevel

igf_WhiteningLevel	Meaning
0	use medium whitening
1	do not use whitening
2	use pseudo-random noise

usacExtElementType This element allows to signal bitstream extensions types. The meaning of usacExtElementType is defined in Table 75.

Table 75 — Value of usacExtElementType

usacExtElementType	Value
ID_EXT_ELE_FILL	0
ID_EXT_ELE_MPEGS	1
ID_EXT_ELE_SAOC	2
ID_EXT_ELE_AUDIOPREROLL	3
ID_EXT_ELE_UNI_DRC	4
ID_EXT_ELE_OBJ_METADATA	5
ID_EXT_ELE_SAOC_3D	6
ID_EXT_ELE_HOA	7
ID_EXT_ELE_FMT_CNVRTR	8
ID_EXT_ELE_MCT	9
ID_EXT_ELE_TCC	10
ID_EXT_ELE_HOA_ENH_LAYER	11
ID_EXT_ELE_HREP	12
ID_EXT_ELE_ENHANCED_OBJ_METADATA	13
/* reserved for ISO use */	14-127
/* reserved for use outside of ISO scope */	128 and higher
NOTE Application-specific usacExtElementType values are mandated to be in the space reserved for use outside of ISO scope. These are skipped by a decoder as a minimum of structure is required by the decoder to skip these extensions.	

usacExtElementSegmentData The concatenation of all usacExtElementSegmentData from mpegh3daExtElement() of consecutive frames, starting from the mpegh3daExtElement() with usacExtElementStart==1 up to and including the mpegh3daExtElement() with usacExtElementStop==1 forms one data block. In case a complete data block is contained in one mpegh3daExtElement(), usacExtElementStart and usacExtElementStop shall both be set to 1. The data blocks are interpreted as a byte aligned extension payload depending on usacExtElementType according to Table 76.

Table 76 — Interpretation of data blocks for extension payload decoding

usacExtElementType	The concatenated usacExtElementSegmentData shall be:
ID_EXT_ELE_FILL	Series of **fill_byte**
ID_EXT_ELE_MPEGS	SpatialFrame() as defined in ISO/IEC 23003-1
ID_EXT_ELE_SAOC	SAOCFrame() as defined in ISO/IEC 23003-2
ID_EXT_ELE_AUDIOPREROLL	AudioPreRoll()
ID_EXT_ELE_UNI_DRC	uniDrcGain() as defined in ISO/IEC 23003-4
ID_EXT_ELE_OBJ_METADATA	objectMetadataFrame()
ID_EXT_ELE_SAOC_3D	Saoc3DFrame()
ID_EXT_ELE_HOA	HOAFrame()
ID_EXT_ELE_FMT_CNVRTR	FormatConverterFrame()
ID_EXT_ELE_MCT	MultichannelCodingFrame()
ID_EXT_ELE_TCC	TccGroupOfSegments()
ID_EXT_ELE_HOA_ENH_LAYER	HOAEnhFrame()
ID_EXT_ELE_HREP	HREPFrame(outputFrameLength, current_signal_group)
ID_EXT_ELE_ENHANCED_OBJ_METADATA	EnhancedObjectMetadataFrame()
unknown	unknown data. The data block shall be discarded.

usacConfigExtType This element allows to signal configuration extension types. The meaning of usacConfigExtType is defined in Table 77.

Table 77 — Value of usacConfigExtType

usacConfigExtType	Value
ID_CONFIG_EXT_FILL	0
ID_CONFIG_EXT_DOWNMIX	1
ID_CONFIG_EXT_LOUDNESS_INFO	2
ID_CONFIG_EXT_AUDIOSCENE_INFO	3
ID_CONFIG_EXT_HOA_MATRIX	4
ID_CONFIG_EXT_ICG	5
ID_CONFIG_EXT_SIG_GROUP_INFO	6
/* reserved for ISO use */	7-127
/* reserved for use outside of ISO scope */	128 and higher

5.3.5 Downmix matrix data elements

downmixIdCount number of downmixId definitions present in the bitstream element.

downmixId This field uniquely defines an ID for a default downmix matrix available on the decoder side or a transmitted downmix matrix. downmixId has two reserved values, which are forbidden, namely 0x0 and 0x7F. All other values can be freely chosen. Further details on the usage of downmixId can be found in subclause 6.4.3.

downmixType This index defines whether a downmixId is connected with a default downmix matrix available on the decoder side or a transmitted downmix matrix.

Table 78 — Value of downmixType

downmixType	Meaning
0	Format conversion with default downmix matrix available on the decoder side
1	Format conversion with transmitted downmix matrix
2	reserved
3	reserved

CICPspeakerLayoutIdx This value describes the target loudspeaker layout for the given downmix matrix. The value shall correspond to **ChannelConfiguration** as defined in ISO/IEC 23001-8.

DmxMatrixLenBits length of the following bitstream element in bits.

paramConfig, inputConfig, outputConfig — Channel configuration vectors specifying the information about each loudspeaker. The information is assumed to be known from the channel configurations of the input and output layouts. Each entry, paramConfig[i], is a structure with the members:

- AzimuthAngle, the absolute value of the loudspeaker azimuth angle
- AzimuthDirection, the direction of increasing azimuth, 0 (left) or 1 (right)
- ElevationAngle, the absolute value of the loudspeaker elevation angle
- ElevationDirection, the direction of increasing elevation, 0 (up) or 1 (down)
- isLFE, indicates whether the loudspeaker is a LFE loudspeaker

paramCount, inputCount, outputCount — Number of loudspeakers in the corresponding channel configuration vectors.

compactParamConfig, compactInputConfig, compactOutputConfig	Compact channel configuration vectors specifying the information about each loudspeaker group. Each entry, compactParamConfig[i], is a structure with the members:
	pairType, type of the loudspeaker group, which can be SYMMETRIC (a symmetric pair of two loudspeakers), CENTRE, or ASYMMETRIC
	isLFE, indicates whether the loudspeaker group consists of LFE loudspeakers
	originalPosition, position in the original channel configuration of the first (i.e. left) loudspeaker, or the only loudspeaker, in the group
	symmetricPair.originalPosition, position in the original channel configuration of the second (i.e. right) loudspeaker in the group, for SYMMETRIC groups only
compactParamCount, compactInputCount, compactOutputCount	Number of loudspeaker groups in the corresponding compact channel configuration vectors.
equalizerPresent	Boolean indicating whether equalizer information that is to be applied to the input channels is present.
precisionLevel	Precision used for uniform quantization of the gains according to Table 79.

Table 79 — Uniform quantization step size of gain depending on precisionLevel

precisionLevel	**smallest quantization step size** [dB]
0	1
1	0,5
2	0,25
3	reserved

maxGain	Maximum actual gain in the matrix, expressed in dB: possible values from 0 to 22.
minGain	Minimum actual gain in the matrix, expressed in dB: possible values from -1 to -47.
isAllSeparable	Boolean indicating whether all the output loudspeaker groups satisfy the separability property.
isSeparable[i]	Boolean indicating whether the output loudspeaker group with index i satisfies the separability property.

isAllSymmetric	Boolean indicating whether all the output loudspeaker groups satisfy the symmetry property.
isSymmetric[i]	Boolean indicating whether the output loudspeaker group with index i satisfies the symmetry property.
mixLFEOnlyToLFE	Boolean indicating whether the LFE loudspeakers are mixed only to LFE loudspeakers and, at the same time, the non-LFE loudspeakers are mixed only to non-LFE loudspeakers.
rawCodingCompactMatrix	Boolean indicating whether compactDownmixMatrix is coded raw (using one bit per entry) or it is coded using run-length coding followed by limited Golomb-Rice.
compactDownmixMatrix[i][j]	An entry in compactDownmixMatrix corresponding to input loudspeaker group i and output loudspeaker group j, indicating whether any of the associated gains is nonzero: 0 = all gains are zero, 1 = at least one gain is nonzero.
useCompactTemplate	Boolean indicating whether to apply an element-wise XOR to compactDownmixMatrix with a predefined compact template matrix, to improve the efficiency of the run-length coding.
runLGRParam	Limited Golomb-Rice parameter p used to code the zero run-lengths in the linearized flatCompactMatrix.
flatCompactMatrix	Linearized version of compactDownmixMatrix with the predefined compact template matrix already applied; When mixLFEOnlyToLFE is enabled, it does not include the entries known to be zero (due to mixing between non-LFE and LFE) or those used for LFE to LFE mixing.
compactTemplate	Predefined compact template matrix, having “typical” entries, which is XORed element-wise to compactDownmixMatrix, in order to improve coding efficiency by creating mostly zero value entries.
zeroRunLength	The length of a zero run always followed by a one, in the flatCompactMatrix, which is coded with limited Golomb-Rice coding, using the parameter runLGRParam.
fullForAsymmetricInputs	Boolean indicating whether to ignore the symmetry property for every asymmetric input loudspeaker group; When enabled, every asymmetric input loudspeaker group will have two gain values decoded for each symmetric output loudspeaker group with index i, regardless of isSymmetric[i].
gainTable	Dynamically generated gain table which contains the list of all possible gains between minGain and maxGain with precision precisionLevel.
rawCodingNonzeros	Boolean indicating whether the nonzero gain values are coded raw (uniform coding, using the ReadRange function) or their indexes in the gainTable list are coded using limited Golomb-Rice coding.

gainLGRParam — Limited Golomb-Rice parameter p used to code the nonzero gain indexes, computed by searching each gain in the gainTable list.

5.3.6 HOA rendering matrix data elements

numOfHoaRenderingMatrices — Number of *HoaRenderingMatrixId* definitions present in the bitstream element.

HoaRenderingMatrixId — This field references a downmixId for which the HOA rendering matrix is available.

CICPspeakerLayoutIdx — This value describes the output loudspeaker layout for the given HOA rendering matrix. The value shall correspond to **ChannelConfiguration** as defined in ISO/IEC 23001-8 and shall be consistent with the CICPspeakerLayoutIdx value as defined in the DownmixMatrixSet() for the same dowmnixId.

HoaRenderingMatrixLenBits — Length of the following bitstream element in bits.

outputConfig — Channel configuration vectors specifying the information about each loudspeaker. The information is assumed to be known from the channel configurations of the output layout. Each entry, outputConfig[i], is a structure with the members:

- AzimuthAngle, the absolute value of the loudspeaker azimuth angle;
- AzimuthDirection, direction of increasing azimuth, 0 (left) or 1 (right);
- ElevationAngle, the absolute value of the loudspeaker elevation angle;
- ElevationDirection, direction of increasing elevation, 0 (up) or 1 (down);
- isLFE, indicates whether the loudspeaker is a LFE loudspeaker.

The helper function *findSymmetricSpeakers* further specifies the following specifications.

- pairType, can be SYMMETRIC (a symmetric pair of two loudspeakers), CENTRE, or ASYMMETRIC;
- symmetricPair->originalPosition, position in the original channel configuration of the second (i.e. right) loudspeaker in the group, for SYMMETRIC groups only.

outputCount — Number of loudspeakers the HOA rendering matrix is defined for.

numPairs — Number of symmetric loudspeaker pairs identified in the output loudspeaker setup which can be considered for efficient symmetry coding.

maxHoaOrder — HOA Order of the transmitted matrix.

isNormalized Indicates if the HOA rendering matrix $\boldsymbol{D}$ is energy normalized, so that $||\boldsymbol{D}||_f = \sum_{l=1}^{L}\sum_{n=1}^{(N+1)^2} D_{l,n}^2 = 1$ with l being the non-LFE loudspeakers in the outputConfig.

precisionLevel Precision used for uniform quantization of the gains according to Table 80.

Table 80 — Uniform quantization step size for HOA rendering matrices as a function of the precisionLevel

precisionLevel	**Smallest quantization step size** [dB]
0	1.0
1	0.5
2	0.25
3	0.125

gainLimitPerHoaOrder This flag indicates if the maxGain and minGain are individually specified for each order or for the entire HOA matrix.

maxGain[i] Maximum actual gain in the matrix for coefficients for the HOA order *i*, expressed in dB.

minGain[i] Minimum actual gain in the matrix for coefficients of the HOA order *i*, expressed in dB.

isFullMatrix This flag indicates if the HOA matrix is sparse or full.

nbitsHoaOrder Number of bits reading firstSparseOrder.

firstSparseOrder In case the HOA matrix was specified as sparse, this field defines the first HOA order which is sparsely coded.

isHoaCoefSparse Bitmask vector derived from firstSparseOrder.

currentHoaOrder Variable which indicates the current HOA order during the decoding process.

hasValue A flag used in case the matrix element is sparsely coded.

lfeExist This flag indicates if one or more LFE channels exist in outputConfig.

hasLfeRendering This flag indicates if the rendering matrix contains non-zero elements for the LFE channels. The signals for the LFE channels are rendered without additional spectral filtering.

zerothOrderAlwaysPositive This flag indicates if the 0th HOA order has only positive values.

isAllValueSymmetric This flag indicates if all symmetric loudspeaker pairs have equal absolute values in the HOA rendering matrix.

isAnyValueSymmetric if *isAllValueSymmetric* is false, this flag indicates if some of the symmetric loudspeaker pairs have equal absolute values in the HOA rendering matrix.

valueSymmetricPairs Bitmask of length *numPairs* indicating the loudspeaker pairs with value-symmetry.

isValueSymmetric Bitmask derived from *valueSymmetricPairs*.

isAllSignSymmetric If there are no value symmetries in the matrix, this flag indicates if all symmetric loudspeaker pairs have at least number sign symmetries.

isAnySignSymmetric This flag indicates if there are at least some symmetric loudspeaker pairs with number sign symmetries.

signSymmetricPairs Bitmask of length *numPairs* indicating the loudspeaker pairs with sign-symmetry.

signVal Number sign value.

isSignSymmetric Bitmask derived from *signSymmetricPairs*.

hasVerticalCoef This flag indicates if this is a horizontal-only HOA rendering matrix.

boolVal A variable used in the decoding loop.

signMatrix Matrix with the sign values of the HOA rendering matrix in linearized vector-form.

hoaMatrix Final HOA rendering matrix values in linearized vector-form.

5.3.7 Signal group information elements

groupPriority This field defines the priority of the group. It can take integer values between 0 and 7. The group may be discarded from rendering and decoding if the priority is lower than 7. If groups are discarded, the groups with lowest priority should be discarded first.

fixedPosition This field defines if the positions of the members of a group shall be updated in the context of processing of tracking data (scene displacement data). In case the flag is equal to zero, the position of the corresponding group's members are updated during the processing of scene displacement angles. In case the flag is equal to one, the positions of the corresponding group's members are not updated during the processing of scene displacement angles. The default value for fixedPosition is zero for all groups.

5.3.8 Low frequency enhancement (LFE) channel element, mpegh3daLfeElement()

In order to maintain a regular structure in the decoder, the mpegh3daLfeElement() is defined as a standard fd_channel_stream(0,0,0,0,0,x) element, i.e. equal to a mpegh3daCoreCoderData() using the frequency domain coder (core_mode[ch] equals 0). Thus, decoding can be done using the standard procedure for decoding a mpegh3daCoreCoderData()-element.

In order to accommodate a more bitrate and hardware efficient implementation of the LFE decoder, however, several restrictions apply to the options used for the encoding of this element:

— The window_sequence field shall always be set to 0 (ONLY_LONG_SEQUENCE);

— The prev_aliasing_symmetry field and curr_aliasing_symmetry field shall always be set to 0;

— Only the lowest 24 spectral coefficients of any LFE may be non-zero;

— No temporal noise shaping is to be used, i.e. tns_data_present shall be set to 0;

— Time warping shall not be active, i.e. tw_mdct shall be set to 0;

— No noise filling shall be applied, i.e. noiseFilling shall be set to 0;

— No enhanced noise filling shall be applied, i.e. enhancedNoiseFilling shall be set to 0;

— No long term prediction filter shall be used, i.e. ltpf_data_present and common_ltpf shall be set to 0;

— No frequency domain predictor shall be used, i.e. fdp_data_present shall be set to 0.

5.4 Configuration element descriptions

5.4.1 General

The configuration elements are based on ISO/IEC 23003-3:2012, subclause 6.1. Additional information is given in the following subclause.

The mapping of core audio elements to audio channels is applied according to the following rule.

1) Obtain the number of audio channels (numAudioChannels) from Signals3d().

2) Determine the already decoded successive elements until the currently processed element from mpegh3daDecoderConfig().

3) Decode shiftChannel0 and shiftChannel1 from corresponding CPE elements. The default value for shiftChannel0 and shiftChannel1 shall be zero.

The decoding procedure is specified as follows.

a) Create an array with the size of the number of all loudspeaker signals and initialize it with -1.

b) Determine the smallest non-assigned channel ch that is missing in the array starting with ch=0.

c) Proceed to the next position pos in the array that equals -1. i.e. find the smallest pos such that array[pos] = -1.

d) if shiftIndexX is zero: write the channel index ch into the array at the position pos. i.e. array[pos] = ch.

e) if shiftIndexX is greater than zero: increment ch by (shiftIndexX+1) and write it into the array at the position pos. i.e. array[pos] = ch + shiftIndexX + 1.

f) Repeat b) - e) for all channels in an element.

g) Repeat b) - f) for all successive elements.

Apply the same algorithm to audio objects for mapping the core audio elements to audio object channels.

5.4.2 Downmix configuration

5.4.2.1 General

Downmix matrix coefficients and/or active downmix setting parameters may be transmitted by the encoder to enable control over the format conversion process at the decoder. Transmission is facilitated by means of a mpegh3daConfigExtension of Type ID_CONFIG_EXT_DOWNMIX for downmixType == 1. The mpegh3daConfigExtension may contain downmix matrices as well as an active downmix setting parameter. If downmix matrices are transmitted, each downmix matrix signals its associated target loudspeaker layout that determines the matrix dimensions and identifies which kind of downmix matrix operation the transmitted coefficients are suitable for.

The transmission of a unique *downmixId* allows referencing to a default downmix matrix available on the decoder side, or to a transmitted downmix matrix from outside of the audio stream, e.g. from dynamic range compression related information in higher systems layers.

5.4.2.2 Data elements and variables

downmixConfigType This parameter defines whether an active downmix control parameter (value 0) or downmix matrices (value 1) or both (value 2) are transmitted. Value 3 is reserved.

passiveDownmixFlag Signals that a passive downmix shall be applied in the format converter downmix if value=1. If not transmitted, **passiveDownmixFlag** shall be set to 0, resulting in the application of the active downmix processing of the format converter downmix.

immersiveDownmixFlag Signals whether the immersive rendering format converter or the generic format converter shall be applied in the decoder to convert channel signals to the target loudspeaker configuration as defined in Table 81. If not transmitted, **immersiveDownmixFlag** shall be set to 0.

Table 81 — Meaning of immersiveDownmixFlag

immersiveDownmixFlag	Meaning
0	Generic format converter shall be applied as defined in Clauses 10 and 24 according to subclause 4.4.5.
1	If the target layout, signalled by LoudspeakerRendering(), is signalled as (speakerLayoutType==0,CICPspeakerLayoutIdx==5) or as (speakerLayoutType==0,CICPspeakerLayoutIdx==6), independently of potentially signalled loudspeaker displacement angles, then immersive rendering format converter shall be applied as defined in Clauses 11 and 25 according to subclause 4.4.5. In all other case the generic format converter shall be applied as defined in Clauses 10 and 24 according to subclause 4.4.5.

5.4.2.3 Golomb-Rice coding

Golomb-Rice coding is used to code any non-negative integer $n \geq 0$ using a given non-negative integer parameter $p \geq 0$ as follows: first code the number $h = \lfloor n/2^p \rfloor$ using unary coding, as h one bits followed by a terminating zero bit; then code the number $l = n - h \cdot 2^p$ uniformly using p bits.

Limited Golomb-Rice coding is a trivial variant used when it is known in advance that $n < N$, for a given integer $N \geq 1$. It does not include the terminating zero bit when coding the maximum possible value of h, which is $h_{max} = \lfloor (N-1)/2^p \rfloor$. More exactly, to encode $h = h_{max}$ we write only h one bits, but not the terminating zero bit, which is not needed because the decoder can implicitly detect this condition.

5.4.2.4 Helper functions

The function `ConvertToCompactConfig()` specified below is used to convert the given *paramConfig* configuration consisting of *paramCount* loudspeakers into the compact *compactParamConfig* configuration consisting of *compactParamCount* loudspeaker groups. The *compactParamConfig[i].pairType* field can be SYMMETRIC (S), when the group represents a pair of symmetric loudspeakers, CENTRE (C), when the group represents a centre loudspeaker, or ASYMMETRIC (A), when the group represents a loudspeaker without a symmetric pair.

```
ConvertToCompactConfig(paramConfig, paramCount)
{
    for (i = 0; i < paramCount; ++i) {
        alreadyUsed[i] = 0;
    }

    idx = 0;
    for (i = 0; i < paramCount; ++i) {
        if (alreadyUsed[i]) continue;
        compactParamConfig[idx].isLFE = paramConfig[i].isLFE;

        if ((paramConfig[i].AzimuthAngle == 0) ||
                (paramConfig[i].AzimuthAngle == 180°) {
            compactParamConfig[idx].pairType = CENTER;
            compactParamConfig[idx].originalPosition = i;
        } else {
            j = SearchForSymmetricSpeaker(paramConfig, paramCount, i);
            if (j != -1) {
                compactParamConfig[idx].pairType = SYMMETRIC;
```

```
                if (paramConfig.AzimuthDirection == 0) {
                    compactParamConfig[idx].originalPosition = i;
                    compactParamConfig[idx].symmetricPair.originalPosition = j;
                } else {
                    compactParamConfig[idx].originalPosition = j;
                    compactParamConfig[idx].symmetricPair.originalPosition = i;
                }
                alreadyUsed[j] = 1;
            } else {
                compactParamConfig[idx].pairType = ASYMMETRIC;
                compactParamConfig[idx].originalPosition = i;
            }
        }
        idx++;
    }

    compactParamCount = idx;

    return (compactParamConfig, compactParamCount);
}


CalculateTotalCount()
{
    if (mixLFEOnlyToLFE) {
        compactInputLFECount = 0;
        compactOutputLFECount = 0;
        for (i = 0; i < compactInputCount; i++) {
            if (compactInputConfig[i].isLFE) compactInputLFECount++;
        }
        for (i = 0; i < compactOutputCount; i++) {
            if (compactOutputConfig[i].isLFE) compactOutputLFECount++;
        }
        totalCount = (compactInputCount - compactInputLFECount) *
            (compactOutputCount - compactOutputLFECount);
    } else {
        totalCount = compactInputCount * compactOutputCount;
    }
    return totalCount;
}
```

The function `FindCompactTemplate(inputConfig, inputCount, outputConfig, outputCount)` is used to find a compact template matrix matching the input channel configuration represented by *inputConfig* and *inputCount*, and the output channel configuration represented by *outputConfig* and *outputCount*.

The compact template matrix shall be found by searching in the predefined list of compact template matrices defined in Annex E, available at both the encoder and decoder, for the one with the same list of input loudspeakers as *inputConfig* and the same list of output loudspeakers as *outputConfig*.

Use of template matrices for efficient coding of downmix matrices is only allowed when both the input and output channel configurations exactly correspond to a codepoint ChannelConfiguration as defined in ISO/IEC 23001-8 and if an exactly matching template matrix is defined in this standard. This ensures that the correct order of channels is obeyed at the decoder as well as at the encoder. Template matrices are defined in this standard only for pairs of ChannelConfiguration codepoints. For matching of a template matrix, if an input or output configuration is given, instead of a ChannelConfiguration

codepoint, as a list of predefined loudspeakers indexes or geometry description according to subclause 5.2.2.2, an additional check shall be made to verify whether this provided description exactly matches a ChannelConfiguration codepoint. In case of an exact match, the corresponding ChannelConfiguration codepoint will therefore be used in the selection procedure of a matching template matrix.

The function `SearchForSymmetricSpeaker(paramConfig, paramCount, i)` is used to search the channel configuration represented by *paramConfig* and *paramCount* for the symmetric loudspeaker corresponding to the loudspeaker *paramConfig[i]*. This symmetric loudspeaker, *paramConfig[j],* shall be situated after the loudspeaker *paramConfig[i]*, therefore *j* can be in the range *i+1* to paramCount – *1*, inclusive.

5.4.2.5 Gain value quantization

The function `readRange()` is used to read a uniformly distributed integer in the range [0 *alphabetSize*-1] inclusive.

The function `generateGainTable(maxGain, minGain, precisionLevel)` is used to dynamically generate the gain table *gainTable* which contains the list of all possible gains between *minGain* and *maxGain* with precision *precisionLevel*. The table is terminated by adding an element signalling minus infinity dB gain having the placeholder value *minGain-1*. The order of the values is chosen so that the most frequently used values and also more "round" values would be typically closer to the beginning of the list. The gain table of length gainTableSize with the list of all possible gain values is generated as follows:

— add integer multiples of 3 dB, going down from 0 dB to *minGain*;
— add integer multiples of 3 dB, going up from 3 dB to *maxGain*;
— add remaining integer multiples of 1 dB, going down from 0 dB to *minGain*;
— add remaining integer multiples of 1 dB, going up from 1 dB to *maxGain*;
— stop here if *precisionLevel* is 0 (corresponding to 1 dB) and add *minGain-1*;
— add remaining integer multiples of 0,5 dB, going down from 0 dB to *minGain*;
— add remaining integer multiples of 0,5 dB, going up from 0,5 dB to *maxGain*;
— stop here if *precisionLevel* is 1 (corresponding to 0,5 dB) and add *minGain-1*;
— add remaining integer multiples of 0,25 dB, going down from 0 dB to *minGain*;
— add remaining integer multiples of 0,25 dB, going up from 0,25 dB to *maxGain*;
— add *minGain-1.*

EXAMPLE When *maxGain* is 2 dB and *minGain* is -6 dB, and precisionLevel is 0,5 dB, the following list is created:
0, -3, -6, -1, -2, -4, -5, 1, 2, -0.5, -1.5, -2.5, -3.5, -4.5, -5.5, 0.5, 1.5, -7.

5.4.2.6 Equalizer Config

numEqualizers	Number of different equalizer filters present
eqPrecisionLevel	Precision used for uniform quantization of the gains: 0 = 1 dB, 1 = 0,5 dB, 2 = 0,25 dB, 3 = 0,1 dB
eqExtendedRange	Boolean indicating whether to use an extended range for the gains; if enabled, the available range is doubled
numSections	Number of sections of an equalizer filter, each one being a peak filter
centerFreqLd2	The leading two decimal digits of the centre frequency for a peak filter; the maximum range is 10 .. 99
centerFreqP10	Number of zeros to be appended to centerFreqLd2; the maximum range is 0 .. 3
qFactorIndex	Quality factor index for a peak filter
qFactorExtra	Extra bits for decoding a quality factor larger than 1.0
centerGainIndex	Gain at the centre frequency for a peak filter
scalingGainIndex	Scaling gain for an equalizer filter
hasEqualizer[i]	Boolean indicating whether the input channel with index *i* has an equalizer associated to it
equalizerIndex[i]	The index of the equalizer associated with the input channel with index *i*

5.4.2.7 Decoding of downmix coefficients

The syntax element *DownmixMatrixSet()* contains one or more downmix matrices that may be applied to achieve a format conversion to a desired loudspeaker layout. A given bitstream shall not contain more than one instance of *DownmixMatrixSet()*.

The syntax element *DownmixMatrix()* contains the downmix matrix information. The decoder first reads the equalizer information represented by the syntax element *EqualizerConfig(),* if enabled. The fields *precisionLevel, maxGain,* and *minGain* are then read. The input and output configurations are converted to compact configurations using the function *ConvertToCompactConfig()*. Then, the flags indicating if the separability and symmetry properties are satisfied for each output loudspeaker group are read.

The significance matrix *compactDownmixMatrix* is then read, either a) raw using one bit per entry, or b) using the limited Golomb-Rice coding of the run lengths, and then copying the decoded bits from *flatCompactMatrix* to *compactDownmixMatrix* and applying the *compactTemplate* matrix.

Finally, the nonzero gains are read. For each nonzero entry of *compactDownmixMatrix*, depending on the field *pairType* of the corresponding input group and the field *pairType* of the corresponding output group, a sub-matrix of size up to 2 by 2 has to be reconstructed. Using the associated separability and symmetry properties, a number of gain values are read using the function *DecodeGainValue()*. A gain value can be decoded uniformly, by using the function *ReadRange()*, or using the limited Golomb-Rice decoding of the index of the gain in the *gainTable* table, which contains all the possible gain values.

5.4.2.8 Decoding of equalizer config

The syntax element *EqualizerConfig()* contains the equalizer information that is to be applied to the input channels. A number of *numEqualizers* equalizer filters is first decoded and thereafter selected for specific input channels using *equalizerIndex[i]*. The fields *eqPrecisionLevel* and *eqExtendedRange* indicate the quantization precision and the available range of the scaling gains and of the peak filter gains.

Each equalizer filter is a serial cascade consisting in a number of *numSections* of peak filters and one *scalingGain*. Each peak filter is fully defined by its *centerFreq*, *qualityFactor*, and *centerGain*.

The *centerFreq* parameters of the peak filters which belong to a given equalizer filter shall be given in ascending order. The parameter is limited to 10 .. 24000 Hz inclusive, and it is calculated as

$$centerFreq = centerFreqLd2 \cdot 10^{centerFreqP10}$$

The *qualityFactor* parameter of the peak filter can represent values between 0.05 and 1.0 inclusive with a precision of 0.05 and from 1.1 to 10.6 inclusive with a precision of 0.1 and it is calculated as

$$\text{qualityFactor} = \begin{cases} 0.05 \cdot (\text{qFactorIndex} + 1), & \text{if qFactorIndex} \leq 19 \\ 1.0 + 0.1 \cdot [(\text{qFactorIndex} - 20) \cdot 8 + \text{qFactorExtra} + 1], & \text{otherwise} \end{cases}$$

The vector *eqPrecisions* gives the precision in dB corresponding to a given *eqPrecisionLevel*, and the *eqMinRanges* and *eqMaxRanges* matrices give the minimum and maximum values in dB for the gains corresponding to a given *eqExtendedRange* and *eqPrecisionLevel*.

eqPrecisions[4] = {1.0, 0.5, 0.25, 0.1};

eqMinRanges[2][4] = {{-8.0, -8.0, -8.0, -6.4}, {-16.0, -16.0, -16.0, -12.8}};

eqMaxRanges[2][4] = {{7.0, 7.5, 7.75, 6.3}, {15.0, 15.5, 15.75, 12.7}};

The parameter *scalingGain* uses the precision level min(eqPrecisionLevel + 1,3), which is the next better precision level if not already the last one. The mappings from the fields *centerGainIndex* and *scalingGainIndex* to the gain parameters *centerGain* and *scalingGain* are calculated as:

centerGain
= *eqMinRanges*[*eqExtendedRange*][*eqPrecisionLevel*]
+ *eqPrecisions*[*eqPrecisionLevel*] · *centerGainIndex*

scalingGain
= *eqMinRanges*[*eqExtendedRange*][min(*eqPrecisionLevel* + 1,3)]
+ *eqPrecisions*[min(*eqPrecisionLevel* + 1,3)] · *scalingGainIndex*

5.4.3 HOA rendering matrix configuration

5.4.3.1 General

Higher Order Ambisonics (HOA) rendering matrices may be transmitted by the encoder to enable control over the HOA rendering process at the decoder. Transmission is facilitated by means of a mpegh3daConfigExtension of Type ID_CONFIG_EXT_HOA_MATRIX. The mpegh3daConfigExtension may contain several HOA rendering matrices for different loudspeaker reproduction configurations. If HOA rendering matrices are transmitted, each HOA rendering matrix signals its associated target loudspeaker layout that, together with the *HoaOrder* (see Table 187), determines the dimensions of the rendering matrix.

The transmission of a unique *HoaRenderingMatrixId* allows referencing to a default HOA rendering matrix available on the decoder side, or to a transmitted HOA rendering matrix from outside of the audio stream.

A signalled HOA rendering matrix shall expect full 3D normalised (N3D) HOA coefficients as its input. Further, a rendering matrix shall adhere to the ordering of the HOA coefficients c_n^m that is used by the HOA decoder, where n is the order, m is the degree, and N is the maximum HOA order. (see also Annex F):

$$\boldsymbol{c} = \begin{bmatrix} c_0^0 & c_1^{-1} & c_1^0 & c_1^1 & c_2^{-2} & c_2^{-1} & c_2^0 & c_2^1 & c_2^2 & \dots\ c_N^{N-1} & c_N^N \end{bmatrix}^T$$

5.4.3.2 Helper functions

The function `findSymmetricSpeakers` (not displayed here) indicates the total number and position of all loudspeaker pairs within the provided loudspeaker setup which are left-right symmetric for a listener at the sweet spot.

```
int findSymmetricSpeakers(int outputCount, SpeakerInformation*
outputConfig, int hasLfeRendering);
```

The function *createSymSigns* is used to compute a vector of 1 and -1 values which is used to generate the matrix elements associated with symmetric loudspeakers.

```
void createSymSigns(int* symSigns, int hoaOrder)
{
   int n, m, k = 0;
   for (n = 0; n<=hoaOrder; ++n) {
      for (m = -n; m<=n; ++m)
         symSigns[k++] = ((m>=0)*2)-1;
   }
}
```

The function *create2dBitmask* generates a bitmask to identify the HOA coefficients that are only used in the horizontal plane.

```
void create2dBitmask(int* bitmask, int hoaOrder)
{
   int n, m, k = 0;
   bitmask[k++] = 0;
   for (n = 1; n<=hoaOrder; ++n) {
      for (m = -n; m<=n; ++m)
         bitmask[k++] = abs(m)!=n;
   }
}
```

5.4.3.3 Decoding of HOA rendering matrix coefficients

The syntax element *HoaRenderingMatrixSet()* contains one or more HOA rendering matrices that may be applied to achieve HOA rendering to a desired loudspeaker layout. A given bitstream shall not contain more than one instance of *HoaRenderingMatrixSet()*.

The syntax element *HoaRenderingMatrix()* contains the HOA rendering matrix information. The decoder first reads the config information that guides the reading and matrix decoding process. After that the matrix elements are read from the bitstream accordingly.

At the beginning the fields `precisionLevel` and `gainLimitPerOrder` are read. If the flag `gainLimitPerOrder` is set, then the following fields `maxGain`, and `minGain` are read and decoded for each HOA order separately. If the flag `gainLimitPerOrder` is not set, then the fields `maxGain` and `minGain` are only read and decoded once and applied to all HOA orders during the decoding process. The `minGain` value shall be between 0 db and -101 dB. The `maxGain` value shall be between 1 dB and 111 dB lower than the minGain value.

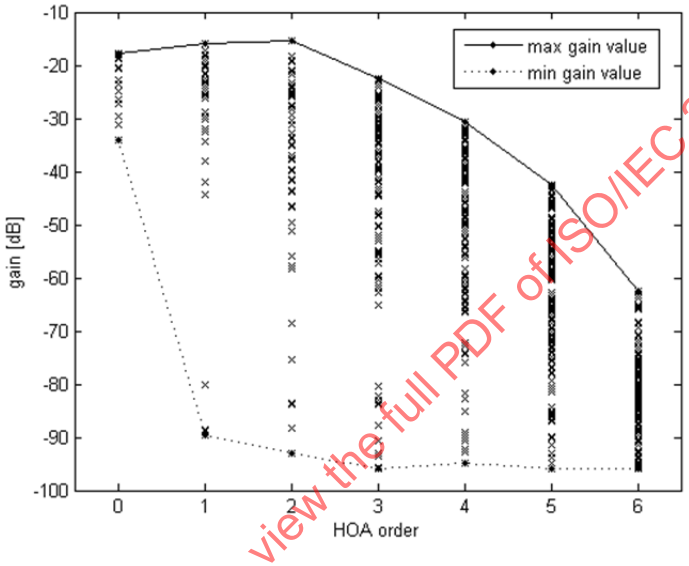


Figure 4 — Example for HOA order dependent min and max gains within an HOA rendering matrix

The next field to read is the flag `isFullMatrix` which signals if a matrix is defined as full or as partially sparse. In case the matrix is defined as partially sparse, the next field `firstSparseOrder` specifies the starting HOA order, after which the HOA rendering matrix is sparsely coded. For example, if `firstSparseOrder` signals 4, then all matrix elements associated with the HOA order 4 and above are sparsely coded. HOA rendering matrices are dense for low order and often become sparse in the higher orders, depending on the loudspeaker reproduction setup. As an example Figure 5 depicts a partially sparse 6th order HOA rendering matrix for 22 loudspeakers.

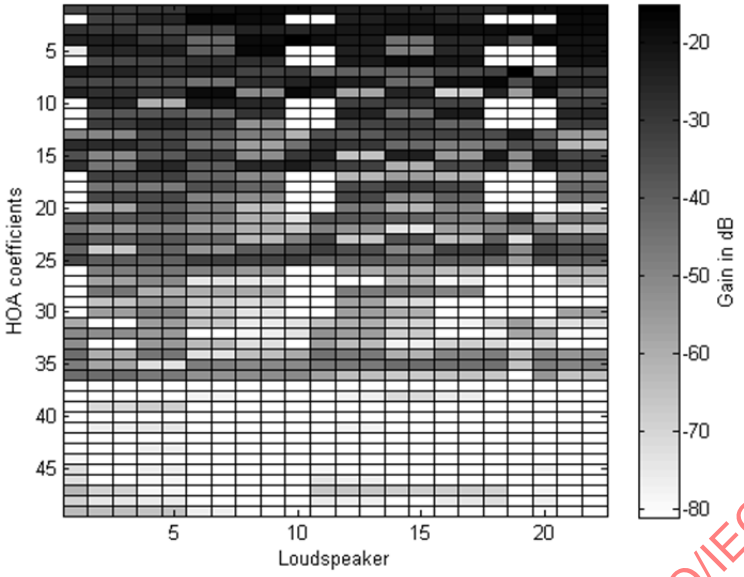


Figure 5 — Example for an HOA rendering matrix with sparsity starting at the 26th HOA coefficient (HOA order 5)

Depending whether LFE channels exist within the loudspeaker reproduction setup, the field `hasLfeRendering` is read. If `hasLfeRendering` is not set, then the matrix elements related to the LFE channels are assumed to be digital zeroes. The next field read by the decoder is the flag `zerothOrderAlwaysPositive` and signals if the matrix elements associated with the coefficient of the 0th order are positive. In this case no number signs are coded.

In the following, properties of the HOA rendering matrix are signalled for loudspeaker pairs symmetric with regards to the median plane. There are two symmetry properties: a) value symmetry and b) sign symmetry. In the case of value symmetry, the matrix elements of the left loudspeaker of the symmetric loudspeaker pair are not coded, but derived from the decoded matrix elements of the right loudspeaker by employing the helper function `createSymSigns`:

```
pairIdx = outputConfig[j].symmetricPair->originalPosition;
hoaMatrix[i * outputCount + j] = hoaMatrix[i * outputCount + pairIdx];
signMatrix[i * outputCount + j] = symSigns[i] * signMatrix[i * outputCount
+ pairIdx];
```

If a loudspeaker pair is not value symmetric, then the matrix elements may be symmetric with regards to their number signs. If a loudspeaker pair is sign symmetric, the number signs of the matrix elements of the left loudspeaker of the symmetric loudspeaker pair are not coded, but derived from the number signs of the matrix elements associated to the right loudspeaker by employing the helper function `createSymSigns`:

```
pairIdx = outputConfig[j].symmetricPair->originalPosition;
signMatrix[i * outputCount + j] = symSigns[i] * signMatrix[i * outputCount
+ pairIdx];
```

The signalling of the symmetry properties is visualised in Figure 6. A loudspeaker pair cannot be defined as value symmetric and sign symmetric at the same time. The final decoding flag `hasVerticalCoef` specifies if only the matrix elements associated with circular (i.e., 2D) HOA

coefficients are coded. If `hasVerticalCoef` is not set the matrix elements associated with the HOA coefficients defined with the helper function `create2dBitmask` are set to digital zero.

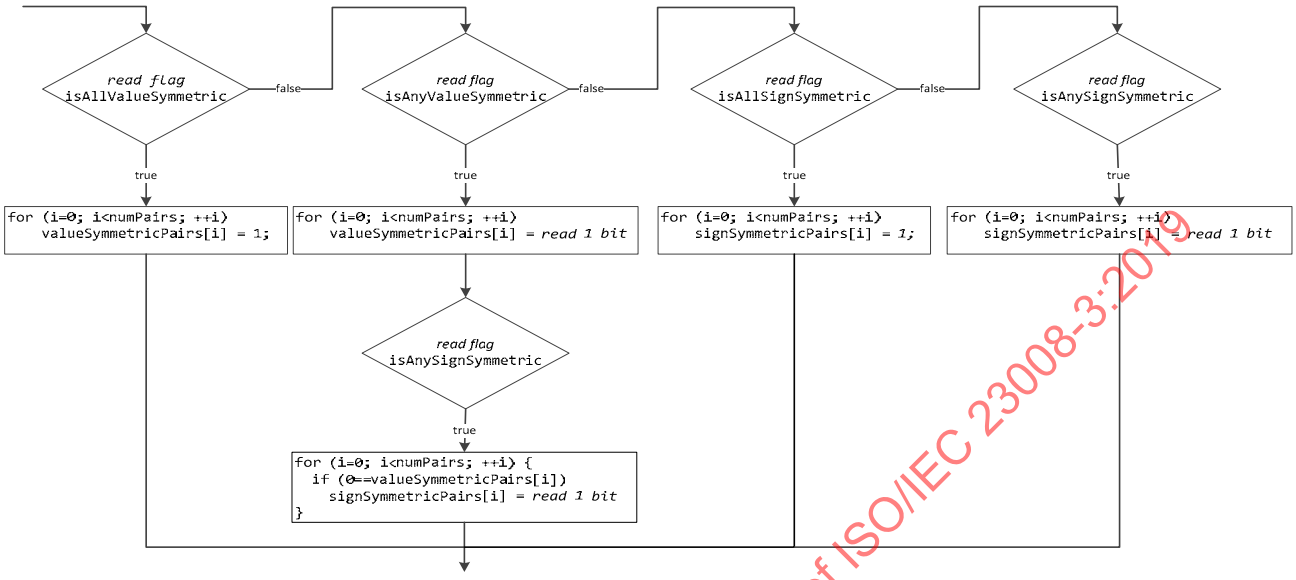


Figure 6 — Signalling of symmetry properties in HOA rendering matrices

Using the above mentioned properties, a series of matrix element gain values are read. To read the absolute gain value, the function *DecodeGainValue()* is used. The gain values are uniformly decoded by using the function *ReadRange()* of the alphabet index. If the decoded gain value is not digital zero, the number sign value (`signVal`) is additionally read [see also Figure 7 a)]. If the matrix element is associated with an HOA coefficient that was signalled to be sparse (via `isHoaCoefSparse`), the `hasValue` flag precedes the `gainValueIndex` [see Figure 7 b)]. If the `hasValue` flag is zero, this element is set to digital zero and no `gainValueIndex` and `signVal` are signalled.

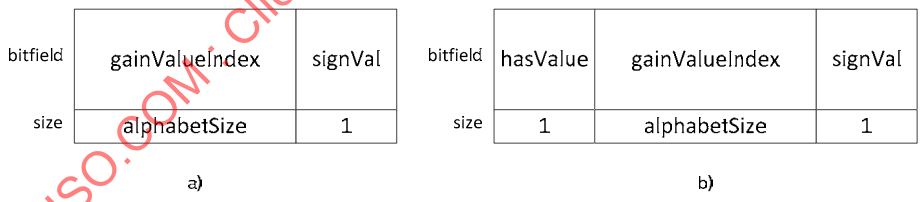


Figure 7 — Examples for bitstream syntax to decode a rendering matrix element

Depending on the specified symmetry properties for loudspeaker pairs, the matrix elements associated with the left loudspeaker are derived from the right loudspeaker. In this case the code words to decode the individual matrix elements for the left loudspeaker are reduced or completely omitted.

If the bitfield isNormalized was set to 1 the final HOA rendering matrix $\boldsymbol{D}$ shall be created by dividing each weighting value in the L rows of the HOA rendering matrix that are associated with non-LFE loudspeakers by the matrix's Frobenius Norm $\sum_{l=1}^{L}\sum_{n=1}^{(N+1)^2} D_{l,n}^2$ computed from its L rows associated with non-LFE loudspeakers.

5.5 Tool descriptions

5.5.1 General

The tool descriptions are based on ISO/IEC 23003-3:2012, Clause 7.

Modifications and amendments to the existing tool descriptions are listed below.

5.5.2 Quad channel element

5.5.2.1 Tool description

The quad channel element (QCE) is a method for joint coding of four channels for more efficient coding of horizontally and vertically distributed channels. A QCE consists of two consecutive CPEs and is formed by hierarchically combining the joint stereo tool with possibility of complex stereo prediction in horizontal direction and the MPEG surround based stereo tool in vertical direction. This is achieved by enabling both stereo tools and swapping output channels between applying the tools. Stereo SBR is performed in horizontal direction to preserve the left-right relations of high frequencies.

5.5.2.2 Definitions

Help elements:

cplx_out_dmx_L[ ]	First channel of first CPE after complex prediction stereo decoding.
cplx_out_dmx_R[ ]	Second channel of first CPE after complex prediction stereo decoding.
cplx_out_res_L[ ]	Second CPE after complex prediction stereo decoding. (zero if qceIndex = 1)
cplx_out_res_R[ ]	Second channel of second CPE after complex prediction stereo decoding. (zero if qceIndex = 1)
mps_out_L_1[ ]	First output channel of first MPS box.
mps_out_L_2 [ ]	Second output channel of first MPS box.
mps_out_R_1[ ]	First output channel of second MPS box.
mps_out_R_2[ ]	Second output channel of second MPS box.
sbr_out_L_1[ ]	First output channel of first Stereo SBR box.
sbr_out_R_1[ ]	Second output channel of first Stereo SBR box.
sbr_out_L_2[ ]	First output channel of second Stereo SBR box.
sbr_out_R_2[ ]	Second output channel of second Stereo SBR box.

5.5.2.3 Decoding process

The syntax element **qceIndex** in mpegh3daChannelPairElementConfig() indicates whether a CPE belongs to a QCE and if residual coding is used. In case that **qceIndex** is unequal 0, the current CPE forms a QCE together with its subsequent element which shall be a CPE having the same **qceIndex**.

Stereo SBR is always used for the QCE, thus the syntax item **stereoConfigIndex** shall be 3 and **bsStereoSbr** shall be 1.

In case of **qceIndex** == 1 only the payloads for MPEG Surround and SBR and no relevant audio signal data is contained in the second CPE and the syntax element **bsResidualCoding** is set to 0.

The presence of a residual signal in the second CPE is indicated by **qceIndex** == 2. In this case the syntax element **bsResidualCoding** is set to 1.

Decoding of Joint Stereo is performed as specified in ISO/IEC 23003-3:2012, 7.7. The resulting output of the first CPE are the MPS downmix signals cplx_out_dmx_L[] and cplx_out_dmx_R[]. If residual coding is used (i.e. **qceIndex** == 2), the output of the second CPE are the MPS residual signals cplx_out_res_L[], cplx_out_res_R[], if no residual signal has been transmitted (i.e. **qceIndex** == 1), zero signals are inserted.

The structure of the QCE decoding process is illustrated in Figure 8.

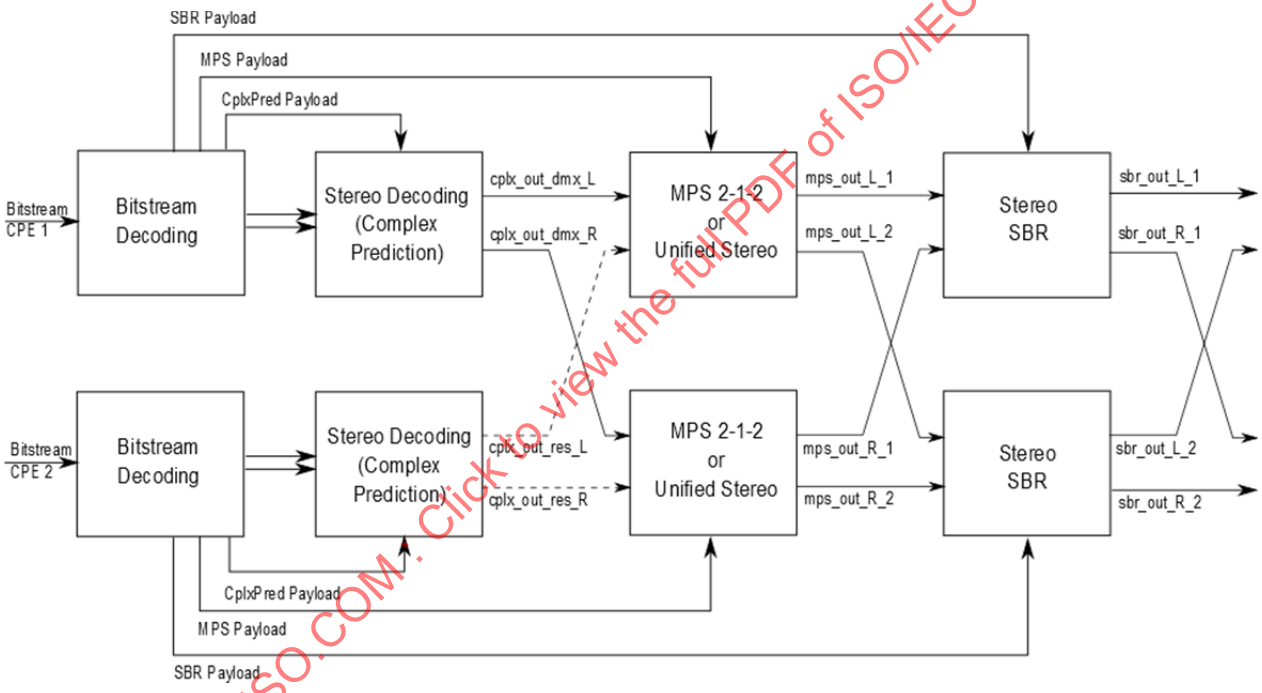


Figure 8 — QCE decoder schematics

Before applying MPEG surround decoding, the second channel of the first element (cplx_out_dmx_R[]) and the first channel of the second element (cplx_out_res_L[]) are swapped.

Decoding of MPEG surround with residual is performed as specified in ISO/IEC 23003-3:2012, 7.11. Decoding of MPEG surround without residual using SBR as defined in ISO/IEC 23003-3:2012, subclause 7.11.2.7, is modified so that stereo SBR is also used for **bsResidualCoding** == 1, resulting in the following decoder schematics (see Figure 9).

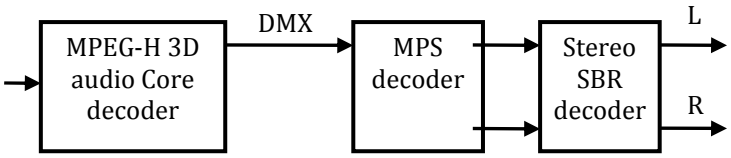


Figure 9 — bsResidualCoding == 0, bsStereoSbr == 1

Before applying stereo SBR, the second channel of the first element (mps_out_L_2[]) and the first channel of the second element (mps_out_R_1[]) are swapped to allow right-left stereo SBR. After application of stereo SBR, the second output channel of the first element (sbr_out_R_1[]) and the first channel of the second element (sbr_out_L_2[]) are swapped again to restore the input channel order.

5.5.3 Transform splitting

5.5.3.1 Tool description

When transform splitting (TS) is active in a long transform, two half-length lapped transforms are employed instead of one full-length transform. The coefficients of the two lapped transforms are transmitted in a line-by-line interleaved fashion as a traditional frequency domain (FD) transform, with the coefficients of the first-in-time transform placed at even and the coefficients of the second-in-time transform placed at odd indices.

5.5.3.2 Operational constraints

TS can only be used in a FD *long-start* or *stop-start* window (**window_sequence** == 1). Also, TS can only be applied when **noiseFilling** is 1 in mpegh3daCoreConfig( ). When TS is signalled, all FD tools except for TNS and inverse transform operate on the interleaved (long) set of TS coefficients. This allows the reuse of the scalefactor band offset and long-transform arithmetic coder tables as well as the window shapes and overlap lengths.

5.5.3.3 Definitions

Help elements:

common_window indicates if channel 0 and channel 1 of a CPE use identical window parameters (see ISO/IEC 23003-3:2012 subclause 6.2.5.1.1).

window_sequence FD window sequence type for the current frame and channel (see ISO/IEC 23003-3:2012 subclause 6.2.9).

tns_on_lr indicates the mode of operation for TNS filtering (see ISO/IEC 23003-3:2012 subclause 7.8.2).

noiseFilling this flag signals the usage of the noise filling of spectral holes in the FD core coder (see ISO/IEC 23003-3:2012 subclause 6.1.1.1).

noise_offset noise-fill offset to modify scale factors of zero-quantized bands (see ISO/IEC 23003-3:2012 subclause 7.2).

noise_level noise-fill level representing amplitude of added spectrum noise (see ISO/IEC 23003-3:2012 subclause 7.2).

split_transform	binary flag indicating whether TS is utilized in the current frame and channel.
half_transform_length	one half of coreCoderFrameLength (ccfl, the transform length, see ISO/IEC 23003-3:2012, subclause 6.1.1).
half_lowpass_line	one half of the number of transform lines transmitted for the current channel.

5.5.3.4 Decoding process

5.5.3.4.1 General

The decoding of an FD (stop-)start transform with TS is performed in three sequential steps as follows.

5.5.3.4.2 Decoding of split_transform and half_lowpass_line

The help element split_transform does not represent an independent bit-stream element but is derived from the noise filling elements, **noise_offset** and **noise_level**, and in case of a mpegh3daChannelPairElement(), the **common_window** flag in StereoCoreToolInfo(). If **noiseFilling** == 0, split_transform is 0. Otherwise,

```
if ((noiseFilling != 0) &&
    (noise_level == 0) &&
    (noise_offset != 0) &&
    (window_sequence == 1)) {
  split_transform = 1;
  noise_level     = (noise_offset & 28) / 4;
  noise_offset    = (noise_offset & 3) * 8;
}
else {
  split_transform = 0;
}
```

In other words, if split_transform == 1, **noise_offset** contains 5 bits of noise filling data, which are then rearranged. Since this operation changes the values of **noise_level** and **noise_offset**, it shall be executed before the noise filling process of ISO/IEC 23003-3:2012, subclause 7.2.

Furthermore, if **common_window** == 1 in a mpegh3daChannelPairElement(), split_transform is determined only in the left (first) channel; the right channel's split_transform is set equal to (i.e. copied from) the left channel's split_transform, and the following pseudo-code is not executed in the right channel.

The help element half_lowpass_line is determined from the "long" scalefactor band offset table, swb_offset_long_window, and the max_sfb of the current channel, or in case of stereo and **common_window** == 1, max_sfb_ste.

$$\text{lowpass\_sfb} = \begin{cases} \text{max\_sfb\_ste} & \text{in elements with StereoCoreToolInfo() and } \textbf{common\_window} == 1, \\ \text{max\_sfb} & \text{otherwise.} \end{cases}$$

Based on the **enhancedNoiseFilling** flag, half_lowpass_line is derived:

```
if (enhancedNoiseFilling != 0) {
  lowpass_sfb = max(lowpass_sfb, m_igFStopSfb);
}
half_lowpass_line = swb_offset_long_window[lowpass_sfb] / 2;
```

5.5.3.4.3 De-interleaving of half-length spectra for temporal noise shaping

After spectrum de-quantization, noise filling, and scalefactor application and prior to the application of temporal noise shaping (TNS), the TS coefficients in spec[] are de-interleaved using a helper buffer[]:

```
for (i = 0, i2 = 0; i < half_lowpass_line; i += 1, i2 += 2) {
  spec[i]   = spec[i2]; /* isolate 1st window */
  buffer[i] = spec[i2+1];  /* isolate 2nd window */
}
for (i = 0; i < half_lowpass_line; i += 1) {
  spec[i+half_lowpass_line] = buffer[i];    /* copy 2nd window */
}
```

The in-place de-interleaving effectively places the two half-length TS spectra on top of each other, and the TNS tool now operates as usual on the resulting full-length pseudo-spectrum.

5.5.3.4.4 Temporary re-interleaving, two sequential inverse lapped transforms

If **common_window** == 1 in the current frame or the stereo decoding is performed after TNS decoding (**tns_on_lr** == 0 in ISO/IEC 23003-3:2012, subclause 7.8), spec[] shall be re-interleaved temporarily into a full-length spectrum:

```
for (i = 0; i < half_lowpass_line; i += 1) {
  buffer[i] = spec[i];  /* copy 1st window */
}
for (i = 0, i2 = 0; i < half_lowpass_line; i += 1, i2 += 2) {
  spec[i2]   = buffer[i];  /* merge 1st window */
  spec[i2+1] = spec[i+half_lowpass_line];  /* merge 2nd window */
}
```

The resulting pseudo-spectrum is used for stereo decoding (ISO/IEC 23003-3:2012, subclause 7.7) and to update dmx_re_prev[] (ISO/IEC 23003-3:2012 subclauses 7.7.2 and 5.5.3.6). In case of **tns_on_lr** == 0, the stereo-decoded full-length spectra are again de-interleaved by repeating the process of subclause 5.5.3.4.3. Finally, the 2 lapped transforms are calculated with ccfl and the channel's **window_shape** of the current and last frame.

5.5.3.5 Filterbank and block switching

5.5.3.5.1 Inverse Lapped Transform

The processing for TS follows the description given in ISO/IEC 23003-3:2012, subclause 7.9. The following modifications or additions shall be taken into account. See also subclause 5.5.11.

The TS coefficients in spec[] are de-interleaved using a helper buffer[] with N, the window length based on the **window_sequence** value (N = ccfl·2 since, by definition, the sequence isn't an EIGHT_SHORT_SEQUENCE):

```
for (i = 0, i2 = 0; i < N/2; i += 1, i2 += 2) {
  spec[0][i] = spec[i2];   /* isolate 1st window */
  buffer[i]  = spec[i2+1]; /* isolate 2nd window */
}
for (i = 0; i < N/2; i += 1) {
  spec[1][i] = buffer[i];  /* copy 2nd window */
}
```

The inverse transform for each half-length TS spectrum spec[0, 1] is then defined as follows, with cs() and k_0 as specified, via the prev_aliasing_symmetry and curr_aliasing_symmetry values, by Table 113.

$$x_{j,n} = \frac{2}{N}\sum_{k=0}^{\frac{N}{4}-1} spec[j][k] \cdot \mathrm{cs}\left(\frac{4\pi}{N}(n+n_0)(k+k_0)\right) \quad \text{for } 0 \le n < N/2$$

where

j is the window index, shall be 0 or 1;

n is the time-domain sample index;

k is the spectral coefficient index;

n_0 (N/4 + 1)/2.

Note that for the second inverse transform $x_{1,n}$, prev_aliasing_symmetry is set to curr_aliasing_symmetry. Subsequent windowing and block switching steps for the transform outputs $x_{(0,1)}$ are defined in the next subclauses.

5.5.3.5.2 Transform splitting with STOP_START_SEQUENCE

The STOP_START_SEQUENCE in combination with transform splitting is depicted in Figure 10. It comprises two overlapped and added half-length windows with a length of *N_l/2* which is 1024 (768). *N_s* is set to 256 (192) respectively.

Figure 10 — Transform splitting for STOP_START_WINDOW

The windows (0,1) for the two half-length inverse lapped transforms are given as follows:

$$W_{(0,1)}(n) = \begin{cases} 0.0 & \text{, for } 0 \le n < \frac{N_l/2 - N_s}{4} \\ W_{(0,1),LEFT,N_s}\left(n - \frac{N_l/2 - N_s}{4}\right) & \text{, for } \frac{N_l/2 - N_s}{4} \le n < \frac{N_l/2 + N_s}{4} \\ 1.0 & \text{, for } \frac{N_l/2 + N_s}{4} \le n < \frac{3N_l/2 - N_s}{4} \\ W_{(0,1),RIGHT,N_s}\left(n - \frac{3N_l/2 - N_s}{4}\right) & \text{, for } \frac{3N_l/2 - N_s}{4} \le n < \frac{3N_l/2 + N_s}{4} \\ 0.0 & \text{, for } \frac{3N_l/2 + N_s}{4} \le n < N_l/2 \end{cases}$$

where for the first inverse lapped transform the windows

$$W_{0,LEFT,N_s}(n) = \begin{cases} W_{KBD\_LEFT,N_s}(n), \text{ if window\_shape\_previous\_block} == 1 \\ W_{SIN\_LEFT,N_s}(n), \text{ if window\_shape\_previous\_block} == 0 \end{cases},$$

$$W_{0,RIGHT,N_s}(n) = \begin{cases} W_{KBD\_RIGHT,N_s}(n), \text{ if window\_shape} == 1 \\ W_{SIN\_RIGHT,N_s}(n), \text{ if window\_shape} == 0 \end{cases}$$

are applied and for the second inverse lapped transform the windows:

$$W_{1,LEFT,N_s}(n) = \begin{cases} W_{KBD\_LEFT,N_s}(n), \text{ if window\_shape} == 1 \\ W_{SIN\_LEFT,N_s}(n), \text{ if window\_shape} == 0 \end{cases}$$

$$W_{1,RIGHT,N_s}(n) = \begin{cases} W_{KBD\_RIGHT,N_s}(n), \text{ if window\_shape} == 1 \\ W_{SIN\_RIGHT,N_s}(n), \text{ if window\_shape} == 0 \end{cases}$$

are applied.

The overlap and add between the two half-length windows resulting in the windowed time domain values $z_{i,n}$ is specified as follows. Here, N_l is set to 2048 (1536), N_s to 256 (192) respectively:

$$Z_{i,n}(n) = \begin{cases} 0.0 & \text{,for} \quad 0 \le n < N_s \\ x_{0,n-N_s} \cdot W_0(n-N_s) & \text{,for} \quad N_s \le n < \frac{2N_l - N_s}{4} \\ x_{0,n-N_s} \cdot W_0(n-N_s) + x_{1,n-(N_l/2-N_s)} \cdot W_1(n-(N_l/2-N_s)) & \text{,for} \quad \frac{2N_l - N_s}{4} \le n < \frac{2N_l + N_s}{4} \\ x_{1,n-(N_l/2-N_s)} \cdot W_1(n-(N_l/2-N_s)) & \text{,for} \quad \frac{2N_l + N_s}{4} \le n < N_l - N_s \\ 0.0 & \text{,for} \quad N_l - N_s \le n < N_l \end{cases}$$

5.5.3.5.3 Transform splitting with LONG_START_SEQUENCE

The LONG_START_SEQUENCE in combination with transform splitting is depicted in Figure 11.

Figure 11 — Transform splitting for LONG_START_WINDOW

It comprises three windows defined as follows, where *N_l/2* is set to 1024 (768), *N_s* is set to 256 (192), respectively.

$$W_0(n) = \begin{cases} 1.0 & \text{,for} \quad 0 \le n < \frac{3N_l/2 - N_s}{4} \\ W_{0,RIGHT,N_s}(n - \frac{3N_l/2 - N_s}{4}) & \text{,for} \quad \frac{3N_l/2 - N_s}{4} \le n < \frac{3N_l/2 + N_s}{4} \\ 0.0 & \text{,for} \quad \frac{3N_l/2 + N_s}{4} \le n < N_l/2 \end{cases}$$

$$W_1(n) = \begin{cases} 0.0 & \text{, for} \quad 0 \le n < \frac{N_l/2 - N_s}{4} \\ W_{1,LEFT,N_s}(n - \frac{N_l/2 - N_s}{4}) & \text{, for} \quad \frac{N_l/2 - N_s}{4} \le n < \frac{N_l/2 + N_s}{4} \\ 1.0 & \text{, for} \quad \frac{N_l/2 + N_s}{4} \le n < \frac{3N_l/2 - N_s}{4} \\ W_{1,RIGHT,N_s}(n - \frac{3N_l/2 - N_s}{4}) & \text{, for} \quad \frac{3N_l/2 - N_s}{4} \le n < \frac{3N_l/2 + N_s}{4} \\ 0.0 & \text{, for} \quad \frac{3N_l/2 + N_s}{4} \le n < N_l/2 \end{cases}$$

The left/right window halves are given by:

$$W_{1,LEFT,N_s}(n) = \begin{cases} W_{KBD\_LEFT,N_s}(n), \text{if} \;\; \text{window\_shape} == 1 \\ W_{SIN\_LEFT,N_s}(n), \text{if} \;\; \text{window\_shape} == 0 \end{cases},$$

$$W_{(0,1),RIGHT,N_s}(n) = \begin{cases} W_{KBD\_RIGHT,N_s}(n), \text{if} \;\; \text{window\_shape} == 1 \\ W_{SIN\_RIGHT,N_s}(n), \text{if} \;\; \text{window\_shape} == 0 \end{cases}$$

The third window equals the left half of a LONG_START_WINDOW:

$$W_2(n) = \begin{cases} W_{LEFT,N_l}(n) & \text{, for} \quad 0 \le n < N_l/2 \\ 1.0 & \text{, for} \quad N_l/2 \le n < N_l \end{cases}$$

with

$$W_{LEFT,N_l}(n) = \begin{cases} W_{KBD\_LEFT,N_l}(n), \;\; \text{if} \;\; \text{window\_shape\_previous\_block} == 1 \\ W_{SIN\_LEFT,N_l}(n), \;\; \text{if} \;\; \text{window\_shape\_previous\_block} == 0 \end{cases}$$

The overlap and add between the two half-length windows resulting in intermediate windowed time domain values $\tilde{Z}_{i,n}$ is specified as follows. Here, N_l is set to 2048 (1536), N_s to 256 (192) respectively:

$$\tilde{Z}_{i,n}(n) = \begin{cases} sign \cdot (x_{0,3N_s-1-n} \cdot W_0(3N_s - 1 - n) + x_{1,N_s-1-n} \cdot W_1(N_s - 1 - n)) & \text{, for} \quad 0 \le n < N_s \\ x_{0,n+N_s} \cdot W_0(n - N_s) & \text{, for} \quad N_s \le n < \frac{2N_l - N_s}{4} \\ x_{0,n-N_s} \cdot W_0(n - N_s) + x_{1,n-(N_l/2-N_s)} \cdot W_1(n - (N_l/2 - N_s)) & \text{, for} \quad \frac{2N_l - N_s}{4} \le n < \frac{2N_l + N_s}{4} \\ x_{1,n-(N_l/2-N_s)} \cdot W_1(n - (N_l/2 - N_s)) & \text{, for} \quad \frac{2N_l + N_s}{4} \le n < N_l - N_s \\ 0.0 & \text{, for} \quad N_l - N_s \le n < N_l \end{cases}$$

where *sign* = –1 if prev_aliasing_symmetry == 0, or *sign* = 1 otherwise. The final windowed time-domain values $Z_{i,n}$ are obtained by applying W_2:

$$Z_{i,n}(n) = \tilde{Z}_{i,n}(n) \cdot W_2(n), \qquad \text{for} \quad 0 \le n < N_l$$

5.5.3.6 Modification to complex prediction stereo decoding

Since the FD stereo tools operate on an interleaved pseudo-spectrum when TS is active in a channel pair, no changes are necessary to the underlying M/S or complex prediction processing. However, the derivation of the previous frame's downmix dmx_re_prev[] and the computation of the downmix MDST dmx_im[] in ISO/IEC 23003-3:2012, subclause 7.7.2 need to be adapted if TS is used in either channel in the last or current frame:

— use_prev_frame shall be 0 if the TS activity changed in either channel from last to current frame. In other words, dmx_re_prev[] shall not be used in that case due to transform length switching.

— If TS was or is active, dmx_re_prev[] and dmx_re[] are interleaved pseudo-spectra and shall be de-interleaved into their corresponding two half-length TS spectra for correct MDST calculation.

— Upon TS activity, 2 half-length MDST downmixes are computed using adapted filter coefficients (filter_coefs in Table 82 and Table 83) and interleaved into a full-length spectrum dmx_im[] (just like dmx_re[]).

— window_sequence: Downmix MDST estimates are computed for each group window pair. use_prev_frame is evaluated only for the first of the two half-window pairs. For the remaining window pair, the preceding window pair is always used in the MDST estimate, which implies use_prev_frame = 1.

— Window shapes: The MDST estimation parameters for the current window, which are filter coefficients as specified below, depend on the shapes of the left and right window halves. For the first window, this means that the filter parameters are a function of the current and previous frames' window_shape flags. The remaining window is only affected by the current window_shape.

Table 82 — MDST filter parameters for current window (filter_coefs)

Current window sequence	**Left half: sine shape right half: sine shape**	**Left half: KBD shape right half: KBD shape**
LONG_START_SEQUENCE STOP_START_SEQUENCE	[0.185618, 0.000000, 0.627371, 0.000000, -0.627371, 0.000000, -0.185618]	[0.204932, 0.000000, 0.634159, 0.000000, -0.634159, 0.000000, -0.204932]
Current window sequence	**Left half: sine shape right half: KBD shape**	**Left half: KBD shape right half: sine shape**
LONG_START_SEQUENCE STOP_START_SEQUENCE	[0.194609, 0.006202, 0.630536, 0.000000, -0.630536, -0.006202, -0.194609]	[0.194609, -0.006202, 0.630536, 0.000000, -0.630536, 0.006202, -0.194609]

Table 83 — MDST filter parameters for previous window (filter_coefs_prev)

Current window sequence	**Left half of current window: sine shape**	**Left half of current window: KBD shape**
LONG_START_SEQUENCE STOP_START_SEQUENCE	[0.069608, 0.075028, 0.078423, 0.079580, 0.078423, 0.075028, 0.069608]	[0.042172, 0.043458, 0.044248, 0.044514, 0.044248, 0.043458, 0.042172]

5.5.4 MPEG surround for mono to stereo upmixing

5.5.4.1 Calculation of pre-matrix M1 and mix-matrix M2

5.5.4.1.1 General

The calculation of mix-matrix M2, which is a interpolated version of $\mathrm{R}_2^{l,m}$, is done according to ISO/IEC 23003-3:2012, 7.11.2.3, but with the modifications specified in the following subclause.

5.5.4.1.2 Upmix without decorrelation

In case a format conversion step is included, the MPEG surround for mono to stereo upmixing tool is modified as follows.

Subclause 10.3.4 defines a downmixing matrix M_{Dmx} which is used to calculate a mix matrix M_{Mix} as follows. Here N_{in} is the number of source channels and N_{out} is the number of destination channels.

```
M_Mix = zero N_in x N_in Matrix
for i = 1 to N_out
   for j = 1 to N_in
      set_j = 0
      if  M_Dmx(i, j) > 0.0
         set_j = 1
      end
      for k = 1 to N_in
         set_k = 0
         if  M_Dmx(i, k) > 0.0
            set_k = 1
         end
         if  set_j == 1 and set_k == 1
            M_Mix(j, k)= 1
         end
      end
   end
end
```

Each OTT decoding block yields two output signals corresponding to channel number *i* and *j*. If the mix matrix $\mathrm{M}_{\mathrm{Mix}}(i,j)$ equals one, decorrelation is switched off for this decoding block which means that the elements $H11_{OTT}^{l,m}$ and $H21_{OTT}^{l,m}$ of the upmix matrix $\mathrm{R}_2^{l,m}$ shall be calculated by setting $ICC^{l,m}=1$.

5.5.4.2 Combined parametric and residual decoding (hybrid residual coding)

5.5.4.2.1 Overview

In addition to using either decorrelator based mono to stereo upmixing or residual coding as specified in ISO/IEC 23003-3:2012, 7.11.1, hybrid residual coding allows a signal dependent combination of both modes. The residual signal and the decorrelator outputs are blended together, using time and frequency dependent weighting factors depending on the signal energies and the spatial parameters, as illustrated in the schematics below (Figure 12).

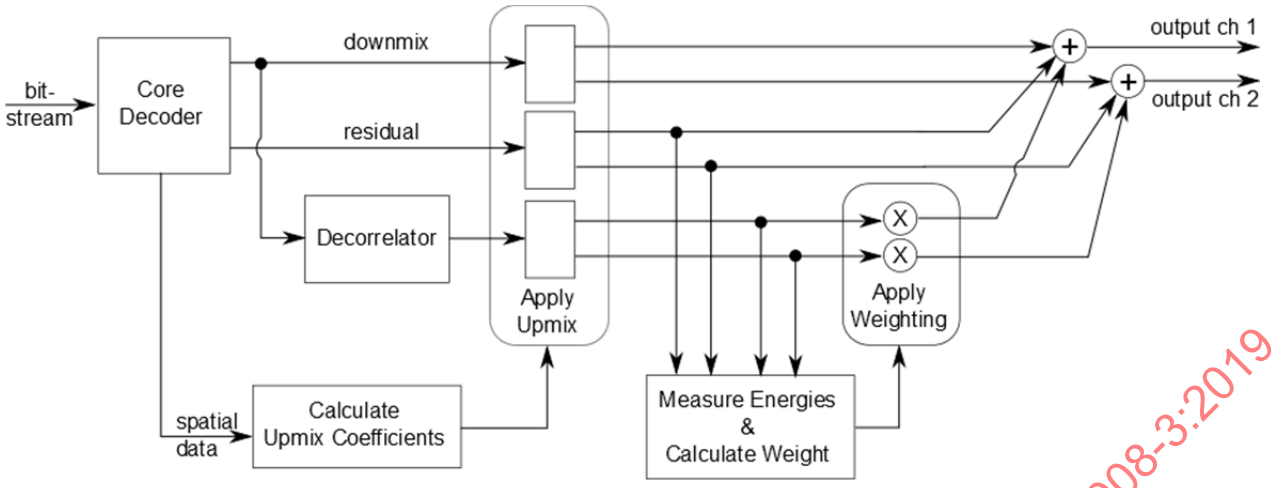


Figure 12 — Schematics of hybrid residual decoder

5.5.4.2.2 Decoding process

Hybrid residual coding mode is indicated by the syntax elements **bsResidualCoding** == 1 and **bsResidualBands** == 1 in Mps212Config(). The calculation of mix-matrix M2 is performed as if **bsResidualCoding** == 0, following the calculation in ISO/IEC 23003-3:2012, subclause 7.11.2.3. The matrix $\mathrm{R}_2^{l,m}$ for the decorrelator based part is defined as:

$$\mathrm{R}_2^{l,m} = \begin{bmatrix} H11_{OTT}^{l,m} & H12_{OTT}^{l,m} \\ H21_{OTT}^{l,m} & H22_{OTT}^{l,m} \end{bmatrix}$$

The upmixing process is split up into Downmix, decorrelator output and residual. The upmixed Downmix u_{dmx} is calculated using:

$$\mathrm{R}_{2,\mathrm{dmx}}^{l,m} = \begin{bmatrix} H11_{OTT}^{l,m} & 0 \\ H21_{OTT}^{l,m} & 0 \end{bmatrix}$$

The upmixed decorrelator output u_{dec} is calculated using:

$$\mathrm{R}_{2,\mathrm{dec}}^{l,m} = \begin{bmatrix} 0 & H12_{OTT}^{l,m} \\ 0 & H22_{OTT}^{l,m} \end{bmatrix}$$

The upmixed residual signal u_{res} is calculated using:

$$\mathrm{R}_{2,\mathrm{res}}^{l,m} = \begin{bmatrix} 0 & H12_{RES}^{l,m} \\ 0 & H22_{RES}^{l,m} \end{bmatrix} = \begin{bmatrix} 0 & \max\{0.5, H11_{OTT}^{l,m}\} \\ 0 & -\max\{0.5, H21_{OTT}^{l,m}\} \end{bmatrix}$$

In case a format conversion step is included the residual upmixing matrix $\mathrm{R}_{2,\mathrm{res}}^{l,m}$ shall be modified as follows.

Each OTT decoding block yields two output signals corresponding to channel number *i* and *j*. If the mix matrix $\mathrm{M}_{\mathrm{Mix}}(i,j)$ equals one ($\mathrm{M}_{\mathrm{Mix}}$ defined in subclause 5.5.4.1.2), the residual upmixing matrix $\mathrm{R}_{2,\mathrm{res}}^{l,m}$ shall be set to:

$$R_{2,res}^{l,m} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

The energies of the upmixed residual signal E_{res} and of the upmixed decorrelator output E_{dec} are calculated per hybrid band as sum over both output channels ch and all timeslots ts and of one frame as:

$$E_{res} = \sum_{ch}\sum_{ts} \left\| u_{res}(ch, ts) \right\|$$

$$E_{dec} = \sum_{ch}\sum_{ts} \left\| u_{dec}(ch, ts) \right\|$$

The upmixed decorrelator output is weighted using a weighting factor r_{dec} calculated for each hybrid band per frame as:

$$r_{dec} = \begin{cases} 0 & \text{if} \quad E_{res} > E_{dec} \\ 1 & \text{if} \quad E_{res} < \varepsilon \\ \sqrt{\left| \dfrac{E_{dec} - E_{res} + \varepsilon}{E_{dec} + \varepsilon} \right|} & \text{otherwise} \end{cases}$$

With ε a small number to prevent division by zero (ε = 1e-9).

All three upmix signals are added to form the decoded output signal.

5.5.5 Enhanced noise filling

5.5.5.1 General

The enhanced noise filling is achieved through a tool named intelligent gap filling (IGF). IGF extends the noise filling in the decoder to alternatively exploit neighbouring spectral portions, predefined source tiles, for filling spectral gaps in predefined target tiles. These gaps originate from coarse quantization in the encoder. Using these tiles for gap filling often provides a perceptually better match than just injecting random noise. The additional side information, igf_data(), conveys control data, e.g. tile source and tile target information, tile target level, etc.

5.5.5.2 Data elements

indepFlag the MPEG-H 3D audio independency flag.

pred_dir indicates the direction of prediction (see ISO/IEC 23003-3:2012, 7.7.2)

ms_used[ ][ ] one-bit flag per scalefactor band indicating that M/S coding or prediction is being used in windowgroup g and scalefactor band sfb, shared with the joint stereo tool (ISO/IEC 23003-3:2012 subclause 7.7).

cplx_pred_used[g][sfb]	One-bit flag per window group g and scalefactor band sfb (after mapping from prediction bands), shared with the complex stereo prediction tool (see ISO/IEC 23003-3:2012, subclause 7.7.2).

5.5.5.3 Helper elements

igfStartSfbLB	the IGF start scalefactor band index used with a long window sequence.
igfStartSfbSB	the IGF start scalefactor band index used with a short window sequence.
igfStopSfbLB	the IGF stop scalefactor band index used with a long window sequence.
igfStopSfbSB	the IGF stop scalefactor band index used with a shortwindow sequence.
igfMin	is a subband index; this index is sampling frequency and core coder framelength dependent and determine a minimal frequency which is used to assign a source tile range.
igfBgn	the IGF start subband; this helper element is used for both, long and short window sequences and is mapped for every frame.
igfEnd	the IGF stop subband; this helper element is used for both, long and short window sequences and is mapped for every frame.
igfP	is 1 if the flag **igfUseHighRes** is set to one, 2 otherwise.
m_igfStartSfb	the mapped IGF start scalefactorband; this helper element is used for both, long and short window sequences and is mapped for every frame.
m_igfStopSfb	the mapped IGF stop scalefactorband; this helper element is used for both, long and short window sequences and is mapped for every frame.
tile[ ]	vector of length 4 containing width information.
igfNTiles	number of target tiles in IGF range (as calculated according to Figure 14).
sbs	start subband of the current target tile.
rng	length of the IGF range in subbands.
ch	channel in scope, this value is either 0 or 1.
sfb	scalefactor band in scope.
num_windows[ch]	contains the current number of windows.
num_window_groups[ch]	contains the number of window groups.
sb	index of the source subband.
tb	index of the target subband.
tileIdx	index of the IGF target tile in which a tb under scope is located.
group_len	identifies the actual length of a window group.

igf_sN[ ]	vector containing information on energies per scalefactor which have been not quantized to zero.
igf_pN[ ]	vector containing information on energies per scalefactor which will be copied from the source tile range.
wg, wa	counter for the actual window (0,1,...,7).
w	the window index in a window group.
window_sequence	signals the USAC window sequence.
isShortWindow	1 if window_sequence == EIGHT_SHORT_SEQUENCE, else 0.
pMDCT[ ]	array containing the transform spectrum.
pMDCT_flat[ ]	array containing the whitened transform spectrum.
width	number of lines in the sfb under scope.
E	energy for the current subband.
val	spectral values for the current IGF subband.
noise_offset	noise-fill offset to modify scale factors of zero-quantized bands (see ISO/IEC 23003-3:2012, subclause 7.2).
noise_level	noise-fill level representing amplitude of added spectrum noise (see ISO/IEC 23003-3:2012, subclause 7.2).
stereo_filling	flag indicating whether SF is utilized in the current frame and channel.

5.5.5.4 IGF signal processing

5.5.5.4.1 Mapping of IGF bitstream elements

5.5.5.4.1.1 Helper elements

swb_offset_long[ ]	offset table for scalefactor bands to use with long windows.
swb_offset_short[ ]	offset table for scalefactor bands to use with short windows.
num_swb_short_window	number of scalefactor bands with a short window sequence.
num_swb_long_window	number of scalefactor bands with a long window sequence.
sampleRate	sampling rate of the core coder.
bl	length of the current block in dependency of the window sequence.

5.5.5.4.1.2 Detailed description

The bitstream elements **igfStartIndex** and **igfStopIndex** are mapped to scale factor band indices:

```
igfStartSfbLB = min(11 + igfStartIndex, num_swb_long_window - 5)
```

If **igfStopIndex** is not 15, calculate:

```
igfStopSfbLB = min(num_swb_long_window, max(igfStartSfbLB + (((num_swb_long_window -
(igfStartSfbLB + 1)) * (igfStopIndex + 2)) >> 4), igfStartSfbLB + 1))
```

If **igfStopIndex** equals 15, set:

```
igfStopSfbLB = num_swb_long_window
```

For the appropriate start and stop boundaries for IGF with short window sequence calculate:

```
igfStartSfbSB = -1;
for (sfb = 0; sfb < num_swb_short_window; sfb++) {
  if (swb_offset_short[sfb] >= swb_offset_long[igfStartSfbLB] >> 3) {
    if (igfStartSfbSB < 0) igfStartSfbSB = sfb;
  }
}
igfStopSfbSB = -1;
for (sfb = 0; sfb < num_swb_short_window; sfb++) {
  if (swb_offset_short[sfb] >= swb_offset_long[igfStopSfbLB] >> 3) {
    if (igfStopSfbSB < 0) igfStopSfbSB = sfb;
  }
}
```

The final mapping to m_igfStartSfb, igfBgn and m_igfStopSfb, igfEnd depends on the window sequence in the actual processed frame.

```
if (window_sequence == EIGHT_SHORT_SEQUENCE) {
  m_igfStartSfb = igfStartSfbSB;
  m_igfStopSfb  = igfStopSfbSB;
  igfBgn = swb_offset_short[m_igfStartSfb];
  igfEnd = swb_offset_short[m_igfStopSfb];
  swb_offset = swb_offset_short;
} else {
  m_igfStartSfb = igfStartSfbLB;
  m_igfStopSfb  = igfStopSfbLB;
  igfBgn = swb_offset_long[m_igfStartSfb];
  igfEnd = swb_offset_long[m_igfStopSfb];
  swb_offset = swb_offset_long;
}
```

In the context of IGF processing the frequency band offset tables as defined in ISO/IEC 14496-3:2009, 4.5.4 shall be extended such that swb_offset_short[num_swb_short_window] equals half the window length of the EIGHT_SHORT_SEQUENCE, e.g. 128, and swb_offset_long[num_swb_long_window] equals half the window length of the each other sequence, e.g. 1024.

If the bitstream element **igfUseHighRes** equals zero, the IGF frequency resolution will be reduced by pairing IGF scalefactor bands.

igfMin is the lowest IGF source subband and it is calculated as follows:

$$igfMin = sb + (sb \bmod 2)$$

where

$$sb = INT(1125 * bl * (2 / sampleRate))$$

The helper element bl is mapped according to Table 84:

Table 84 — Value of bl

window_sequence	bl
EIGHT_SHORT_SEQUENCE	ccfl / 8
medium TCX	ccfl / 2
All other sequences	ccfl

5.5.5.4.2 Computing IGF tiles

IGF works with so called tiles to determine target regions shown in Figure 13.

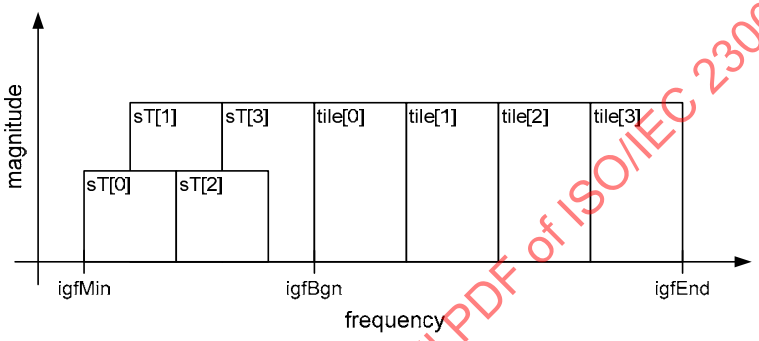


Figure 13 — Computing target tiles, overview

The flowchart in Figure 14 produces the tile vector tile[ ] of length 4 containing the width of each IGF target tile. Please note that this flowchart is used for both, short and long window sequences.

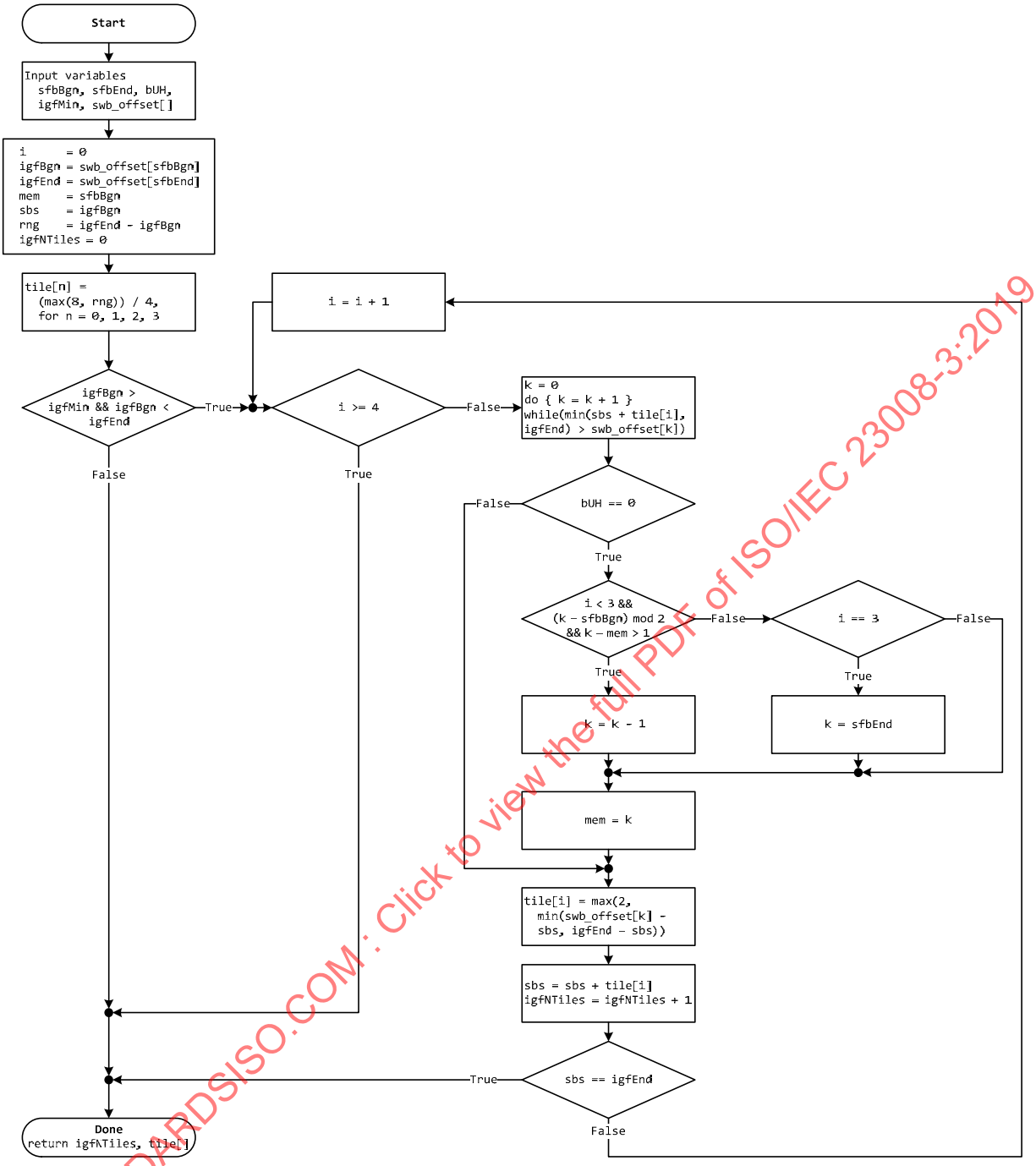


Figure 14 — Flowchart for calculation of tile width and number of tiles

5.5.5.4.3 Decoding of IGF levels

5.5.5.4.3.1 Helper elements

igf_curr[ch][ ][ ]	vector containing the IGF levels for the current window.
igf_prev[ch][ ]	vector containing the IGF levels for the previous window.
igf_arith_t[ch]	time index, to be increment per window since the last reset.
igf_prevD[ch]	context form the window before the previous window.

prev_num_windows[ch] contains the previous number of windows.

core_mode_prev[ch] contains the previous **core_mode**[ch].

5.5.5.4.3.2 Decoding process

The subroutine **igf_level()** is performed as outlined below. Please note that all helper elements associated with **igf_level()** needs their own instance of context memory per audio element (CPE, SCE) and per channel ch in case of CPE, for proper decoding.

First of all, the context memory (igf_prev, igf_prevD, igf_arith_t) of the IGF arithmetic coder, together with other helper elements, needs a reset under the following conditions:

```
if (   igf_AllZero
    || indepFlag
    || ((num_windows[ch] != prev_num_windows[ch]) && core_mode[ch] == 0)
    || ((lpd_mode[ch]    != last_lpd_mode[ch])    && core_mode[ch] == 1)
    || (core_mode[ch]    != core_mode_prev[ch])) {
  igf_curr[ch] = {0};
  igf_prev[ch] = {0};
  igf_arith_t[ch] = 0;
  igf_prevD[ch] = 0;
}
prev_num_windows[ch] = num_windows[ch];
core_mode_prev[ch] = core_mode[ch];
```

If the bitstream element **igf_AllZero** is not true, the subroutine igf_arith_decode() (see subclause 5.5.5.4.4.2) is called as outlined below:

```
if (!igf_AllZero) {
  for (g = 0; g < num_window_groups; g++) {
    igf_curr[ch][g] = igf_arith_decode(igf_arith_t[ch], igf_prev[ch], igf_prevD[ch])
    igf_prevD[ch] = igf_prev[m_igfStartSfb];
    for (sfb = m_igfStartSfb; sfb < m_igfStopSfb; sfb++) {
      igf_prev[ch][sfb] = igf_curr[ch][g][sfb];
      igf_curr[ch][g][sfb] = igf_curr[ch][g][sfb] * igFP;
    }
    igf_arith_t[ch]++;
  }
  arith_decode_flush(); /* push back 14 bits to the bitstream */
}
```

The result, quantized energy information of scalefactor bands in the IGF region, is stored in igf_curr[ ][ ][ ]. For requantization use the formula:

```
igf_curr[ch][g][sfb] = 2^((igf_curr[ch][g][sfb]-igf_emphasis)*0.25)
```

for each channel ch, window group g and scalefactor band sfb in scope and where igf_emphasis is defined as:

```
igf_emphasis = 0;  if IGF is running in FD  mode
igf_emphasis = 40; if IGF is running in TCX mode
```

5.5.5.4.4 Arithmetic decoding of IGF average levels

5.5.5.4.4.1 Helper elements

nBits	number of bits to read.
nBitRead	decoded number read, of length nBits bits.
cfTable[ ]	cumulative frequency table.
tableOffset	offset to frequency table.
decRes	decoded prediction residual.
extra	the position of the residual in one tail of the distribution.
pred	predicted value computed using the neighbours.
t	time index since the last reset.
prevD	first IGF scf value from the previous frame.
ctx	index to the context containing the probability distribution.

5.5.5.4.4.2 Decoding process

The IGF scalefactors are encoded by using the function arith_decode() as in ISO/IEC 23003-3:2012 subclause 7.4.3, and by using new probability tables (cf_se01, cf_se02[ ], cf_se10, cf_se20[ ], cf_se11[ ][ ], cf_off_se01, cf_off_se02[ ], cf_off_se10, cf_off_se20[ ] and cf_off_se11[ ][ ]), see Annex A.

The subroutine **igf_level()**, see Table 50, provides a vector containing the IGF average energy information per IGF scalefactor, called SFE, of the transform spectral lines for each scale factor band or group of scale factor bands.

The SFEs from up to two of the previous frames and the already decoded SFEs from the current frame are taken into account in deriving a context providing the probability distribution for coding. In each context a fixed linear predictor is employed, based on the same data as used for the quantized context.The prediction residuals instead of the original values are decoded. For large prediction residuals, outside of the centre of the coding distribution, escape coding is used.

```
arith_decode_bits(nBits)
{
    cf_for_bit[2] = {8192, 0};
    nBitRead = 0;
    for (i = nBits - 1; i >= 0; --i) {
       bit = arith_decode(cf_for_bit, 2);
       nBitRead = nBitRead + (bit << i);
    }
    return nBitRead;
}
```

The helper function arith_decode_bits() uses the USAC arithmetic decoder function arith_decode() to obtain a value of length nBits bits from the bitstream.

```
arith_decode_residual(cfTable, tableOffset)
{
    val = arith_decode(cfTable, 27);
    if ((val != 0) && (val != 26)) {
        decRes = val - 13;
    } else {
        extra = arith_decode_bits(4);
        if (extra == 15) {
            extra = 15 + arith_decode_bits(7);
        }
        if (val == 0) {
            decRes = -13 - extra;
        } else {
            decRes = 13 + extra;
        }
    }
    decRes -= tableOffset;
    return decRes;
}
```

The helper function arith_decode_residual() returns the decoded residual value, which has to be added to the predicted value (pred) to obtain the original value.

```
igf_arith_decode(t, igf_prev, prevD)
{
    igf_prev += m_igfStartSfb;
    igf_curr = {0};
    igf_curr += m_igfStartSfb;
    igfInc = igfP;
    if (isShortBlock) igfInc = 1;
    for (f = 0; f < m_igfStopSfb - m_igfStartSfb; f += igfInc) {
        if (t == 0) {
            if (f == 0) {
                igf_curr[f] = arith_decode_bits(7);
            } else if (f == igfInc) {
                pred = igf_curr[f - igfInc];
                igf_curr[f] = pred + arith_decode_residual(cf_se01, cf_off_se01);
            } else {
                pred = igf_curr[f - igfInc];
                ctx = quant_ctx(igf_curr[f - igfInc] - igf_curr[f - 2 * igfInc]);
                igf_curr[f] = pred + arith_decode_residual(
                    cf_se02[3 + ctx], cf_off_se02[3 + ctx]);
            }
        } else if (f == 0) {
            if (t == 1) {
                pred = igf_prev[f];
                igf_curr[f] = pred + arith_decode_residual(cf_se10, cf_off_se10);
            } else {
                pred = igf_prev[f];
                ctx = quant_ctx(igf_prev[f] - prevD);
                igf_curr[f] = pred + arith_decode_residual(
                    cf_se20[3 + ctx], cf_off_se20[3 + ctx]);
            }
        } else {
            pred = igf_prev[f] + igf_curr[f - igfInc] - igf_prev[f - igfInc];
            ctx_f = quant_ctx(igf_prev[f] - igf_prev[f - igfInc]);
            ctx_t = quant_ctx(igf_curr[f - igfInc] - igf_prev[f - igfInc]);
            igf_curr[f] = pred + arith_decode_residual(
                cf_se11[3 + ctx_t][3 + ctx_f],
```

```
                cf_off_se11[3 + ctx_t][3 + ctx_f]);
        }
        for (z = f + 1; z < min(f + igfInc, m_igfStopSfb - m_igfStartSfb); ++z) {
            igf_curr[z] = igf_curr[f];
        }
    }
    igf_curr -= m_igfStartSfb;
    return igf_curr;
}
```

The quantization function quant_ctx(), limits large integer values to ±3. It is defined as:

quant_ctx(x) = x; for |x| ≤ 3 and

quant_ctx(x) = 3 · sign(x); for |x| > 3

5.5.5.4.5 Applying IGF

5.5.5.4.5.1 General

Application of IGF shall occur after the following processing steps:

a) inverse quantization of MDCT values;

b) if applicable: noise filling (as specified in ISO/IEC 23003-3:2012, 7.2), but only up to subband swb_offset[m_igfStartSfb]-1;

c) if applicable: joint stereo coding (as specified in ISO/IEC 23003-3:2012, 7.7), but only up to subband swb_offset[max_sfb_ste]-1;

d) if the bitstream element **igfUseEnf** is 1, IGF envelope noise flattening shall be applied in the IGF decoding process.

Please note that the temporal noise shaping (TNS) tool was extended in case of IGF to additionally perform temporal tile shaping (TTS) on IGF tiles. Therefore, the TNS shaping filters are also applied on IGF generated frequency tiles if indicated by **igfAfterTnsSynth**. Figure 15 shows the position of the IGF tool in dependency of **igfAfterTnsSynth** in the core coder.

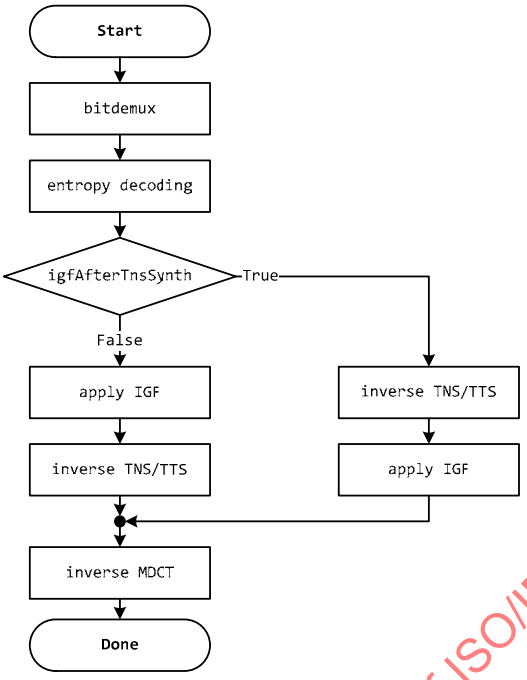


Figure 15 — Position of IGF in the core coder

5.5.5.4.5.2 Helper elements

top TNS stop band, see 14496-3:2009 subclause 4.6.9

5.5.5.4.5.3 Decoding process

There are two options to apply IGF, signalled with **igfAfterTnsSynth**:

— before TNS synthesis; IGF is applied to the TNS residual, and subsequently TNS/TTS is applied.

— after TNS synthesis; IGF is applied to the fully reconstructed and TNS filtered core coder signal.

Please note that the behaviour of TNS changes in case of IGF. The following code sequence adjusts the TNS stop band according to the IGF start and stop scale factor band offset indices:

```
nbands = max_sfb;
if (enhancedNoiseFilling) {
  if (!igfAfterTnsSynth) {
    if (!isShortWindow) {
      if (igfStopSfbLB > nbands) nbands = igfStopSfbLB;
    } else {
      if (igfStopSfbSB > nbands) nbands = igfStopSfbSB;
    }
  } else {
    if (!isShortWindow) {
      if (igfStartSfbLB < nbands) nbands = igfStartSfbLB;
    } else {
      if (igfStartSfbSB < nbands) nbands = igfStartSfbSB;
    }
  }
  top = MIN(top, nbands);
}
```

5.5.5.4.5.4 Creation of source tile spectra with independent noise filling

If the bitstream element **igfUseEnf** equals 1, *nT* copies of the dequantized MDCT core-coder spectrum shall be created and noise filling shall be applied for each copy observing the descriptions and restrictions defined in subclauses 5.5.5.4.5 and 5.5.5.4.9. With **igfUseEnf** active the *nT* MDCT core-coder spectra with different applied noise filling are the dedicated sources for further IGF processing where every `tileIdx` uses one of the created copies.

5.5.5.4.6 Computing a source subband in IGF

5.5.5.4.6.1 Helper elements

oS	offset between sb and tb.
nST	number of source tiles.
src	length of the IGF source range in subbands.

5.5.5.4.6.2 Detailed description

While applying IGF, target subbands (tb) are identified which have been quantized to zero by the encoder. For each of those target subbands a source subband (sb) is obtained with Figure 16. Note that for each element and each channel, there could be a different set of igfCurrTileIdx obtained from the bitstream.

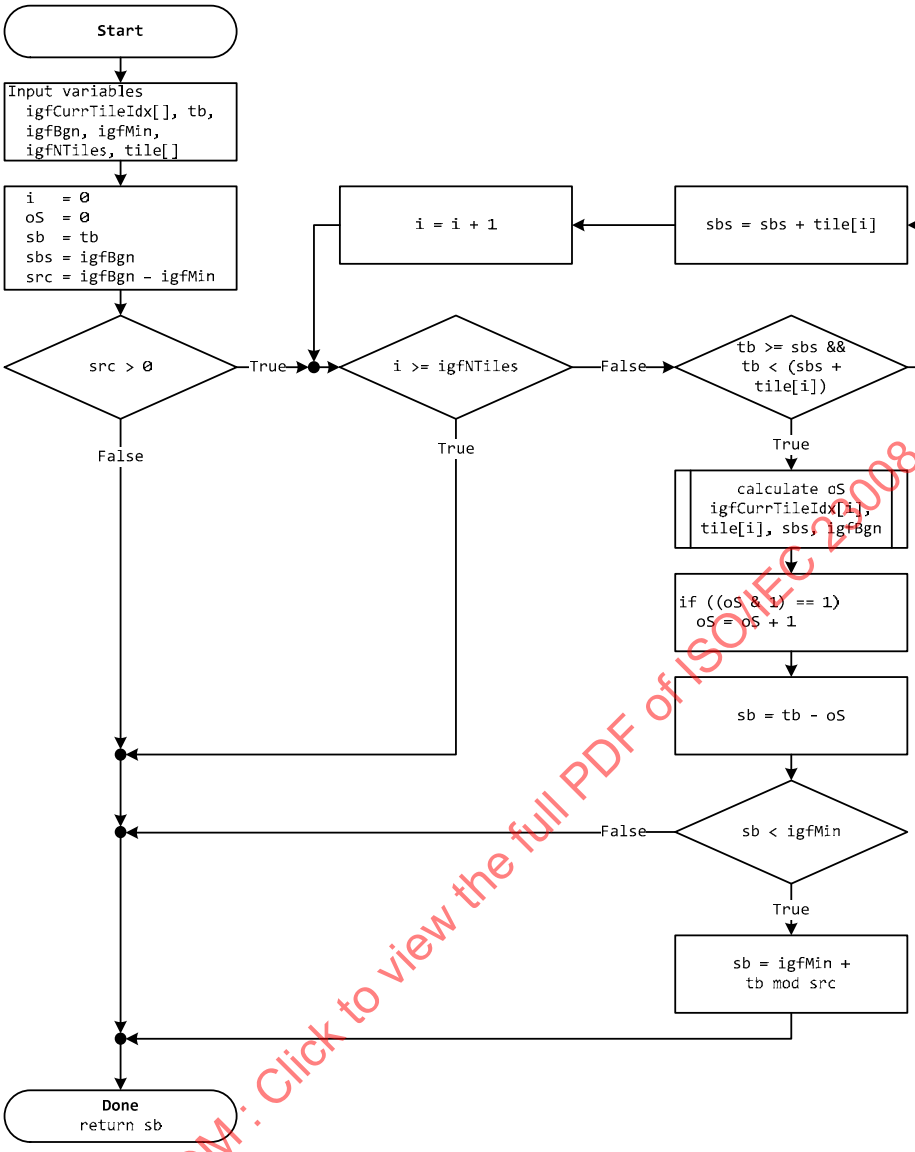


Figure 16 — Flowchart for computing a source subband sb

The function “calculate oS()” returns the offset oS between sb and tb using the following formula:

$$oS = INT((-0.5 * \textbf{igfCurrTileIdx}[ch][i] + 2.5) * tile[i] + sbs - igfBgn)$$

where tile[ ] is a vector with a maximum length of 4 containing the width of each IGF target tile according to subclause 5.5.5.4.2.

The helper function get_IGF_tile_idx() identifies the tile where tb is located.

```
get_IGF_tile_idx(tb, igfBgn, igfNTiles, tile[]) {
  sbs = igfBgn;
  for (tileIdx = 0; tileIdx < igfNTiles; tileIdx ++) {
    sbs += tile[tileIdx];
    if (tb < sbs) break;
  }
  return tileIdx;
}
```

5.5.5.4.6.3 Requirements

The computing of a source subband sb is constrained to the following requirement for the transmitted **igfCurrTileIdx**.

igfCurrTileIdx[ch][i] + nST[i] >= 4, for i = 0, 1, .., igfNTiles

The vector nST shall be computed as follows:

nST[i] = max(1, min((src / (tile[i] / 2)) – 1, 4)), for i = 0, 1, ..., igfNTiles

5.5.5.4.7 Spectral whitening in IGF

To reduce tonality or spectral tilt of an IGF tile, the following procedure uses the MDCT signal, if the bitstream element **igfUseWhitening** is equal to 1, to produce the whitened spectrum pMDCT_flat[ ], which is further used in IGF_mono() and IGF_stereo().

The IGF whitening is performed by dividing the spectrum in pMDCT[] by the scaled square root of its estimated spectral envelope in env[]:

$$pMDCT_{flat}[k] = pMDCT[k] \cdot \frac{2^{21}}{\sqrt{env[k]}}, k = igfMin, igfMin+1, \ldots, igfBgn-1$$

where the spectral envelope is estimated by filtering the squared spectrum using a moving average filter of length 7 where each filter coefficient equals 1.

```
igf_apply_whitening(pMDCT[], pMDCT_flat[])
{
  stop = swb_offset[m_igfStartSfb];
  for (i = igfMin; i < stop - 3; i++) {
    env = 1e-3;
    for (j = i - 3; j <= i + 3; j++) {
      env += pMDCT[j] * pMDCT[j];
    }

    n = FLOOR(log(env)/log(2)); /* C/C++ (see NOTE): frexp(env, &n); n--; */
    fac = pow(2, 21 - 0.5 * n);
    pMDCT_flat[i] = pMDCT[i] * fac;
  }

  for (; i < stop; i++) {
    pMDCT_flat[i] = pMDCT[i] * fac;
  }
}
```

NOTE The estimation of the spectral envelope comprises an approximation kernel allowing for an efficient implementation:

$$n = \lfloor \log_2(env) \rfloor = \left\lfloor \frac{\log_{10}(env)}{\log_{10}(2)} \right\rfloor$$

This kernel calculates the integral exponent for 2 of env. As a guidance it is noted that the standardized function frexp() as defined in ISO/IEC 9899 (Reference [14]) is well suited for the given operation. The function frexp() breaks the floating-point number env into a normalized fraction (s) in the range $[0.5, 1)$ or zero, and an integral power of 2 (n), such that:

$$env = s \cdot 2^n$$

In the given context the returned significant (s) is not used, the calculated exponent (n) needs to be decremented by 1.

5.5.5.4.8 Single/dual channel processing in IGF

5.5.5.4.8.1 Helper elements

randomSign() function returning (pseudo) random sign as defined in ISO/IEC 23003-3:2012, subclause 7.2.4 using latest seed state from USAC noise filling.

5.5.5.4.8.2 Decoding process

If IGF is applied in the discrete channel mode, e.g. in a SCE element or in a CPE element which does not make use of joint stereo feature (**igfIndependentTiling** is true) the following calling sequence shall be applied.

```
IGF_mono(ch, num_window_groups, group_len) {
  if (!isShortWindow && igfUseWhitening) {
    igf_apply_whitening (pMDCT, pMDCT_flat);
  }
  wg = wa = 0;
  for (g = 0; g < num_window_groups; g++) {
    igf_sN[ch] = {0};
    igf_pN[ch] = {0};
for (w = 0; w < group_len[g]; w++) {
  IGF_calc_mono(ch, wg, group_len[g], nfSeed1);
  Wg++;
}
for (w = 0; w y group_len[g]; w++) {
  IGF_apply_mono(ch, wa, nfSeed2);
       wa++;
    }
  }
}
```

The initial value of the pseudo-random noise seeds nfSeed1 and nfSeed2 shall be equal in order to synchronize the pseudo-random noise generator between IGF_calc_mono() and IGF_apply_mono().

IGF_calc_mono() is used to compute the vectors igf_sN[ ] and igf_pN[ ] respectively. The subroutine get_IGF_sb() is specified in subclause 5.5.5.4.6. Note that igfCurrTileIdx[ch] is a vector of length 4, describing the current tile indices of the current channel ch, where ch is 0 in a SCE and ch is 0 or 1 in a CPE.

```
IGF_calc_mono(ch, w, group_len, nfSeed1) {
  igfInc = igfP;
  if (isShortBlock) igfInc = 1;
  for (sfb = m_igfStartSfb; sfb < m_igfStopSfb; sfb += igfInc) {
    width = (swb_offset[MIN(sfb + igfInc, m_igfStopSfb)] - swb_offset[sfb]);
    E = 0;
    for (bin = 0; bin < width; bin++) {
      tb = swb_offset[sfb]+bin;
      E += pMDCT[w][tb] * pMDCT[w][tb];
    }
    igf_sN[ch][sfb] += E/group_len;
    E = 0;
    for (bin = 0; bin < width; bin++) {
      tb = swb_offset[sfb]+bin;
      if (pMDCT[w][tb] == 0) {
        sb = get_IGF_sb(igfCurrTileIdx[ch], tb);
        val = pMDCT[w][sb];
        if (!isShortWindow && igfUseWhitening) {
```

```
          tileIdx = get_IGF_tile_idx(tb);
          if (igf_WhiteningLevel[tileIdx] == 0) {
            val = pMDCT_flat[w][sb];
          }
          if (igf_WhiteningLevel[tileIdx] == 2) {
            val = randomSign(nfSeed1)*pow(2,21);
          }
        }
        E += val * val;
      }
    }
    igf_pN[ch][sfb] += E/group_len;
  }
}
```

IGF_apply_mono() will fill spectral gaps with previous calculated values:

```
IGF_apply_mono(ch, w, nfSeed2) {

  igfInc = igfP;
  if (isShortBlock) igfInc = 1;
  for (sfb = m_igfStartSfb; sfb < m_igfStopSfb; sfb+= igfInc) {
    width = (swb_offset[MIN(sfb + igfInc, m_igfStopSfb)] - swb_offset[sfb]);

    dE = igf_curr[sfb];
    sN = igf_sN[ch][sfb];
    pN = igf_pN[ch][sfb];
    mN = (dE*dE)*width -sN;

    if (mN > 0 && pN > 0) {
      gn = min(10, sqrt(mN/pN));
    } else {
      gn = 0;
    }
    for (bin = 0; bin < width; bin++) {
      tb = swb_offset[sfb]+bin;
      if (pMDCT[w][tb] == 0) {
        sb = get_IGF_sb(igfCurrTileIdx[ch], tb);
        val = pMDCT[w][sb];
        if (!isShortWindow && igfUseWhitening) {
          tileIdx = get_IGF_tile_idx(tb);
          if (igf_WhiteningLevel[tileIdx] == 0) {
            val = pMDCT_flat[w][sb];
          }
          if (igf_WhiteningLevel[tileIdx] == 2) {
            val = randomSign(nfSeed2)*pow(2,21);
          }
        }
        pMDCT[w][tb] = gn * val;
      }
    }
  }
}
```

5.5.5.4.9 Stereo filling in IGF

5.5.5.4.9.1 General

When stereo filling (SF) is active in a FD-only CPE, the MDCT coefficients of empty (i.e. fully zero-quantized) scale factor bands of the right (second) channel are replaced by a sum or difference of the corresponding decoded left and right channels MDCT coefficients of the previous frame. If USAC noise filling is active for the second channel, pseudo-random values are also added to each coefficient. The resulting coefficients of each scale factor band are then scaled such that the RMS (root of the mean coefficient square) of each band matches the value transmitted by way of that band's scale factor. See in ISO/IEC 23003-3:2012, subclause 7.3.

5.5.5.4.9.2 Helper elements

downmix_prev[ ][ ]	downmix (i.e. sum or difference) of the previous frame's left and right channels.
noiseFillingStartOffset	line index at or above which noise filling is used (ISO/IEC 23003-3:2012, 7.2).
sfbWidth[ ]	array containing the number of lines per sfb.
energy[ ]	energy of the signal per sfb.
energy_dmx[ ]	energy of the downmix signal per sfb.
spectrum[ ][ ]	MDCT spectrum for group g after noise filling (i.e. x_ac_invquant[g][ ][sfb][ ]).
window	indices of windows of group in scope, i.e. all windows belonging to group g.

5.5.5.4.9.3 Operational constraints

SF can only be used in the right FD channel of a common FD channel pair element (CPE), i.e. a channel pair element transmitting a StereoCoreToolInfo() with **common_window** == 1. Besides, due to its signaling, SF can only be applied when **noiseFilling** == 1 in mpegh3daCoreConfig(). If either of the channels in the pair is in LPD **core_mode**, SF is not used, even if the right channel is in FD mode.

5.5.5.4.9.4 Decoding process

The decoding of a joint-stereo coded FD channel with SF is executed in 3 sequential steps as follows.

Step 1: Decoding of stereo_filling

stereo_filling does not represent an independent bit-stream element but is derived from the noise-fill elements, noise_offset and noise_level, in a mpegh3daChannelPairElement() and the **common_window** flag in StereoCoreToolInfo(). If **noiseFilling** == 0 or **common_window** == 0 or the current channel is the left (first) channel in the element, stereo_filling is 0, and the stereo filling process ends. Otherwise,

```
if ((noiseFilling != 0) &&
    (noise_level == 0) &&
    (noise_offset != 0) &&
    (common_window != 0)) {
  stereo_filling = 1;
  noise_level    = (noise_offset & 28) / 4;
  noise_offset   = (noise_offset & 3) * 8;
}
else {
  stereo_filling = 0;
}
```

In other words, if stereo_filling == 1, **noise_offset** contains 5 bits of noise filling data, which are then rearranged. Since this operation alters the values of **noise_level** and **noise_offset**, it shall be performed before the noise filling process as in ISO/IEC 23003-3:2012, subclause 7.2. Moreover, the above pseudo-code is not executed in the left channel of a mpegh3daChannelPairElement() or any other element.

Step 2: Calculation of downmix_prev

downmix_prev[], the spectral mix which is to be used for stereo filling, is identical to the dmx_re_prev[] used for the MDST spectrum estimation in complex stereo prediction (see ISO/IEC 23003-3:2012, subclause 7.7.2.3). This means that:

— All coefficients of downmix_prev[] shall be zero if any of the channels of the frame and element with which the downmixing is performed – i.e. the frame before the currently decoded one – use **core_mode** == 1 (LPD) or if the channels use unequal transform lengths (**split_transform** == 1 or block switching to **window_sequence** == EIGHT_SHORT_SEQUENCE in only one channel).

— All coefficients of downmix_prev[] shall be zero during the stereo filling process if the channel's transform length changed from the last to the current frame (i.e. **split_transform** == 1 preceded by **split_transform** == 0, or **window_sequence** == EIGHT_SHORT_SEQUENCE preceded by **window_sequence** != EIGHT_SHORT_SEQUENCE, or vice versa resp.) in the current element or usacIndependencyFlag == 1.

— If transform splitting is applied in the channels of the previous or current frame, downmix_prev[] represents a line-by-line interleaved spectral downmix. See the transform splitting tool for details.

— If complex stereo prediction is not utilized in the current frame and element, pred_dir equals 0.

Consequently, the previous downmix only has to be computed once for both tools, saving complexity. The only difference between downmix_prev[] and dmx_re_prev[] in ISO/IEC 23003-3:2012, subclause 7.7.2 is the behaviour when complex stereo prediction is not currently used, or when it is active but **use_prev_frame** == 0. In that case, downmix_prev[] is computed for stereo filling decoding according to ISO/IEC 23003-3:2012, subclause 7.7.2.3 even though dmx_re_prev[] is not needed for complex stereo prediction decoding and is, therefore, undefined/zero.

Step 3: Stereo filling of empty scale factor bands

If stereo_filling == 1, the following procedure is carried out after the noise filling process in all initially empty scale factor bands sfb at or above noiseFillingStartOffset (see FD noise filling) and below max_sfb_ste, i.e. all bands in which all MDCT lines were quantized to zero. First, the energies energy[ ] and energy_dmx[ ] of the given sfb and the corresponding lines in downmix_prev[], respectively, are computed via sums of the squares.

```
energy_dmx[sfb] = 1e-8;
energy[sfb] = 0;
for (index = swb_offset[sfb]; index < swb_offset[sfb+1]; index++) {
  energy_dmx[sfb] += downmix_prev[window][index] * downmix_prev[window][index];
  spectrum[window][index] *= 4;
  energy[sfb] += spectrum[window][index] * spectrum[window][index];
}
```

Then, given sfbWidth[ ] containing the number of lines per sfb,


```
if (energy[sfb] < sfbWidth[sfb]) {
  tmp = min(10, sqrt((sfbWidth[sfb] - energy[sfb]) / energy_dmx[sfb]));
  factor = 0;
  for (index = swb_offset[sfb]; index < swb_offset[sfb+1]; index++) {
    spectrum[window][index] += downmix_prev[window][index] * tmp;
    factor += spectrum[window][index] * spectrum[window][index];
  }
  if ((factor != sfbWidth[sfb]) && (factor > 0)) {
    factor = min(10, sqrt(sfbWidth[sfb] / (factor + 1e-8)));
    for (index = swb_offset[sfb]; index < swb_offset[sfb+1]; index++) {
      spectrum[window][index] *= factor;
    }
  }
}
```

for the spectrum of each group window. Then the scale factors are applied on the resulting spectrum as in ISO/IEC 23003-3:2012, 7.3 with the scale factors of the empty bands being processed like regular scale factors.

Note that, unlike described in ISO/IEC 23003-3:2012, 7.3, **noise_offset** is allowed to be non-zero (i.e. scf[g][sfb] of an empty band can be adjusted by **noise_offset**) even if **noise_level** equals zero after Step 1.

5.5.5.4.10 MS and complex prediction processing in IGF

5.5.5.4.10.1 General

IGF stereo coding is applied to CPEs where **igfIndependentTiling** is zero and then replaces the channel-wise decoding in subclause 5.5.5.4.8.

The IGF joint stereo tool is based on the joint stereo tool, the decoding after filling is done the same way, in comparison to independent patching the residuals are transformed from the already decoded left/right values to the appropriate mid/side values for bands where joint stereo is active.

5.5.5.4.10.2 Helper elements

l_spec[ ]	Array containing the left channel spectrum of the respective channel pair.
r_spec[ ]	Array containing the right channel spectrum of the respective channel pair.
l_pMDCT_flat[ ]	Array containing the left channel whitened spectrum.
r_pMDCT_flat[ ]	Array containing the right channel whitened spectrum.
l_E, r_E	Energies for the current subband, for the left and right channel respectively.
l_igf_sN[ ], r_igf_sN[ ]	vectors containing information on energies per scalefactor which have been not quantized to zero, for the left and right channel respectively.
l_igf_pN[ ], r_igf_pN[ ]	vector containing information on energies per scalefactor which will be copied from the source tile range, for the left and right channel respectively.
l_sb, r_sb	current IGF source subbands, for the left and right channel respectively.

l_val, r_val	left and right spectral values for the current IGF subband.
sfb_per_ms_band	Number of scalefactor bands per M/S band, dependent on igfUseHighRes.
SFB_PER_PRED_BAND	Number of scalefactor bands per complex prediction band, equal to 2.
dpcm_alpha_q_re[ ][ ]	Differentially coded real part of prediction coefficient of group g, scalefactor band sfb.
alpha_q[ ][ ]	real or imaginary parts of prediction coefficients.

5.5.5.4.10.3 Decoding process

The decoding of MS and complex prediction is divided into 3 parts as follows.

Part 1: Decoding of prediction coefficients

Equal to the decoding of the complex prediction coefficients, but only for the real part as in ISO/IEC 23003-3:2012, subclause 7.7.2.3.2.

Part 2: Inverse quantization of prediction coefficients

Equal to the inverse quantization of the complex prediction coefficients as in ISO/IEC 23003-3:2012, subclause 7.7.2.3.3.

Part 3: IGF apply joint stereo process

If IGF is used in a joint stereo manner (**igfIndependentTiling** is false) the following calling sequence shall be applied:

```
IGF_stereo(num_window_groups, num_windows, group_len) {
  if (!isShortWindow && igfUseWhitening) {
    igf_apply_whitening (l_spec, l_pMDCT_flat);
    igf_apply_whitening (r_spec, r_pMDCT_flat);
  }

  wg = wa = 0;
  for (g = 0; g < num_window_groups; g++) {
    l_igf_sN = {0};
    r_igf_sN = {0};
    l_igf_pN = {0};
    r_igf_pN = {0};
    for (w = 0; w < group_len[g]; w++) {
         IGF_calc_stereo(wg, group_len[g], l_nfSeed1, r_nfSeed1);
       Wg++;
    }
    for (w = 0; w y group_len[g]; w++) {
       IGF_apply_stereo(wa, l_nfSeed2, r_nfSeed2);
       wa++;
    }
  }
}
```

The initial value of the pseudo-random noise seeds l_nfSeed1 and l_nfSeed2 as well as the initial values of r_nfSeed1 and r_nfSeed2 shall be equal in order to synchronize the pseudo-random noise generator

between IGF_calc_stereo() and IGF_apply_stereo() for the first channel and for the second channel respectively.

IGF_calc_stereo() is used to compute the value sets igf_sN and igf_pN respectively. The subroutine get_IGF_sb() is specified in subclause 5.5.5.4.6. Please note that igfCurrTileIdx[0] is a vector of length 4, describing the current tile indices of channel 0, and igfCurrTileIdx[1] is a vector of length 4 too, describing the current tile indices of channel 1 of the actual CPE element.

```
IGF_calc_stereo(w, group_len, l_nfSeed1, r_nfSeed1) {
  igfInc = igfP;
  if (isShortBlock) igfInc = 1;
  for (sfb = m_igfStartSfb; sfb < m_igfStopSfb; sfb+=igfInc) {
    width = (swb_offset[MIN(sfb + igfInc, m_igfStopSfb)] - swb_offset[sfb]);

    r_E = 0; l_E = 0;
    for (bin = 0; bin < width; bin++) {
      tb = swb_offset[sfb]+bin;
      l_E   += l_spec[w][tb] * l_spec[w][tb];
      r_E   += r_spec[w][tb] * r_spec[w][tb];
    }
    l_igf_sN[sfb] += l_E/group_len;
    r_igf_sN[sfb] += r_E/group_len;

    r_E = 0; l_E = 0;
    for (bin = 0 ; bin < width ; bin++) {
      tb = swb_offset[sfb]+bin;
      l_sb = get_IGF_sb(igfCurrTileIdx[0], tb);
      r_sb = get_IGF_sb(igfCurrTileIdx[1], tb);
      l_val = l_spec[w][l_sb];
      r_val = r_spec[w][r_sb];
      if (!isShortWindow && igfUseWhitening) {
        l_tileIdx = get_IGF_tile_idx(tb);
        if (l_igf_WhiteningLevel[l_tileIdx] == 0) {
          l_val = l_pMDCT_flat[w][l_sb];
        }
        if (l_igf_WhiteningLevel[l_tileIdx] == 2) {
          l_val = randomSign(l_nfSeed1)*pow(2,21);
        }
        r_tileIdx = get_IGF_tile_idx(tb);
        if (r_igf_WhiteningLevel[r_tileIdx] == 0) {
          r_val = r_pMDCT_flat[w][r_sb];
        }
        if (r_igf_WhiteningLevel[r_tileIdx] == 2) {
          r_val = randomSign(r_nfSeed1)*pow(2,21);
        }
      }
      if (ms_used[g][sfb] || cplx_pred_used[g][sfb]) {
        tmp = l_val;
        l_val = 0.5 * (tmp + r_val);
        r_val = 0.5 * (tmp - r_val);
      }
      if (l_spec[w][tb] == 0) {
        l_E += l_val * l_val;
      }
      if (r_spec[w][tb] == 0) {
        r_E += r_val * r_val;
      }
    }
    l_igf_pN[sfb] += l_E/group_len;
    r_igf_pN[sfb] += r_E/group_len;
  }
}
```

IGF_apply_stereo will fill spectral gaps with previous calculated values:

```
IGF_apply_stereo(w, l_nfSeed2, r_nfSeed2) {
  igfInc = igfP;
  if (isShortBlock) igfInc = 1;
  for (sfb = m_igfStartSfb; sfb < m_igfStopSfb; sfb += igfInc) {
    width = (swb_offset[MIN(sfb + igfInc, m_igfStopSfb)] - swb_offset[sfb]);

    l_dE = l_igf_curr[sfb];
    r_dE = r_igf_curr[sfb];
    l_sN = l_igf_sN[sfb];
    r_sN = r_igf_sN[sfb];
    l_pN = l_igf_pN[ch][sfb];
    r_pN = r_igf_pN[ch][sfb];
    l_mN = (l_dE*l_dE) * width - l_sN;
    r_mN = (r_dE*r_dE) * width - r_sN;

    if (l_mN > 0 && l_pN > 0) {
      l_gn = min(10, sqrt(l_mN/l_pN));
    } else {
      l_gn = 0;
    }
    if (r_mN > 0 && r_pN > 0) {
      r_gn = min(10, sqrt(r_mN/r_pN));
    } else {
      r_gn = 0;
    }
    for (bin = 0; bin < width; bin++) {
      tb = swb_offset[sfb]+bin;
      l_sb= get_IGF_sb(igfCurrTileIdx[0], tb);
      r_sb= get_IGF_sb(igfCurrTileIdx[1], tb);
      l_val = l_spec[w][l_sb];
      r_val = r_spec[w][r_sb];
      if (!isShortWindow && igfUseWhitening) {
        l_tileIdx = get_IGF_tile_idx(tb);
        if (l_igf_WhiteningLevel[l_tileIdx] == 0) {
          l_val = l_pMDCT_flat[w][l_sb];
        }
        if (l_igf_WhiteningLevel[l_tileIdx] == 2) {
          l_val = randomSign(l_nfSeed2)*pow(2,21);
        }
        r_tileIdx = get_IGF_tile_idx(tb);
        if (r_igf_WhiteningLevel[r_tileIdx] == 0) {
          r_val = r_pMDCT_flat[w][r_sb];
        }
        if (r_igf_WhiteningLevel[r_tileIdx] == 2) {
          r_val = randomSign(r_nfSeed2)*pow(2,21);
        }
      }
      if (ms_used[g][sfb] || cplx_pred_used[g][sfb]) {
        tmp = l_val;
        l_val = 0.5 * (tmp + r_val);
        r_val = 0.5 * (tmp - r_val);
      }
      if (l_spec[w][tb] == 0) {
        l_spec[w][tb] = l_gn * l_val;
      }
      if (r_spec[w][tb] == 0) {
        r_spec[w][tb] = r_gn * r_val;
      }
    }
  }
}
```

After the filling process, the upmix is now applied from subband swb_offset[m_igfStartSfb] to subband swb_offset[m_igfStopSfb] as specified in ISO/IEC 23003-3:2012, 7.7 replacing pred_dir by **igf_pred_dir** if prediction is active. Note that since only real prediction is used, no MDST has to be generated for these bands.

5.5.5.4.11 Temporal noise flattening in IGF

If **igfApplyTNF** is 1, the reconstructed signal by IGF is temporally flattened in the frequency domain. The temporal noise flattening (TNF) is performed in a frequency-selective manner. The selection of the spectral contents to be temporally flattened is achieved by comparing the quantized MDCT coefficients with 0. The contents whose coefficients are quantized to 0 are selected.

In order to maintain the significant MDCT contents, they are temporarily replaced by the MDCT coefficients which are similarly generated to the filled coefficients by IGF:

$$pMDCT_{temp}[w][tb] = \begin{cases} pMDCT[w][tb] & ,if\ pMDCT_{before}[w][tb] = 0 \\ gn \cdot val & ,if\ abs(pMDCT_{before}[w][tb]) > 0' \end{cases}$$

where $pMDCT_{before}[w][tb]$ are the quantized MDCT coefficients, $pMDCT[w][tb]$ are the MDCT coefficients after applying IGF, and gn and val are obtained by the same manner as defined in IGF_apply_mono() and IGF_apply_stereo(), see subclause 5.5.5.4.8.

Afterwards the linear prediction of the MDCT coefficients $pMDCT_{temp}[w][tb]$ is performed in order to obtain the temporally flattened MDCT coefficients $pMDCT_{tnf}[w][tb]$ according to the following filtering:

$$pMDCT_{tnf}[w][tb] = pMDCT_{temp}[w][tb] + \sum_{m=1}^{M} a_{igfTnf}(m) \cdot pMDCT_{temp}[w][tb-m]$$

where M is equal to 8. The required linear prediction coefficients $a_{igfTnf}(m)$ are derived as described in the following three steps:

— First, for each TNF subdivisions $sDiv$ the normalization factors $tnf_{norm}(sDiv)$ are calculated:

$$tnf_{Norm}(sDiv) = \sum_{tb=tnf_{Bgn}(sDiv)}^{tnf_{End}(sDiv)} pMDCT_{temp}[w][tb]^2, sDiv = 0, \dots, 2$$

where the TNF start and stop bands for each subdivision are defined as:

$$tnf_{Bgn}(sDiv) = \left\lfloor igf_{Bgn} + \left(igf_{End} - igf_{Bgn}\right) \cdot \frac{sDiv}{3} \right\rfloor, sDiv = 0, \dots, 2$$

$$tnf_{End}(sDiv) = \left\lfloor igf_{Bgn} + \left(igf_{End} - igf_{Bgn}\right) \cdot \frac{sDiv+1}{3} \right\rfloor, sDiv = 0, \dots, 2$$

— Second, the windowed normalized autocorrelation r_{xx} of the MDCT spectrum in the TNF range is computed if the normalization factor $tnf_{Norm}(sDiv)$ exceeds the given threshold. The following pseudo code describes this process.

```
tnfAcfWindow[8] = {
  0.997803, 0.991211, 0.980225, 0.964844,
  0.945068, 0.920898, 0.892334, 0.859375}

threshold = 0.00000000372529029846191406250;
```

```
for (sDiv = 0; (sDiv < 3) && (tnfNorm[sDiv] > threshold); sDiv ++) {
  rxx[0] = 3;
  for (lag = 1; lag <= 8; lag++) {
    rxx[lag] = 0;
  }
  for (lag = 1; lag <= 8; lag++) {
    acc = 0;
    for (tb = tnfBgn(sDiv); tb < tnfEnd(sDiv) - lag; tb++) {
      acc += pMDCTtemp[tb] * pMDCTtemp[tb + lag];
    }
    rxx[lag] += 1 / tnfNorm(sDiv) * tnfAcfWindow[lag - 1] * acc;
  }
}
```

— Third, the windowed normalized autocorrelation values, r_{xx}, are used in order to finally calculate the linear prediction coefficients (LPC) $a_{igfTnf}(m)$ by solving the set of equations given by:

$$\sum_{m=1}^{M} a_{igfTnf}(m) \cdot r_{xx}(|i-m|) = -r_{xx}(i), i = 1, \ldots, M$$

This set of equations is solved using the Levinson-Durbin recursion given by the following pseudo code.

```
pMemory = &memory[8]
for (i = 0; i < 8; i++) {
  memory[i]  = rxx[i];
  pMemory[i] = rxx[i+1];
}

for (i = 0; i < 8; i++) {
  tmp = 0;
  if (memory[0] >= 1 / 65536) {
    tmp = -pMemory[i] / memory[0];
  }

  tmp = MIN(0.999, MAX(-0.999, tmp));
  parCoeff[i] = tmp;

  for (j = i; j < 8; j++) {
    k               = pMemory[j] + tmp * memory[j - i];
    memory[j - i] += tmp * pMemory[j];
    pMemory[j]      = k;
  }
}

a[0] = 1.0f;
a[1] = parCoeff[0];

for (i = 1; i < 8; i++) {
  for (j = 0; j < (i >> 1); j++) {
    tmp                 = a[j + 1];
    a[j + 1]          += parCoeff[i] * a[i - 1 - j + 1];
    a[i - 1 - j + 1] += parCoeff[i] * tmp;
  }
  if (i & 1) {
    a[j + 1] += parCoeff[i] * a[j + 1];
  }
  a[i + 1] = parCoeff[i];
}
```

The final solution for the LP coefficients is then given as:

$$a_{igfTnf}(m) = a(m), m = 0, \ldots, M-1$$

5.5.5.4.12 IGF core rescaling

If IGF is used in TCX mode, i.e. **fullbandLpd** is equal to 1 and the current frame is LPD/TCX coded**,** the decoded IGF levels are used for rescaling the MDCT coefficients in the IGF range. This process mimics the quantization noise adjustment described in 4.6.2 in ISO/IEC 14496-3:2009 for the TCX core coder.

After decoding the IGF levels as described in subclause 5.5.5.4.4, for each IGF scalefactor band the corresponding IGF level stored in igf_curr[ ] is multiplied on the MDCT coefficients of the TCX spectrum of the current frame:

```
for (tile = 0; tile < nT; tile++) {
  for (sfb = m_igfStartSfb; sfb < m_igfStopSfb; sfb++)
    width = (swb_offset[MIN(sfb + 1, m_igfStopSfb)] - swb_offset[sfb]);
    for (bin = 0; bin < width; bin++) {
      tb = swb_offset[sfb]+bin;
      pMDCT[tb] *= igf_curr[sfb];
  }
}
```

As TCX does not support grouping of windows, i.e. num_windows = 1, the window index w is always 0.

5.5.6 Audio pre-roll

5.5.6.1 General

The *AudioPreRoll()* syntax element is used to transmit audio information of previous frames along with the data of the present frame. The additional audio data can be used to initialize the decoder processing pipeline (pre-roll), thus enabling random access at stream access points (SAP) that make use of *AudioPreRoll()*, seamless transition between different codec configurations including bitrate adaptation.

An *mpegh3daExtElement()* with the usacExtElementType of ID_EXT_ELE_AUDIOPREROLL shall be used to transmit the *AudioPreRoll()*.

5.5.6.2 Semantics

configLen Size of the configuration syntax element in bytes.

Config() The decoder configuration syntax element. In the context of this standard this shall be the Mpegh3daConfig() as defined in subclause 5.2.2.1. The Config() field may be transmitted to be able to respond to changes in the audio configuration (e.g. switching of streams).

numPreRollFrames The number of pre-roll access units (AUs) transmitted as audio pre-roll data. Typically, a value of 1 is signalled for initializing inverse transform filterbanks of the core decoder and the renderers.

auLen AU length in bytes.

AccessUnit() The pre-roll AU(s).

applyCrossfade If this flag is set to 1, a linear crossfade shall be applied in case of configuration change, as defined in subclause 5.5.6.3.3.

apr_reserved reserved bit shall be zero.

NOTE The pre-roll data carried in the extension element can be transmitted "out of band", i.e. the buffer requirements might not be satisfied.

In order to use AudioPreRoll() for both random access and seamless configuration changes the following restrictions apply.

— The first element of every frame shall be an extension element (mpegh3daExtElement) of type ID_EXT_ELE_AUDIOPREROLL.

— The corresponding mpegh3daExtElement() shall be configured as specified in Table 85.

— Consequently, if pre-roll data is present, this mpegh3daFrame() shall start with the following bit sequence:

  — "1": usacIndependencyFlag;

  — "1": usacExtElementPresent (referring to audio pre-roll extension element);

  — "0": usacExtElementUseDefaultLength (referring to audio pre-roll extension element).

— If no *AudioPreRoll()* is transmitted, the extension payload shall not be present (usacExtElementPresent = 0).

— The pre-roll frames with index "0" shall be independently decodable, i.e. *usacIndependencyFlag* shall be set to "1".

— To enable seamless configuration changes and bitrate adaptations the involved bitstreams shall have the element **receiverDelayCompensation** set to one.

Table 85 — Setup of mpegh3daExtElementConfig() for AudioPreRoll()

Bitstream Field	Value
usacExtElementType	ID_EXT_ELE_AUDIOPREROLL
usacExtElementConfigLength	0
usacExtElementDefaultLengthPresent	0
usacExtElementPayloadFrag	0

5.5.6.3 Decoding process

5.5.6.3.1 General

This subclause describes the decoding process for both random access/immediate play-out and configuration changes including bitrate adoption scenarios.

5.5.6.3.2 Random access and immediate play-out

Random access and immediate play-out is possible at every frame that utilizes the *AudioPreRoll*() structure as specified in this subclause. The following pseudo-code describes the decoding process.

```
if(usacIndependencyFlag == 1){
  if(usacExtElementPresent == 1){

    /* In this case usacExtElementUseDefaultLength shall be 0! */
    if(usacExtElementUseDefaultLength != 0) goto error;

    /* parse over length information and discard */
    getmpegh3daExtElementPayloadLength();
```



```
    /* Check for presence of config and re-initialize if necessary */
    int configLen = getConfigLen();
    if(configLen > 0){
      config c = getConfig(configLen);
      ReConfigureDecoder(c);
    }

    /* Get pre-roll AUs and decode, discard output samples */
    int numPreRollFrames = getNumPreRollFrames();
    for(auIdx = 0; auIdx < numPreRollFrames; auIdx++)
      int auLen = getAuLen();
      AU nextAU = getPreRollAU(auLen);
      DecodeAU(nextAU);
      /* outSamplesFrame are discarded */
    }
  }
}
/* Internal decoder states are initialized at this point. Continue normal
decoding */
```

5.5.6.3.3 Configuration change and bitrate adaption

If **receiverDelayCompensation==1** the *AudioPreRoll()* structure as specified in this subclause can be used to enable seamless configuration changes and bitrate adaptions. The decoding process is specified below.

If a configuration change is detected by the decoder the following steps shall be applied. Note that a configuration (and thus a configuration change) may be signalled to the decoder either by a new mpegh3daConfig() element (e.g. in an MHAS packet of type PACTYP_MPEGH3DACFG) or within the *AudioPreRoll()* structure.

— Flush the internal decoder states and buffers for the core decoder and rendering path (`FlushDecoder()`) by decoding hypothetical access units composed of all zero samples. Store the resulting output samples (`outSamplesFlush`) in a temporary buffer. The number of flushed samples (`numSamplesFlushed`) shall equal the core decoder and rendering path delay.

— Re-initialize the decoder with the new configuration.

— Decode and render all contained pre-roll AUs and discard the resulting rendered samples at the mixer.

— Decode and render the current AU and following AUs and store the resulting output samples (`outSamplesFrame`). Discard the first `numSamplesFlushed` samples of `outSamplesFrame`. After discarding, the first remaining sample in `outSamplesFrame` is the first valid decoded and rendered sample.

— During startup of the reinitialized decoder feed samples from the temporary buffer (`outSamplesFlush`) into the post-processor. Note that due to the constant decoder delay enforced by **receiverDelayCompensation==1** the temporary buffer will cover exactly the startup phase of length `numSamplesFlushed` until valid samples from the reinitialized decoder will arrive at the mixer.

— After the startup of the reinitialized decoder feed samples from the reinitialized decoder (`outSamplesFrame`) into the post-processor.

— In case **applyCrossfade** is set to 1 and the mixer operates in the time domain, an additional buffer of 128 samples (`crossfadeFlush`) shall be **flushed** in the `FlushDecoder()` call. The

`crossfadeFlush` buffer shall be used for a 128 sample linear crossfade between `crossfadeFlush` and the first 128 valid samples of the reinitialized decoder arriving at the mixer:

```
/* Apply crossfade */
if(applyCrossfade) {
  for(i = 0; i < 128; i++){
    outSamples[i] = crossfadeFlush [i] * (1-i/127) +
                    outSamplesFrame[i] * (i/127)
  }
}
```

5.5.7 Fullband LPD

5.5.7.1 Tool description

In the fullband LPD mode, the ACELP core mode (see ISO/IEC 23003-3:2012, 7.14) is operated at half the sampling frequency of the MDCT based TCX core mode (see ISO/IEC 23003-3:2012, 7.15), i.e.

$$f_{s,TCX} = 2 \cdot f_{s,ACELP}$$

To adjust the output signals at different sampling frequencies, the ACELP output is upsampled applying a time-domain resampler. The missing bandwidth ranging from $f_{s,ACELP}/2$ to $f_{s,TCX}/2$ in case of ACELP processing in comparison to TCX in combination with enhanced noise filling (see subclause 5.5.7.8.5) is addressed by the time-domain bandwidth extension (see subclause 5.5.8). This subclause describes the decoding of fullband LPD and related changes in decoding with existing tools.

5.5.7.2 Data elements

fullbandLpd this flag signals the usage of the fullband LPD tool.

window_shape window shape of the current subframe.

5.5.7.3 Helper elements

$f_{s,TCX}$ sampling rate of the fullband TCX.

$f_{s,ACELP}$ sampling rate of the ACELP.

lg number of spectral coefficients of the current TCX subframe.

rl[ ] TCX noise generation flag array, contains only the values 1 and 0.

rr[ ] reconstructed spectrum.

x[ ] output of the IMDCT.

nfBgn subband index; this index indicates the start of the TCX noise generation.

nfEnd subband index; this index indicates the stop of the TCX noise generation.

g re-scaling gain factor.

$z_{TCX,FB}$ time-domain output of TCX sampled at $f_{s,TCX}$.

$z_{TCX,LB}$ time-domain output of TCX sampled at $f_{s,ACELP}$.

z_{ACELP} time-domain output of ACELP sampled at $f_{s,ACELP}$.

z_{OUT} mixed time-domain output sampled at $f_{s,TCX}$.

z_fb[ ] decoded windowed time domain signal of the fullband TCX @ $f_{s,TCX}$.

z_lb[ ] decoded windowed time domain signal of the fullband TCX @ $f_{s,ACELP}$.

5.5.7.4 Framing

In case of fullband LPD coding, i.e. **fullbandLpd** is equal to 1, long TCX frames correspond to coreCoderFrameLength samples @ $f_{s,TCX}$ and medium TCX frames correspond to coreCoderFrameLength/2 samples @ $f_{s,TCX}$. For ACELP, the standard frame size of coreCoderFrameLength/4 samples @ $f_{s,ACELP}$ as described in ISO/IEC 23003-3:2012, 7.14 is used, while coreCoderFrameLength=1024 is fixed. Possible frame combinations within one superframe are shown in Figure 17.

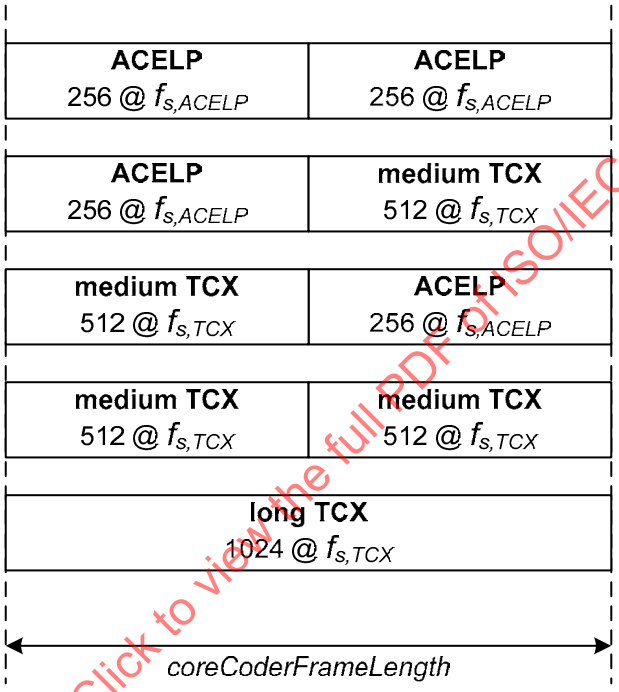


Figure 17 — Possible frame combinations in fullband LPD within one superframe

5.5.7.4.1 Mode coding

In case of fullband LPD, i.e. **fullbandLpd** is equal to 1, the lpd mode coding is the same as described in ISO/IEC 23003-3:2012, 6.2.10 while one "superframe" consists of only two frames instead of four. Due to this, the definitions in ISO/IEC 23003-3:2012, Table 89 are replaced with those in Table 86.

Table 86 — Mapping of coding modes for lpd_channel_stream() in case of fullband LPD

	meaning of bits in bit-field lpd_mode			**remaining mod[] entries**
lpd_mode	**bit 2**	**bit 1**	**bit 0**	
0..3	0	mod[1]	mod[0]	
4	1	0	0	mod[1]=2 mod[0]=2

In case of fullband LPD, as there are only two frames, coding mode 3 is not applicable, thus the definitions in ISO/IEC 23003-3:2012, Table 92 are replaced with those in Table 87.

Table 87 — Coding modes indicated by mod[ ] in case of fullband LPD

Value of mod[x]	Coding mode in frame	Bitstream element
0	ACELP	acelp_coding()
1	medium TCX (ccfl/2)	tcx_coding()
2	long TCX (ccfl)	tcx_coding()

In case of fullband LPD, as there are only two frames, coding mode 3 is not applicable, thus the definitions in ISO/IEC 23003-3:2012, Table 148 are replace with those in Table 88.

Table 88 — number of spectral coefficients as a function of mod[] and coreCoderFrameLength (ccfl)

Value of mod[x]	Number lg of spectral coefficients	ZL	L	M	R	ZR
1	ccfl/2	0	ccfl/2	0	ccfl/2	0
2	ccfl	ccfl/4	ccfl/2	ccfl/2	ccfl/2	ccfl/4

5.5.7.5 TD resampler

The time-domain (TD) resampler is used to upsample the time-domain output of ACELP (see ISO/IEC 23003-3:2012, 7.14) and the forward aliasing cancellation tool (see ISO/IEC 23003-3:2012, 7.16) by the factor of 2, in order to bring the output to the same sampling rate as the fullband MDCT based TCX output.

Every 2 samples of the time-domain input signal are separated by a zero-valued sample to form a sequence with a sampling frequency ($f_{s,out}$) which is increased by the factor of 2 compared to the sampling frequency of the input signal ($f_{s,in} = f_{s,out}/2$). Subsequently, the upsampled sequence s_{up} is filtered using an FIR interpolation filter of length N given by the symmetric filter coefficients b_k in Table 89 to form the time-domain output signal s_{out} for each sample of index i by:

$$s_{out}(i) = \sum_{k=0}^{N-1} b_k \cdot s_{up}(i-k)$$

The resampler introduces a delay of 30 samples at $f_{s,out}$ to the upsampled signal s_{out}.

Table 89 — Interpolation filter coefficients b_k

k	b_k
30	1
29, 31	0,634 876 04
28, 32	0

k	b_k
27, 33	-0,207 013 53
26, 34	0
25, 35	0,118 794 87
24, 36	0
23, 37	-0,079 265 75
22, 38	0
21, 39	0,056 156 42
20, 40	0
19, 41	-0,04 070 711
18, 42	0
17, 43	0,029 576 17
16, 44	0
15, 45	–0,021 220 66
14, 46	0
13, 47	0,014 831 15
12, 48	0
11, 49	0,009 939 03
10, 50	0
9, 51	0,006 248 19
8, 52	0
7, 53	-0,003 554 76
6, 54	0
5, 55	0,001 705 82
4, 56	0
3, 57	-0,000 577 01
2, 58	0
1, 59	0,000 060 13
0, 60	0

5.5.7.6 LPC filter

Yet, as there are only 2 ACELP frames per superframe in case of fullbandLPD, the maximum number of LPC-filters to be present within 1 superframe changes from 5 to 3. The LPC-filters LPC2 and LPC3 are not used, while the remaining LPC-filters correspond to LPC0, LPC1 and LPC4. Thus, in case of fullbandLPD, the conditions of presence of LPC-filters are defined according to Table 90, and possible quantization modes are defined according to Table 91.

Table 90 — Conditions for the presence of a given LPC filter in the bitstream in fullbandLPD

LPC filter	Present if
LPC0	**first_lpd_flag**=1
LPC1	mod[0]<2
LPC4	always

Table 91 — Allowed absolute and relative quantization modes, corresponding bitstream signaling of mode_lpc and coding modes for codebook numbers nk

Pos. in Bitstr.	Present if	Filter	Quantization mode	mode_lpc	Binary Code	n_k mode
1.	always	LPC4	Absolute	0	(none)	0
2.	first_lpd_flag=1	LPC0	Absolute	0	0	0
			Relative to LPC4	1	1	3
3.	mod[0]<2	LPC1	Absolute	0	10	0
			Relative to (LPC0+LPC4)/2 [a]	1	11	1
			Relative to LPC4	2	0	3

[a] In this mode, there is no second-stage AVQ quantizer.

5.5.7.7 ACELP decoding

In case of fullband LPD, i.e. **fullbandLpd** is equal to 1, ACELP is decoded and processed as described in ISO/IEC 23003-3:2012, 7.14. Subsequent to the ACELP synthesis and prior to writing it to the output buffer, the ACELP output is upsampled by the factor of 2 using the time-domain resampler described in subclause 5.5.7.5. The TD resampler is initialized by filtering previous samples to enable delayless upsamling.

5.5.7.8 TCX decoding

5.5.7.8.1 TCX frequency band offset tables

For the noise filling, intelligent gap filling, and temporal noise shaping algorithms applied in the MDCT based TCX, a vector swb_offset_tcx[] of frequency band offsets (i. e. indices of spectral bins representing frequency band borders) is required. Identically to 8-short-window FD channels and frames coded using window_sequence == EIGHT_SHORT_SEQUENCE, these TCX band offsets are based on the sampling-rate dependent swb_offset_short_window[] values defined in ISO/IEC 14496-3:2009, Tables 4.130 to 4.141. In the context of TCX decoding the frequency band offset tables shall be extended such that swb_offset_short[num_swb_short_window] equals half the window length of the EIGHT_SHORT_SEQUENCE, e.g. 128.

The swb_offset_tcx[] vector for a given TCX window is determined using value lg from Table 88:

swb_offset_tcx[i] = swb_offset_short_window[i] · f(lg), i = 0, 1, 2, …, num_swb_short_window,

where f(lg) = lg / 128, with lg being based on vector mod[ ] defined according to Table 53 in 5.2.3.2.

5.5.7.8.2 TCX noise generation

The noise filling in MDCT based TCX is applied as described in ISO/IEC 23003-3:2012, subclause 7.15.3. However, the noise filling start index, lg/6 and stop index, lg are modified as follows if **fullbandLpd** == 1.

A run of 8 non-zeros is detected according to the following pseudo code, where nfBgn and nfEnd depend on **fullbandLpd** and **enhancedNoiseFilling** and, in case of the latter, the swb_offset_tcx bin index array:

```
if (fullbandLpd) {
  nfBgn = (lg/6) & 2040;
  nfEnd = enhancedNoiseFilling ? igfBgn : lg;
  nfEnd = min(nfEnd, min(lg, swb_offset_tcx[max_sfb]));
} else {
  nfBgn = (lg/6);
  nfEnd = lg;
}

for (i = 0; i < nfBgn; i++) {
  rl[i] = 1;
}

for (i = nfBgn; i < nfEnd; i += 8) {
  int k, maxK = min(nfEnd, i+8);
  tmp = 0;
  for (k = i; k < maxK; k++) {
    tmp += x_tcx_invquant[k] * x_tcx_invquant[k];
  }

  if (tmp != 0) {
    for (k = i; k < maxK; k++) {
      rl[k] = 1;
    }
  } else {
    for (k = i; k < maxK; k++) {
      rl[k] = 0;
    }
  }
}
```

5.5.7.8.3 Adaptive low-frequency de-emphasis

The purpose of the adaptive low-frequency emphasis and de-emphasis (ALFE) processes is to improve the subjective performance of the frequency-domain TCX codec at low frequencies. To this end, the TCX low-frequency MDCT spectral lines are amplified prior to quantization in the encoder, thereby increasing their quantization SNR, and this boosting is reversed prior to the inverse MDCT operation in the decoder.

The ALFE operates on the spectral lines in vector x[] directly after the above-noted inverse quantization, noise filling and if enabled, frequency-domain prediction (i. e. x[] represents either the x_tcx_invquant[] or outputSpecCurr[]). If both **fullbandLpd** and **enhancedNoiseFilling** are zero the conventional ALFE algorithm, described as a four-step “de-shaping” procedure in ISO/IEC 23003-3:2012, 7.15, is applied.

Otherwise, the ALFE operates based on the LPC frequency-band gains, lpcGains[], which are derived from the gains g1 and g2 used for FD noise shaping, as defined in ISO/IEC 23003-3:2012, 7.15, by:

lpcGains[k] = sqrt(g1[k] * g2[k]), i. e. the geometric mean of g1 and g2 at each index k.

The ALFE decoding is achieved as follows. First, the minimum and maximum of the first nine gains – the low-frequency gains – are found using comparison operations executed within a loop over the gain indices 0 to 8., i. e. over lpcGains[*i*] with *i* = 0, 1,.., 8. Then, if the ratio between the minimum and maximum gain values exceeds a threshold of 1/32, a gradual lowering of the lowest lines in x is performed such that the line at index 0 is attenuated by $(\max/(32 \cdot \min))^{0.25}$ and the line at index alfeLength is not attenuated:

```
alfeLength = lg / 8;
tmp = 32 * min;
if ((max < tmp) && (tmp > 0)) {
  fac = tmp = pow(max / tmp, 1/(4 * alfeLength));
  /* gradual lowering of lowest alfeLength lines: */
  for (i = alfeLength-1; i >= 0; i--) {
    x[i] *= fac;
    fac  *= tmp;
  }
}
```

5.5.7.8.4 MDCT domain rescaling in TCX

In MDCT based TCX coding, the reconstructed spectrum rr[] is fed into an inverse MDCT. The non-windowed output signal, x[], is re-scaled by the gain, g, obtained by an inverse quantization of the decoded global_gain index as described in ISO/IEC 23003-3:2012, 7.15.3. If **fullbandLpd** is equal to 1, the rescaling of the spectrum is performed in the MDCT domain. The stop index of the core coder rescaling depends on **enhancedNoiseFilling**:

```
rsEnd = enhancedNoiseFilling ? min(lg, igfBgn) : lg;
rr[i] = rr[i]·g; i = 0 .. rsEnd – 1
```

5.5.7.8.5 Enhanced noise filling in TCX

In MDCT based TCX coding, enhanced noise filling is carried out by the intelligent gap filling (IGF) tool as described in subclause 5.5.5 if **enhancedNoiseFilling** is equal to 1.

The IGF decoding process for each TCX spectrum is performed after the arithmetic decoding noise filling and rescaling, but before the de-shaping is applied to the spectrum.

5.5.7.8.6 De-shaping

The spectrum de-shaping is applied to the reconstructed spectrum according to ISO/IEC 23003-3:2012, 7.15.3. If **fullbandLpd** is equal to 1, the inverse FDNS operation consists in filtering the reconstructed spectrum r[i] using the recursive filter:

```
rr[i] = a[i]·r[i]+b[i]·rr[i-1], i = 0 ... lg/2-1,
```

where a[i] and b[i] are derived from the left and right gains g1[k], g2[k] using the formulae:

```
a[i] = 2·g1[k]·g2[k]/(g1[k]+g2[k]),
b[i] = (g2[k]-g1[k])/(g1[k]+g2[k])
```

In the above, the variable k is equal to i/(lg/M) to take into consideration the fact that the LPC spectrums are decimated, where M=coreCoderFrameLenght/16.

The spectral coefficients between lg/2 and lg are filtered by holding the last calculated filter coefficients:

```
rr[i] = a[i]·r[i]+b[i]·rr[i-1], i = lg/2 ... lg-1,
```

where a[i] and b[i] are derived from the left and right gains g1[M-1], g2[M-1] using the formulae:

```
a[i] = 2·g1[M-1]·g2[M-1]/(g1[M-1]+g2[M-1]),
b[i] = (g2[M-1]-g1[M-1])/(g1[M-1]+g2[M-1])
```

5.5.7.8.7 Temporal noise shaping in TCX

In MDCT based TCX coding, temporal noise shaping (TNS) is applied as in short-block FD channels (i.e. channels and frames with **window_sequence**==EIGHT_SHORT_SEQUENCE) and follows the bitstream syntax and description specified in ISO/IEC 23003-3:2012, 5.3.2 and 7.8. If **tns_data_present** ≠ 0 for a given lpd_channel_stream(), a single-window instance of tns_data() is read for each TCX spectrum.

The TNS decoding (i.e. synthesis filtering) process for each TCX spectrum is performed after the arithmetic decoding, noise filling, frequency-domain prediction and enhanced noise filling steps, as referenced in ISO/IEC 23003-3:2012, 7.8 and, in case of intelligent gap filling with TTS, in subclause 5.5.5.4.5. The only difference is that the swb_offset_short_window[] values employed for frequency band restriction of the TNS filtering process are multiplied with a factor *f*(mod[k]), as described in subclause 5.5.7.8.1.

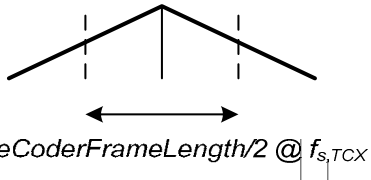
5.5.7.8.8 Inverse MDCT in TCX

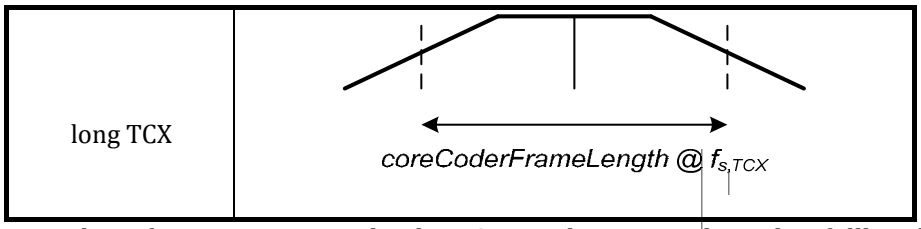
The reconstructed spectrum rr[] is fed into an inverse modified discrete cosine transform (IMDCT) to obtain the non-windowed time domain output signal x[] as described in ISO/IEC 23003-3. If **fullbandLpd** is equal to 1, it is necessary to perform two independent IMDCTs, one for the fullband TCX output z_fb[] and one for the down-sampled lowband TCX output z_lb[] which will be later used for initializing the ACELP core. The down-sampling by the constant factor of 2 is achieved by applying an IMDCT of half the length of the regular IMDCT, i.e. by applying an IMDCT of length lg/2.

5.5.7.8.9 TCX windowing

In case of fullband LPD, i.e. **fullbandLpd** is equal to 1, the windows applied to the TCX frames prior to the transform and after inverse transform are depicted in Table 92.

Table 92 — Window shapes for TCX frames in fullband LPD

Frame	Window shape (schematic)
medium TCX	



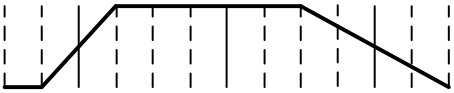
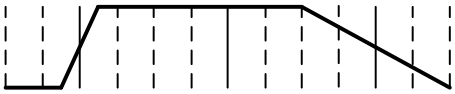
However, when switching from or to FD mode, the TCX windows are adapted in fullband LPD mode, i.e. **fullbandLpd** is equal to 1, while the windows for the FD core remain the same. The TCX transitions window shapes are schematically depicted in Table 93 and Table 94.

Table 93 — Transition window shapes of length N=1024 samples for medium TCX frames in fullband LPD

Transition	Window shape (schematic)
medium TCX → LONG_STOP_SEQUENCE / STOP_START_SEQUENCE	
medium TCX → EIGHT_SHORT_SEQUENCE	
LONG_START_SEQUENCE / STOP_START_SEQUENCE → medium TCX	
EIGHT_SHORT_SEQUENCE → medium TCX	

Table 94 — Transition window shapes of length N=1536 samples for long TCX frames in fullband LPD

Transition	Window shape (schematic)
long TCX → LONG_STOP_SEQUENCE / STOP_START_SEQUENCE	
long TCX → EIGHT_SHORT_SEQUENCE	
LONG_START_SEQUENCE / STOP_START_SEQUENCE →	

long TCX	
EIGHT_SHORT_SEQUENCE → long TCX	

The calculation formulae of the window shapes are described in ISO/IEC 23003-3:2012, 7.9.3.2. In accordance to the previous frame, the left slope of the window shape and according to the presence of **window_shape**, the right slope of the window shape can either be a sine or a KBD window. For **window_shape** == 1, the window coefficients used for the IMDCT are given by the Kaiser–Bessel derived (KBD) window, otherwise the sine window is employed (see ISO/IEC 23003-3:2012, 7.9.3.2).

If the first access unit (AU) is in fullband LPD mode, i.e. **fullbandLpd** is equal to 1, and the first subframe within this AU is TCX coded, then the **window_shape** of the left half of this subframe window shall adopt the **window_shape** of the right half.

5.5.7.9 Forward aliasing cancellation (FAC) tool

In case of fullband LPD, i.e. **fullbandLpd** is equal to 1, the FAC tool is decoded as described in ISO/IEC 23003-3:2012, 7.16. Subsequent to the decoding and prior to writing it to the output buffer, the FAC signal is upsampled by the factor of 2, using the time-domain resampler described in subclause 5.5.7.5.

Therefore the FAC signal fac is extended to fac_ext as described by the following pseudo code in order to allow for delayless upsampling using the TD resampler:

```
for (i = 0; i < 2 * lfac; i++) {
  fac_ext[15 + i] = fac[i];
}
for (i = 0; i < 15; i++ ) {
  fac_ext[15 - 1 - i]         = 2 * fac[0]             - fac[1 + i];
  fac_ext[15 + 2 * lfac + i] = 2 * fac[2 * lfac - 1] - fac[2 * lfac - 2 - i];
}
```

Subsequently the TD resampler is applied to the signal fac_ext as described in subclause 5.5.7.5. After upsampling the first 60 samples of the upsampled signal are discarded.

The different core coder sampling frequencies $f_{s,ACELP}$ & $f_{s,TCX}$ in case of fullband LPD need to be considered when determining the length of the FAC transform (fac_length) for decoding ($f_{s,ACELP}$) and writing to the output buffer ($f_{s,TCX}$):

fac_length = *coreCoderFrameLength* / 32 @ $f_{s,ACELP}$ if transitioning between ACELP and EIGHT_SHORT_SEQUENCES;

fac_length = *coreCoderFrameLength* / 16 @ $f_{s,ACELP}$ if transitioning from ACELP to LONG_STOP_SEQUENCE or from LONG_START_SQUENCE to ACELP;

fac_length = *coreCoderFrameLength* / 8 @ $f_{s,ACELP}$ if transitioning between ACELP and TCX;

5.5.7.10 Post-processing of the synthesis signal

In case of fullband LPD, i.e. **fullbandLpd** is equal to 1, the post-processing of the synthesis signal is performed as described in ISO/IEC 23003-3:2012, 7.17. While the control coefficient for the inter-harmonic attenuation a and the post-filter gain g_{PF} are determined based on the ACELP synthesis signal before resampling (see subclause 5.5.7.7), the filtering is applied to the upsampled ACELP synthesis output. In case of post-processing of transitions between FD mode to and from ACELP, the adapted FAC area lengths shall be considered (see subclause 5.5.7.9). The bass-post filter is always enabled for ACELP frames and FAC areas in case of fullband LPD, constantly changing the value of bpf_control_info in ISO/IEC 23003-3:2012, Table 90 to bpf_control_info=1.

5.5.7.11 Coding mode switching

As described in subclause 5.5.7.8.8, the MDCT based TCX decoder generates 2 time-domain output signals at sampling frequencies $f_{s,ACELP}$ and $f_{s,TCX}$, respectively. The signal sampled at $f_{s,ACELP}$ is used to update the ACELP memories to enable seamless transitions when switching between MDCT based TCX and ACELP.

To avoid discontinuities, crossfading is applied as given by:

$$z_{OUT}[N_{frame}-N_{xfade}+k] = z_{TCX,FB}[N_{frame}-N_{xfade}+k]\cdot\left(\frac{N_{xfade}-k}{N_{xfade}}\right) + z_{TCX,LB}[N_{frame}-N_{xfade}+k]\cdot\left(\frac{k}{N_{xfade}}\right)$$

$$for\ k = 0,\ldots,N_{xfade}-1$$

and depicted schematically in Table 95 for switching from MDCT based TCX to ACELP, and by:

$$z_{OUT}[k] = z_{TCX,FB}[k]\cdot\left(\frac{k}{N_{xfade}}\right) + z_{TCX,LB}[k]\cdot\left(\frac{N_{xfade}-k}{N_{xfade}}\right)$$

$$for\ k = 0,\ldots,N_{xfade}-1$$

and depicted schematically in Table 96 for switching from ACELP to MDCT based TCX, where N_{frame} is the frame length of the TCX frame in samples at $f_{s,TCX}$ and N_{xfade} is the length of the crossfade range in samples at $f_{s,TCX}$, which corresponds to the delay of the TD resampler (30 samples at $f_{s,TCX}$) described in subclause 5.5.7.5.

Table 95 — Schematic depiction of switching from TCX to ACELP

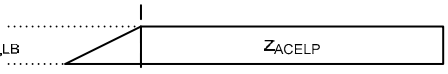
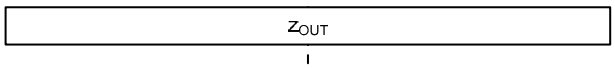
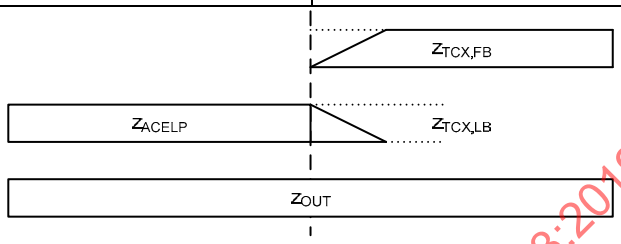
Coding mode	TCX	ACELP
Frame index	n-1	n
Fullband TCX output		
Upsampled lowband TCX and ACELP output		
Mixed fullband output		

Table 96 — Schematic depiction of switching from ACELP to TCX

Coding mode	ACELP	TCX
Frame index	n-1	n
Fullband TCX output		
Upsampled lowband TCX and ACELP output		
Mixed fullband output		

5.5.8 Time-domain bandwidth extension

5.5.8.1 General

This clause describes the decoding process of time-domain bandwidth extension (TBE). The TBE decoder tool is used to enable low bit rate coding of speech via the MPEG-H 3D audio codec's LPD path for ACELP mode.

5.5.8.2 Overview

Figure 18 shows a high level framework of the TBE decoder. The input bitstream is de-multiplexed and decoded by the MPEG-H 3D audio core decoder to produce the ACELP low band (LB) excitation and low band synthesis. The TBE bitstream is parsed and the parameters are passed to the TBE decoding tool. The high band synthesis is performed using the TBE parameters and the harmonically-extended high band excitation signal.

The synthesized high band is then up-sampled and spectrally flipped in the time-domain to generate a high band component associated with the final decoded audio. The low band is also up-sampled to the same sampling rate as the high band, and then mixed with the "delay-adjusted" high band component to generate the output. In particular, the low-band core decoder may exhibit more delay than the high band processing, which would require that the high band is delayed accordingly before mixing with low band such that the low band and high band are time-aligned to avoid any artifacts.

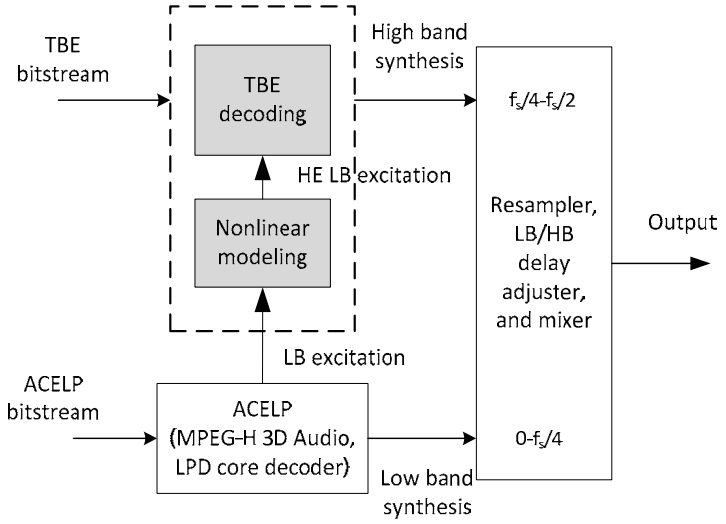


Figure 18 — Simplified MPEG-H 3D audio core decoder with TBE tools

5.5.8.3 Summary of the TBE decoding tools

Figure 19 shows an overview of the TBE decoder tools. The TBE frame converter parses the TBE bitstream configuration data, tbe_data(), as described in Table 56, and passes the parameters to the TBE decoding module. The parameters associated with the TBE configuration data are described in subclause 5.5.8.4. The acelp_coding() configuration data as described in Table 54 is used by the MPEG-H 3D audio LPD core decoder to generate the low band excitation, E_{LB} (E_LB). The nonlinear modelling module is used to generate the harmonically-extended high band excitation signal, E_{HE} (E_HE).

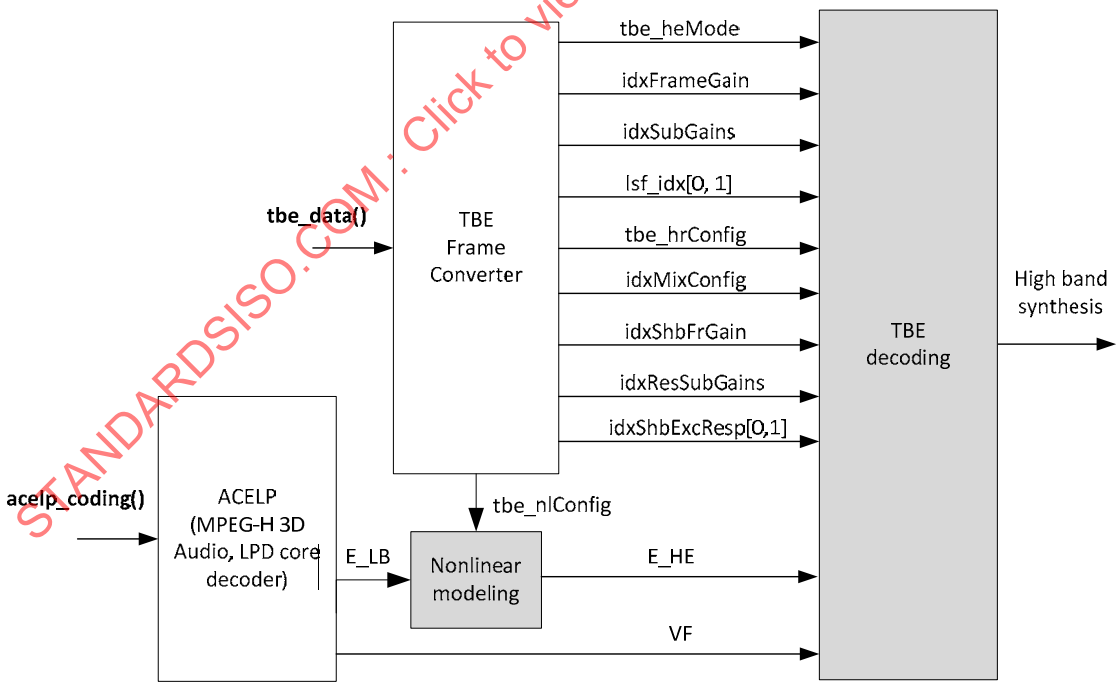


Figure 19 — TBE decoder tools

The TBE decoding process includes the following steps.

- TBE decoding
  - Nonlinear modelling
    - Resampling of LB excitation
    - Harmonically extending the LB excitation based on tbe_nlConfig
  - Dequantization of high band parameters
    - Line spectral frequencies, temporal gains (gain shapes and gain frame), mixing configuration, high band reference gain, residual gain shapes, high band excitation inverse function.
  - High band LP estimation
  - High band excitation generation
    - A: Spectral flip in the time domain and decimation of E_HE
    - B: Adaptive whitening of A
    - C: Temporal envelope-modulated noise generation based on B
    - HB excitation estimation based on B and C
  - High band LP synthesis
  - Temporal envelope adjustment of HB synthesis
  - Spectral flip and upsampling.

5.5.8.4 Definitions of TBE payloads

tbe_data()	This element contains information about the high band audio content.
tbe_heMode	This element determines whether the TBE decoding of current frame uses low bit rate high efficiency mode. If the flag is set to zero, then the high resolution configuration (tbe_hrConfig) is enabled.
idxFrameGain	This payload contains data for the overall frame gain adjustment.
idxSubGains	This payload contains data for the temporal sub-frame gain shape adjustment.
lsf_idx[0]	This payload contains LSF data associated with the first stage LSFs used to estimate the LSP and then subsequently the interpolated sub-frame LP parameters.
lsf_idx[1]	This payload contains LSF data associated with the second stage LSFs used to estimate the LSP and then subsequently the interpolated sub-frame LP parameters.
tbe_hrConfig	This flag signals whether the current frame uses high resolution configuration. The flag is only read from the bitstream if the tbe_heMode is set to zero.
tbe_nlConfig	This flag signals the NL configuration that is to be used to generate the HE LB excitation. The flag is only read from the bitstream if the tbe_heMode is set to zero. The default value of tbe_nlConfig is set to 1, if tbe_heMode is set to 1.

idxMixConfig	This payload contains data to control HB excitation generation based on B and C in subclause 5.5.8.5. The flag is only read from the bitstream if the tbe_heMode is set to zero.
idxShbFrGain	This payload contains data for the overall high band target gain. The flag is only read from the bitstream if the tbe_heMode is set to zero and tbe_hrConfig is set to 1.
idxResSubGains	This payload contains data for temporal sub-frame residual gain shape adjustment. The flag is only read from the bitstream if the tbe_heMode is set to zero and tbe_hrConfig is set to 1.
idxShbExcResp[0]	This payload contains data to filter the HE excitation B in subclause 5.5.8.5. The flag is only read from the bitstream if the tbe_heMode is set to zero and tbe_hrConfig is not set to 1.
idxShbExcResp[1]	This payload contains data to filter the HE_LB excitation B in subclause 5.5.8.5. The flag is only read from the bitstream if the tbe_heMode is set to zero and tbe_hrConfig is not set to 1.

5.5.8.5 TBE decoder processing

5.5.8.5.1 General overview

An overview of TBE decoder processing steps is shown in Figure 20.

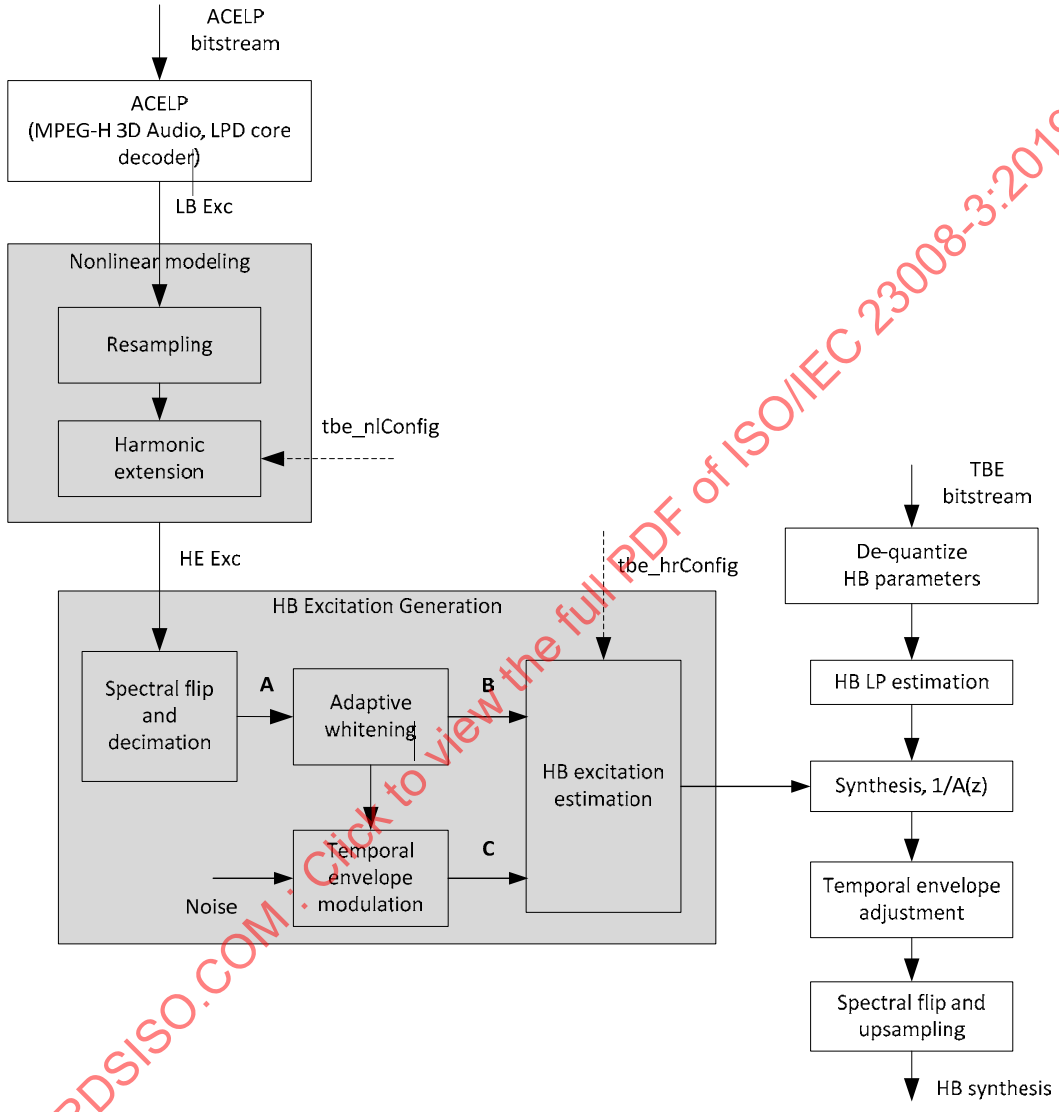


Figure 20 — Overview of the TBE decoder processing

5.5.8.5.2 De-quantization of HB parameters

The codebooks used to de-quantize some of the high band TBE parameters are summarized in Table 97. Pseudo-code that describes the de-quantization process is given below.

Table 97 — Codebook tables used to de-quantize some of the high band TBE parameters

Codebook table	Parameter
SHBCB_GainFrame5bit	Gain frame, Table 0.6, 5 bits
SHBCB_SubGain5bit	Subframe gains, Table 0.5, 5 bits
tbeLSFCB1_7b	LSF first stage, Table 0.1, 7 bits
tbeLSFCB2_7b	LSF second stage, Table 0.2, 7 bits
tbeExcFilterCB1_7b	tbeExcFilter1, Table 0.3, 7 bits
tbeExcFilterCB2_4b	tbeExcFilter2, Table 0.4, 4 bits

```
frameGain = SHBCB_GainFrame5bit[idxFrameGain];

j = 4*idxSubGain;
for (i = 0; i < 4; i++) {
  subGain[i] = SHBCB_SubGain5bit[j++];
}

copyVector(tbeLSFCB1_7b + 10 * lsf_idx[0], qLsf,  10);
copyVector(tbeLSFCB2_7b + 10 * lsf_idx[1], qtemp, 10);
for (i = 0; i < 10; i++) {
  qLsf[i] = qLsf[i] + qtemp[i];
}

if (tbe_heMode==0) {
  mixFac = (idxMixConfig + 1) / 4;
  if (tbe_hrConfig == 1) {
    hbEnerTarget = 10^(0.0625 * idxShbFrGain);
    j = 4 * idxResSubGains;
    for (i = 0; i < 4; i++) {
      resSubGain[i] = SHBCB_SubGain5bit[j++];
    }
  } else {
    copyVector(tbeExcFilterCB1_7b + 10 * idxShbExcResp[0], tbeExcFilter1, 10);
    copyVector(tbeExcFilterCB2_4b +  6 * idxShbExcResp[1], tbeExcFilter2,  6);
  }
}
```

5.5.8.5.3 Nonlinear modelling

5.5.8.5.3.1 General

This clause describes the steps to generate the high band excitation from the low band ACELP core. To generate a high-band excitation signal that preserves the harmonic structure of the low band excitation signal, a nonlinear function is used. The time-domain harmonic extension of the low band excitation is performed after sufficient over-sampling in order to minimize aliasing.

5.5.8.5.3.2 Resampling of LB excitation

As shown in Figure 21, an up-sampled low band excitation signal is derived from both the periodic (adaptive codebook, ACB) and aperiodic (fixed codebook, FCB) excitation components of the low band ACELP core coder. The ACELP innovation codebook excitation is first scaled by the FCB gain, g_c, and then up-sampled by 2. A simple linear interpolator is used to perform the resampling of the scaled fixed codebook excitation by a factor of 2 as shown in Figure 21. The ACB component in the resampled domain is obtained by shifting the past resampled LB excitation, E_{LB}, based on the fractional closed-

loop pitch estimate, T, from the current frame. The up-sampled past excitation samples are scaled by the pitch gain, g_p, and combined with the up-sampled FCB excitation to generate the resampled low band excitation, E_{LB} as shown in Figure 21.

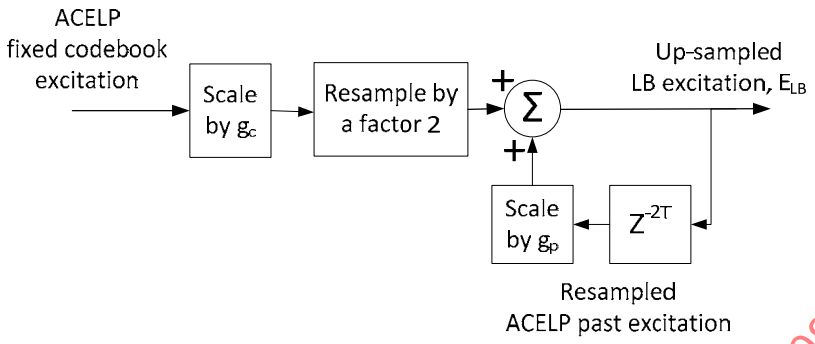


Figure 21 — Overview of low band excitation resampling process

5.5.8.5.3.3 Harmonic extension

The resampled low band excitation signal, E_{LB} , is processed to extend the low band pitch harmonics into the high band. The harmonically extended excitation, E_{HE}, is generated using a nonlinear function, based on the tbe_nlConfig flag from tbe_data().

$$E_{HE} = \begin{cases} |E_{LB}|, & if\ tbe\_nlConfig = 1 \\ \varepsilon_N\ sign(E_{LB})\ E_{LB}^2, & if\ tbe\_nlConfig\ = 0 \\ H_{LP}(z)\ \varepsilon_N\ sign(E_{LB})\ E_{LB}^2 + H_{HP}(z)|E_{LB}|, & if\ tbe\_nlConfig = 0\ \&\&\ idxMixConfig \le 1 \end{cases}$$

where ε_N is the energy normalization factor between E_{LB} and E_{LB}^2. The above NL generation is for tbe_heMode=0 case. In the case, when tbe_heMode flag=1, the NL generation uses the tbe_nlConfig=1 configuration if idxSubGains is an odd value, and the tbe_nlConfig=0 configuration otherwise. In particular, the energy normalization factor is the ratio of frame energies of E_{LB} and E_{LB}^2. The transfer functions $H_{LP}(z)$ and $H_{HP}(z)$ correspond to low pass and high pass filters with a cut-off frequency $3f_s/4$. The transfer functions are given as follows:

$$H_{LP}(z) = \frac{0{,}57(1 + 2z^{-1} + z^{-2})}{1 + 0{,}94z^{-1} + 0{,}33z^{-2}}, H_{HP}(z) = \frac{0{,}098\ (1 - 2z^{-1} + z^{-2})}{1 + 0{,}94z^{-1} + 0{,}33z^{-2}}$$

5.5.8.5.4 High band LP estimation

5.5.8.5.4.1 General

The de-quantized LSF parameters are first converted to LSP.

```
for (i = 0; i < 10; i++) {
  hbLsp_curr[i] = (float) cos(qLsf[i] * 2PI);
}
```

For smoother evolution of the LP polynomial, the LSPs from current frame, hbLsp_curr, are interpolated with LSPs from previous frame over four sub-frames as shown in the pseudocode below. If the previous frame is not a TBE frame (i.e., indicated using first_frame==1), then interpolation is not performed.

```
lspInterCoeff[8] = { 0.7, 0.3, 0.5, 0.5, 0.3, 0.7, 0.1, 0.9 };

if (!first_frame) {
  copyVector(memory->hbLsp_prevmem, hbLsp_prev, 10);
} else {
  copyVector(hbLsp_curr, hbLsp_prev, 10);
}

ptrLspInterpCoef = lspInterCoeff;
for( j = 0; j < 4; j++ ) {
  for( i = 0; i < 10; i++ ) {
    lsp_temp[i] = hbLsp_prev[i] * (*ptrLspInterpCoef)
                + hbLsp_curr[i] * (*(ptrLspInterpCoef+1));
  }

  ptrLspInterpCoef += 2;

  libLsp2A( lsp_temp, lpcShb+j*(11), 10 );
  lpcShb[j*11] = 1.0;
}

copyVector(hbLsp_curr, memory->hbLsp_prevmem, 10);
```

Subsequent to interpolation, the LSPs are converted to LP coefficients as described in ISO/IEC 23003-3:2012, 7.13.11 considering an order of 10.

5.5.8.5.4.2 High band excitation generation

The harmonically-extended excitation, E_{HE}, from subclause 5.5.8.5.3.3 is used as input to the high band excitation generation.

5.5.8.5.4.3 Spectral flip

The harmonic excitation, E_{HE}, is spectrally flipped so that the high band portion of the excitation is modulated down to the low frequency region. This spectral flip is accomplished in the time domain as follows:

$$E_{HE}^{f}(n) = (-1)^n \, E_{HE}(n), \qquad n = 0,1,2 \ldots N-1$$

where $N = 512$ is the number of samples per frame.

```
for(i = 0; i < 512; i++) {
  Ef_HE[i] = ((i%2) == 0) ? (-E_HE[i]) : (E_HE[i]);
}
```

5.5.8.5.4.4 Decimation

The spectrally-flipped harmonic excitation, E_{HE}^{f}, is then decimated using a pair of all-pass filters to obtain an downsampled excitation signal, E_{DS}. This is done by filtering the even samples of, $E_{HE}^{f}(n)$, by an all-pass filter whose transfer function is given by:

$$H_{AP1}(z) = \left(\frac{a_{0,1} + z^{-1}}{1 + a_{0,1}z^{-1}}\right)\left(\frac{a_{1,1} + z^{-1}}{1 + a_{1,1}z^{-1}}\right)\left(\frac{a_{2,1} + z^{-1}}{1 + a_{2,1}z^{-1}}\right)$$

And the odd samples of $E_{HE}^{f}(n)$ are filtered using an all-pass filter whose transfer function is given by:

$$H_{AP2}(z) = \left(\frac{a_{0,2} + z^{-1}}{1 + a_{0,2}z^{-1}}\right)\left(\frac{a_{1,2} + z^{-1}}{1 + a_{1,2}z^{-1}}\right)\left(\frac{a_{2,2} + z^{-1}}{1 + a_{2,2}z^{-1}}\right)$$

The excitation signal, E_{DS}, is estimated by averaging the outputs of the above two filters, $H_{AP1}(z)$ and $H_{AP2}(z)$. The filter coefficients are specified in Table 98.

Table 98 — All-pass filter coefficients for decimation

	All pass filter coefficients
$a_{0,1}$	0,060 565 419 242 91
$a_{1,1}$	0,429 434 015 492 35
$a_{2,1}$	0,808 730 483 065 52
$a_{0,2}$	0,220 630 248 296 30
$a_{1,2}$	0,635 939 439 617 08
$a_{2,2}$	0,941 515 830 956 82

5.5.8.5.4.5 Adaptive spectral whitening

Due to the nonlinear processing applied to obtain the excitation signal, E_{DS}, the spectrum of this excitation is no longer flat. In order to flatten the spectrum of the excitation signal, a fourth-order LP whitening is applied to E_{DS}. The excitation signal is windowed using the Hanning-based window $win_{flatten}$ as given in Annex O.3 prior to calculation of the autocorrelation coefficients. The autocorrelation of the windowed excitation signal is estimated as follows.

$$E_{DS}(n) \quad = E_{DS}(n) \cdot win_{flatten}(n), \qquad n = 0, 1, \ldots, \left(\frac{N}{2} - 1\right), N = 256$$

$$E_{DS}\left(n + \frac{N}{2}\right) = E_{DS}\left(\mathrm{n} + \frac{N}{2}\right) \cdot win_{flatten}\left(\frac{N}{2} - 1 - n\right), n = 0, 1, \ldots, \left(\frac{N}{2} - 1\right), N = 256$$

$$r_{exc}(k) = \sum_{n=0}^{N-1-k} E_{DS}(n) \cdot E_{DS}(n + k), k = 0, 1, \ldots, 4; N = 256$$

A bandwidth expansion is applied to the autocorrelation coefficients by multiplying the coefficients by an expansion function. The expansion function is as given below:

$r_{exc}(k) = r_{exc}(k) * BWexpCoef(k), \quad k = 0,1,2,3,4$

Table 99 — Bandwidth expansion coefficients

	BW expansion coefficients
BWexpCoef[5]	[1.000030000, 0.999876638, 0.999506642, 0.998890286, 0.998028026]

The bandwidth expanded autocorrelation coefficients are used to obtain the LPC using the Levinson-Durbin algorithm. Inverse LP filtering is performed to obtain the whitened excitation, E_{DS_w}. In the tbe_hrConfig==1 mode, the whitened excitation is further modulated by the normalized residual energy (based on idxResSubGains):

$$E_{DS\_w}(n) = E_{DS\_w}(n) * resSubGain\left(\frac{n}{64}\right), \quad k = \frac{n}{64} = 0,1,2,3, \quad n = 0,1,..255$$

In the tbe_hrConfig != 1 mode, the whitened excitation is filtered using an FIR filter that is derived from the idxShbExcResp payload. A FIR filter, $H_{resp}(z)$ is constructed by concatenating the coefficients of tbeExcFilter1 and tbeExcFilter2. The whitened excitation, E_{DS_w}, is filtered using the FIR filter $H_{resp}(z)$.

5.5.8.5.4.6 Envelope modulated noise mixing

The whitened harmonic excitation is further modified by adding random noise whose amplitude is modulated according to the envelope of the whitened excitation, E_{DS_w}. The pseudo-random noise, E_N, is further perceptually shaped using the high band perceptually-weighted filter (PWF = A(z/γ_1)/A(z/γ_2) using $\gamma_1 = 0{,}55$ and $\gamma_2 = 0{,}7$. The pseudo-random noise can be generated using the random noise generator utility function TBE_genRandVec()as described with the following pseudo code.

```
TBE_genRandVec(){
  k1 = (TBE_randomSign(seed1, idx1) < 0) ? -563.154 : 563.154;
  k2 = (TBE_randomSign(seed2, idx2) < 0) ? -225.261 : 225.261;
    for (i = 0; i < length; i++, idx1++, idx2++) {
      idx1 &= 0x00ff;
      idx2 &= 0x00ff;
      output[i] = k1 * RVEC[idx1] + k2 * RVEC[idx2];
  }
}

TBE_randomSign(seed, idx) {
  seed = (short)(seed * 31821L + 13849L);  /* random number generator */
  idx = ABS((short)((float)seed * 0.0078f));  /* ABS is the absolute operator */
  return (seed < 0? -1: 1);
}
```

where `seed1` and `seed2` are initialized to `23` and `59` respectively, if the previous frame was not a TBE frame.

The ratio at which the whitened excitation and the envelope-modulated noise, E_{EM_N}, are mixed is dependent on how strongly-voiced the speech segment is. For example, the envelope-modulated noise and the spectrally whitened excitation are mixed as follows to estimate the high band excitation:

$$E_{HB}(n) = \sqrt{VF_i}\left(E_{DS_W}(n)\right) + \sqrt{P1/P2(1 - VF_i)}\ \left(E_{EM_N}(n)\right),\ \ n = 0{,}1{,}2\ldots{,}127$$

where VF_i are the voice factors derived from LB as explained below, P1 and P2 are the power estimated from spectrally whitened excitation E_{DS_W} and the envelope modulated noise, E_{EM_N}. In particular, given that the fine signal structure in the higher bands is closely related to that in the lower band, the mixing ratio may be estimated from the low band core ACELP parameters. For example, the voicing from low band is normalized to a smaller scale, i.e., $\alpha_i = 0{,}95 * \alpha_i - 0{,}05$ and then mapped to a voice factor as follows. For each subframe, i, the normalized correlation, α_i, from the low band is mapped to a voice factor parameter, VF_i

$$VF_i = \frac{1}{1 + e^{-4\alpha_i}},\ \ i = 1{,}2{,}3{,}4$$

Next the voice factors, VF_i, are limited to the range of [0, 1]. The voice factors undergo further smoothing to compensate for any sudden variations in the low band voicing within a frame. For the HR configuration, the voicing factors are modified based on the idxMixConfig to compensate for any mismatch between the LB and HB voicing. Next the envelope-modulated noise is power normalized such that it is at the same level as that of the harmonic excitation (as shown in the equation to estimate E_{HB} above). At each sub-frame, i, the harmonic excitation that is scaled by the factor, VF_i, and the

normalized modulated noise that is scaled by the factor $(1 - VF_i)$ are mixed to generate the high band excitation.

5.5.8.5.4.7 High band synthesis

The high band excitation is then passed through the high band LP sub-frame synthesis filters to obtain the spectrally shaped excitation. In the tbe_hrConfig==1 mode, first a memory-less synthesis is performed (with past LP filter memories set to zero) and the energy of the synthesized high band is matched to that of the target signal energy (based on idxShbFrGain). In the subsequent step, the scaled or energy compensated excitation signal is used to perform synthesis to obtain the spectrally shaped excitation, S_{HB}. The spectrally shaped high band signal is then scaled using the decoded gain shapes. The gain shape scaled highband signal is finally multiplied by the decoded gain frame to obtain the gain adjusted high band synthesized signal. The gain shapes and gain frame are applied on the synthesized high band, S_{HB}, as follows:

$$S'_{HB}(n) = GF \sum_{j=0}^{3} w(n - j64)gs(j)S_{HB}(n), n = 0,1,2 \dots 271$$

where $gs(j)$ j=0,1,2,3 are the gain shapes, GF is the gain frame, and the window w(n) is defined as follows.

Table 100 — SHB gain shape synthesis window, w(n)

w(n), n = 0,1,2..15	0.007312931, 0.033416940, 0.077119580, 0.136556101, 0.209438491, 0.293364150, 0.385224703, 0.481317588, 0.577845599, 0.671497772, 0.758927143, 0.836651580, 0.901448444, 0.951083203, 0.984171147, 1.000000000
w(n), n=16, 17,..63.	1.0
w(n), n=64,65, 79	0.984171147, 0.951083203, 0.901448444, 0.836651580, 0.758927143, 0.671497772, 0.577845599, 0.481317588, 0.385224703, 0.293364150, 0.209438491, 0.136556101, 0.077119580, 0.033416940, 0.007312931, 0
w(n), for other n	0

5.5.8.5.4.8 Resampling of HB synthesis

The gain adjusted HB synthesis is up-sampled by 2 and flipped (i.e., flip from low to high band, as described in subclause 5.5.8.5.4.3) to generate a high band component associated with the final decoded speech as shown in Figure 22. The upsampling is based on an all-pass filter as shown below:

$$H_{APInterp,1}(z) = \left(\frac{b_{0,1} + z^{-1}}{1 + b_{0,1}z^{-1}}\right)\left(\frac{b_{1,1} + z^{-1}}{1 + b_{1,1}z^{-1}}\right)\left(\frac{b_{2,1} + z^{-1}}{1 + b_{2,1}z^{-1}}\right)$$

and

$$H_{APInterp,2}(z) = \left(\frac{b_{0,2} + z^{-1}}{1 + b_{0,2}z^{-1}}\right)\left(\frac{b_{1,2} + z^{-1}}{1 + b_{1,2}z^{-1}}\right)\left(\frac{b_{2,2} + z^{-1}}{1 + b_{2,2}z^{-1}}\right)$$

Table 101 — All-pass filter coefficients for interpolation by a factor of 2

	All pass coefficients
$b_{0,1}$	0,060 565 419 242 91
$b_{1,1}$	0,429 434 015 492 35
$b_{2,1}$	0,808 730 483 065 52
$b_{0,2}$	0,220 630 248 296 30
$b_{1,2}$	0,635 939 439 617 08
$b_{2,2}$	0,941 515 830 956 82

The low band is up-sampled to the same sampling rate as the high band and mixed with the high band component.

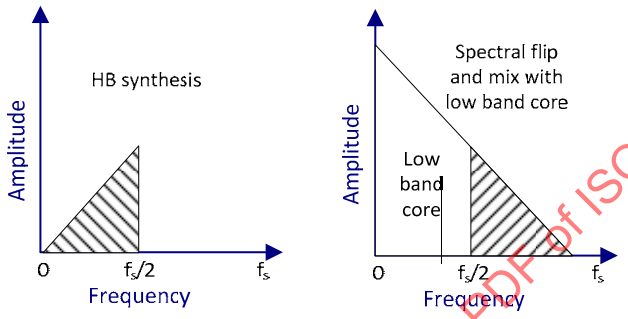


Figure 22 — Spectral flip of HB synthesis

5.5.8.5.4.9 Switching transition signal generation

When switching from ACELP to TCX core, and thus from TBE to IGF, or vice versa, the transition of the high-band signals is performed implicitly by the cross-fade transition mechanism of the core signals as described in subclause 5.5.7.11. To fill the gap caused by the TBE delay (see subclause 5.5.8.5.4.10) and provide overlapping signals for cross-fading, a transition signal syn_{trans} is generated as follows.

To obtain a continuous high band signal to the previous frame, the overlap portion of the high band synthesized signal syn_{scaled} is used, as described in subclause 5.5.8.5.4.7. This overlap portion $syn_{overlap}$ is upsampled using the same filter $H_{I,1}$ and $H_{I,2}$ and flipped subsequently as described in subclause 5.5.8.5.4.8, resulting in the upsampled high band signal $syn_{overlap,up}$.

The targeted length of syn_{trans} is given by:

$$N_{syn,trans} = FrameLength/2 + del_{TDres} - N_{align} = 74\ samples$$

where $N_{align} = 212\ samples$ and $FrameLength/2$ are explained in subclause 5.5.8.5.4.10 and $del_{TDres} = 30\ samples$ is the delay of the TD resampler, which is used for cross-fading, see subclauses 5.5.7.5 and 5.5.7.11.

To obtain the 74 samples, the 32 samples of $syn_{overlap,up}$ are extrapolated using the temporally mirrored end $syn_{prev,mirror}$ of the high band synthesized signal of the previous frame syn_{prev}, where:

$$syn_{prev,mirror}(n) = syn_{prev}(512 + 1 - n); \quad for\ n = 1, \ldots, 74$$

The signals $syn_{overlap,up}$ and $syn_{prev,mirror}$ are merged to generate syn_{trans} by overlap and add using the window win_{trans} as given in Table 102, as:

$$syn_{trans}(n) = \begin{cases} win_{trans}(33-n) \cdot syn_{overlap,up}(n) + win_{trans}(n) \cdot syn_{prev,mirror}(n); & for\ n = 1, \ldots, 32 \\ syn_{prev,mirror}(n); & for\ n = 33\ \ldots, 74 \end{cases}$$

Table 102 — Window for generation of transition signal

n	win_{trans}(n)	n	win_{trans}(n)	n	win_{trans}(n)
1	0.000000000	12	0.250082925	23	0.758927143
2	0.002057320	13	0.293364150	24	0.799044170
3	0.007312931	14	0.338401147	25	0.836651580
4	0.017743483	15	0.385224703	26	0.870644917
5	0.033416940	16	0.432854509	27	0.901448444
6	0.053210367	17	0.481317588	28	0.927953529
7	0.077119580	18	0.529597392	29	0.951083203
8	0.104915089	19	0.577845599	30	0.969773527
9	0.136556101	20	0.625078725	31	0.984171147
10	0.171354547	21	0.671497772	32	0.993264992
11	0.209438491	22	0.716039678	33	1.000000000

5.5.8.5.4.10 Temporal alignment and mixing of HB and ACELP synthesis

Due to the overlap of half a TCX frame length (256 samples at $f_{s,TCX}$), the ACELP synthesis ($synth_{ACELP,orig}$) is delayed by half a frame length, to be synchronized to TCX frames in case of mode switching, before being played out by the LPD module. To compensate for this delay and the intrinsic delay of the TBE, the TBE HB synthesis needs to be temporally aligned, before mixing both synthesis signals.

The intrinsic TBE delay is composed of the internal gain shape look-ahead which corresponds to $del_{GainShape} = 16\ samples$ at internal sampling frequency (i.e. downsampled domain) as described in subclause 5.5.8.5.4.7 and the look-ahead of the harmonic extension which corresponds to $del_{NL} = 12\ samples$ at fullband sampling frequency as described in subclause 5.5.8.5.3.3. Thus, the entire TBE delay is given by

$$del_{TBE} = 2 \cdot del_{GainShape} + del_{NL} = 44\ samples$$

at fullband sampling frequency. The TBE HB synthesis ($synth_{TBE,orig}$) shall be delayed by

$$N_{align} = FrameLength/2 - del_{TBE} = 212\ samples$$

at fullband sampling frequency to obtain a TBE HB synthesis signal ($synth_{TBE,align}$) aligned with the synchronized ACELP synthesis ($synth_{ACELP,sync}$).

By adding the TBE HB synthesis signal and the synchronized ACELP synthesis, the fullband output signal is obtain, given by

$$sig_{FB}(n) = synth_{ACELP,sync} + synth_{TBE,align}(n); \quad for\ n = 1, \ldots, FrameLength$$

5.5.9 LPD stereo coding

5.5.9.1 Tool description

LPD stereo is a discrete M/S stereo coding, where the mid-channel is coded by the mono LPD core coder and the side signal coded in the DFT domain. The decoded mid signal is output from the LPD mono decoder and then processed by the LPD stereo module. The stereo decoding is done in the DFT domain where the L and R channels are decoded. The two decoded channels are transformed back in the time domain and can be then combined in this domain with the decoded channels from the FD mode. FD mode uses its own stereo tools, i.e. discrete stereo with or without complex prediction described in ISO/IEC 23003-3:2012, 7.12.

5.5.9.2 Data elements

lpd_stereo_stream()	Data element to decode the stereo data for the LPD mode.
lpdStereoIndex	Flag which indicates if LPD stereo is activated.
res_mode	Flag which indicates the frequency resolution of the parameter bands.
q_mode	Flag which indicates the time resolution of the parameter bands.
ipd_mode	Bit field which defines the maximum of parameter bands for the IPD parameter.
pred_mode	Flag which indicates if prediction is used.
cod_mode	Bit field which defines the maximum of parameter bands for which the side signal is quantized.
ild_idx[k][b]	ILD parameter index for the frame k and band b.
ipd_idx[k][b]	IPD parameter index for the frame k and band b.
pred_gain_idx[k][b]	Prediction gain index for the frame k and band b.
cod_gain_idx	Global gain index for the quantized side signal.
fullBandLpd	Flag which indicates if LPD mode is in full band mode.

5.5.9.3 Help elements

ccfl	Core coder frame length.
M	Stereo LPD frame length as defined in Table 103.
band_config()	Function that returns the number of coded parameter bands.
band_limits()	Function that returns the number of coded parameter bands.
max_band()	Function that returns the number of coded parameter bands.
cod_L	Number of DFT lines for the decoded side signal.

5.5.9.4 Decoding process

5.5.9.4.1 General

The stereo decoding is performed in the frequency domain. It acts as a post-processing of the LPD decoder. It receives from the LPD decoder the synthesis of the mono mid signal. The side signal is then predicted and decoded in frequency domain. Left (L) and right (R) channel spectrums are then reconstructed in frequency domain before being resynthesized in the time domain. LPD stereo works with a fixed frame size equal to the size of the ACELP frame independently of the coding mode used in LPD mode.

5.5.9.4.2 Frequency analysis

The DFT spectrum of the frame index i is computed from the decoded frame x of the mid signal of length M as follows:

$$\mathrm{X}_i[\mathrm{k}] = \sum_{n=0}^{N-1} w[n] \cdot x[i \cdot \mathrm{M} + n - L] \cdot e^{-2\pi jkn/N}$$

where N is the size of the signal analysis, w is the analysis window and x the decoded time signal from the LPD decoder at frame index i delayed by the overlap size L of the DFT. M is equal to the size of the ACELP frame at the output sampling rate. The DFT analysis window size N is equal to the LPD stereo frame size plus the overlap size of the DFT. The sizes are depending whether the LPD mode is running in full-band mode as defined in Table 103.

Table 103 — DFT and frame sizes of the stereo LPD

fullBandLpd	ccfl	DFT size *N*	Frame size *M*	Overlap size *L*
0	1024	336	256	80
1	1024	672	512	160
0	768	256	192	64
1	768	512	384	128

The window w is a sine window defined as:

$$w[n] = \begin{cases} \sin\left(\frac{\pi}{2L}\left(n + \frac{1}{2}\right)\right) & \text{for } 0 \le n < L \\ 1 & \text{for } L \le n < M \\ \sin\left(\frac{\pi}{2L}\left(L - M + n + \frac{1}{2}\right)\right) & \text{for } M \le n < M + L \end{cases}$$

5.5.9.4.3 Configuration of the parameter bands

The DFT spectrum is divided into non-overlapping frequency bands called parameter bands. The partitioning of the spectrum is non-uniform and mimics the auditory frequency decomposition. Two different divisions of the spectrum are possible with bandwidths following roughly either two or four times the equivalent rectangular bandwidths (ERB).

The spectrum partitioning is selected by the data element **res_mode** and defined by the following pseudo-code:

```
function nbands = band_config(N, res_mod)
band_limits[0] = 1;
nbands = 0;
```

```
while(band_limits[nbands++] < (N/2)) {
  if (res_mode == 0) {
    band_limits[nbands] = band_limits_erb2[nbands];
  } else {
    band_limits[nbands] = band_limits_erb4[nbands];
  }
}
nbands--;
band_limits[nbands] = N/2;
return nbands
```

where *nbands* is the total number of parameter bands and *N* the DFT analysis window size. The tables *band_limits_erb2* and *band_limits_erb4* are defined in Table 104. The decoder can adaptively change the resolution of the parameter bands at every two LPD stereo frames.

Table 104 — Parameter band limits in term of DFT index k

Parameter band index *b*	*band_limits_erb2*	*band_limits_erb4*
0	1	1
1	3	3
2	5	7
3	7	13
4	9	21
5	13	33
6	17	49
7	21	73
8	25	105
9	33	177
10	41	241
11	49	337
12	57	
13	73	
14	89	
15	105	
16	137	
17	177	
18	241	
19	337	

The maximum number of parameter bands for IPD is sent within the 2 bits field **ipd_mode** data element:

$$ipd\_max\_band = max\_band[res\_mode][ipd\_mode]$$

The maximum number of parameter bands for the coding of the Side signal is sent within the 2 bits field **cod_mode** data element:

$$cod\_max\_band = max\_band[res\_mode][cod\_mode]$$

The table max_band[ ][ ] is defined in Table 105.

The number of decoded lines to expect for the side signal is then computed as:

$$cod\_L = 2 \cdot (band\_limits[cod\_max\_band] - 1)$$

Table 105 — Maximum number of bands for different code modes

Mode index	*max_band*[0]	*max_band*[1]
0	0	0
1	7	4
2	9	5
3	11	6

5.5.9.4.4 Inverse quantization of stereo parameters

The stereo parameters interchannel level differences (ILD), interchannel phase differences (IPD) and prediction gains are sent either every frame or every two frames depending of flag **q_mode**. If **q_mode** equal 0, the parameters are updated every frame. Otherwise, the parameters values are only updated for odd indices i of the LPD stereo frame within the USAC frame. The index i of the LPD stereo frame within a USAC frame can be either between 0 and 3 in LPD version 0 and between 0 and 1 in LPD version 1.

The ILD are decoded as follows:

$$\mathrm{ILD}_i[\mathrm{b}] = ild\_q[ild\_idx[i][b]\ ], \text{ for } 0 \le b < nbands$$

The IPD are decoded for the ipd_max_band first bands:

$$\mathrm{IPD}_i[\mathrm{b}] = \frac{\pi}{4} \cdot ipd\_idx[i][b]\ - \pi, \text{ for } 0 \le b < ipd\_max\_band$$

The prediction gains are only decoded if **pred_mode** flag is set to one. The decoded gains are then:

$$pred\_gain_i[\mathrm{b}] = \begin{cases} 0 & , \text{for } 0 \le b < cod\_\max\_band \\ res\_pred\_gain\_q[pred\_gain\_idx[i][b]\ ] & , \text{for } cod\_\max\_band \le b < nbands \end{cases}$$

If the pred_mode is equal to zero, all gains are set to zero.

Independently of the value of **q_mode**, the decoding of the side signal is performed every frame if **cod_mode** is a non-zero value. It first decodes a global gain:

$$cod\_gain_i = 10^{cod\_gain\_idx[i]/(20 \cdot \frac{127}{90})}$$

The decoded shape of the Side signal is the output of the AVQ described in ISO/IEC 23003-3:2012, 7.12.

$$S_i[1 + 8k + n] = kv[k][0][n], \text{ for } 0 \le n < 8 \text{ and } 0 \le k < \frac{cod\_L}{8}$$

Table 106 — Inverse quantization table ild_q[]

index	output	index	Output
0	-50	16	2
1	-45	17	4
2	-40	18	6
3	-35	19	8
4	-30	20	10
5	-25	21	13
6	-22	22	16

index	output	index	Output
7	-19	23	19
8	-16	24	22
9	-13	25	25
10	-10	26	30
11	-8	27	35
12	-6	28	40
13	-4	29	45
14	-2	30	50
15	0	31	reserved

Table 107 — Inverse quantization table res_pred_gain_q[]

index	output
0	0
1	0,117 0
2	0,227 0
3	0,340 7
4	0,464 5
5	0,605 1
6	0,776 3
7	1

5.5.9.4.5 Inverse channel mapping

The mid signal X and side signal S are first converted to the left and right channels L and R as follows:

$$L_i[\mathrm{k}] = X_i[k] + gX_i[k], \text{ for } band\_limits[b] \le k < band\_limits[b+1]$$
$$R_i[\mathrm{k}] = X_i[k] - gX_i[k], \text{ for } band\_limits[b] \le k < band\_limits[b+1]$$

where the gain g per parameter band is derived from the ILD parameter:

$$\mathrm{g} = \frac{\mathrm{c}-1}{c+1}, \text{where } \mathrm{c} = 10^{ILD_i[b]/20}.$$

For parameter bands below cod_max_band, the two channels are updated with the decoded side signal:

$$L_i[k] = L_i[k] + cod\_gain_i \cdot S_i[k], for 1 \le k < band\_limits[cod\_max\_band]$$
$$R_i[k] = R_i[k] - cod\_gain_i \cdot S_i[k], for 1 \le k < band\_limits[cod\_max\_band]$$

For higher parameter bands, the side signal is predicted and the channels updated as:

$$L_i[\mathrm{k}] = L_i[k] + pred\_gain_i[b] \cdot X_{i-1}[k], \text{for } band\_limits[b] \le k < band\_limits[b+1]$$
$$R_i[\mathrm{k}] = R_i[k] - pred\_gain_i[b] \cdot X_{i-1}[k], \text{for } band\_limits[b] \le k < band\_limits[b+1]$$

Finally, the channels are multiplied by a complex value aiming to restore the original energy and the inter-channel phase of the original stereo signal. First, absolute values a[k] are computed for each frequency:

$$a[k] = \sqrt{2 \cdot \frac{X_i^2[k]}{{L_i}^2[k] + {R_i}^2[k]}}$$

before being employed on the left and right spectra in combination with a rotation angle β:

$$L_i[k] = a[k] \cdot e^{j2\pi\beta} \cdot L_i[k], for band\_limits[b] \le k < band\_limits[b+1]$$
$$R_i[k] = a[k] \cdot e^{j2\pi\beta} \cdot R_i[k], for band\_limits[b] \le k < band\_limits[b+1]$$

where a[k] is bounded between -12 and 12 dB, and where $\beta = "atan2"(\sin(\mathrm{IPD_i}[b]), \cos(\mathrm{IPD_i}[b]) + c)$d where atan2(x,y) is the four-quadrant inverse tangent of x over y.

5.5.9.4.6 Time domain synthesis

From the two decoded spectrums L and R, two time domain signals, l and r, are synthesized by an inverse DFT:

$$l_i[n] = \frac{1}{N}\left(\sum_{k=0}^{N/2} L_i[k] \cdot e^{\frac{2\pi jkn}{N}} + \sum_{k=\frac{N}{2}+1}^{N-1} L_i^*[N-k] \cdot e^{\frac{2\pi jkn}{N}}\right), \quad \text{for } 0 \le n < N$$

$$r_i[n] = \frac{1}{N}\left(\sum_{k=0}^{N/2} R_i[k] \cdot e^{\frac{2\pi jkn}{N}} + \sum_{k=\frac{N}{2}+1}^{N-1} R_i^*[N-k] \cdot e^{\frac{2\pi jkn}{N}}\right), \quad \text{for } 0 \le n < N$$

where * denotes the complex conjugation.

Finally, an overlap-add operation allows reconstructing a frame of M samples:

$$l[i \cdot \mathrm{M} + n - L] = \begin{cases} l_{i-1}[M+n] \cdot w[L-1-n] + l_i[n] \cdot w[n], & \text{for } 0 \le n < L \\ l_i[n], & \text{for } L \le n < M \end{cases}$$

$$r[i \cdot \mathrm{M} + n - L] = \begin{cases} r_{i-1}[M+n] \cdot w[L-1-n] + r_i[n] \cdot w[n], & \text{for } 0 \le n < L \\ r_i[n], & \text{for } L \le n < M \end{cases}$$

5.5.9.4.7 Post-processing

The bass post-processing is applied on two channels separately. The processing is for both channels the same as described in ISO/IEC 23003-3:2012, 7.17.

5.5.9.4.8 Transition from FD mode

The transitions from FD to LPD mode are done first on the decoded mid signal as in mono case. It is achieved by artificially creating a Mid-signal from the time domain signal decoded in FD mode.

$$x[n - ccfl/2] = 0.5 \cdot l_{i-1}[n] + 0.5 \cdot r_{i-1}[n], \text{ for } ccfl \le n < \frac{ccfl}{2} + L\_fac$$

This signal is then conveyed to the LPD decoder for updating the memories and applying the FAC decoding as it is done in the mono case for transitions from FD mode to ACELP. The processing is described in ISO/IEC 23003-3:2012, 7.16. In case of FD mode to TCX, a conventional overlap-add is performed. The LPD stereo decoder receives as input signal a decoded Mid signal where the transition is already done. The stereo decoder outputs then a left and right channel signals which overlap the previous frame decoded in FD mode. The signals are then cross-faded on each channel for smoothing the transition in the left and right channels:

$$l\left[n-\frac{ccfl}{2}+L\_fac\right]$$
$$=\begin{cases} l_{i-1}[ccfl+n] & \text{, for } 0 \le n < \frac{ccfl}{2}-L\_fac-L \\ l_{i-1}\left[ccfl+\frac{ccfl}{2}-L\_fac-L+n\right]\cdot w[L-1-n]+l_i[n]\cdot w[n], & \text{for } 0 \le n < L \\ l_i[n] & \text{, for } L \le n < M \end{cases}$$
$$r\left[n-\frac{ccfl}{2}+L\_fac\right]$$
$$=\begin{cases} r_{i-1}[ccfl+n] & \text{, for } 0 \le n < \frac{ccfl}{2}-L\_fac-L \\ r_{i-1}\left[ccfl+\frac{ccfl}{2}-L\_fac-L+n\right]\cdot w[L-1-n]+r_i[n]\cdot w[n], & \text{for } 0 \le n < L \\ r_i[n] & \text{, for } L \le n < M \end{cases}$$

A schematic illustration of the transitions is depicted in Figure 23 in case LPD is in full-band mode where M=ccfl/2.

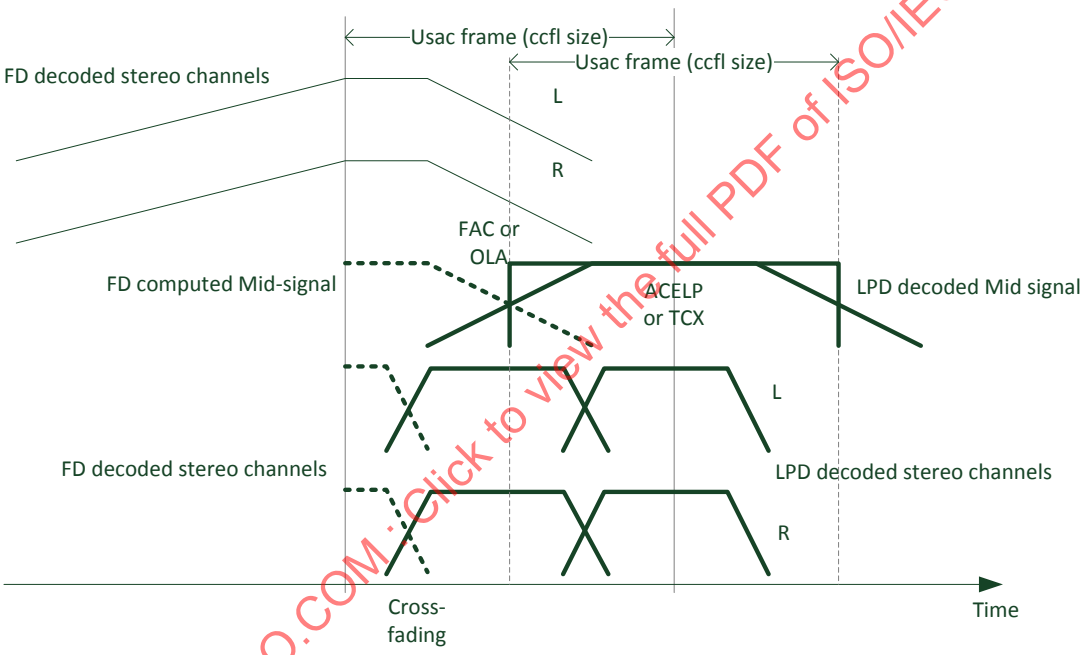


Figure 23 — Schematic representation of the transition from FD to LPD mode with LPD stereo

5.5.9.4.9 Transition to FD mode

For transitions from LPD mode to FD mode, an extra frame is decoded by the LPD stereo decoder. The mid signal coming from the LPD decoder is extended with zero for the frame index i=ccfl/M.

$$x[i\cdot \mathrm{M}+n-L]=\begin{cases} x[i\cdot \mathrm{M}+n-L] & \text{, for } 0 \le n < L+2\cdot L\_fac \\ 0 & \text{, for } L+2\cdot L\_fac \le n < M \end{cases}$$

The stereo decoding as described in the previous subclauses is performed by holding the last stereo parameters, and by switching off the Side signal inverse quantization, i.e. **cod_mode** is set to 0. Moreover, the right side windowing after the inverse DFT is not applied.

The resulting left and right channels are then combined to the FD mode decoded channels of the next frame by using an overlap-add processing in case of TCX to FD mode or by using a FAC for each channel in case of ACELP to FD mode.

In the latter case, the ZIR of the ACELP decoded mid signal shall also be used for the processing of the extra LPD stereo frame. The resulting left and right signals are thus containing channel specific ZIR and folded ACELP synth signals as required for the FAC processing described in ISO/IEC 23003-3:2012, 7.16. Afterwards the left and right signals provided by the LPD stereo tool and the channel specific decoded FAC signals are added to form the final channel dependent FAC synthesis signals.

In case of fullband LPD, i.e. **fullbandLpd** is equal to 1, prior to the addition of the signals only the decoded FAC signal shall be upsampled by the factor of 2, as the ACELP decoded mid signal containing the ZIR signal is already upsampled to fs,tcx prior to the LPD stereo processing. Note that this procedure of upsampling is different than described in 5.5.7.9.

A schematic illustration of the transitions is depicted in Figure 24 in case LPD is in full band mode where M=ccfl/2.

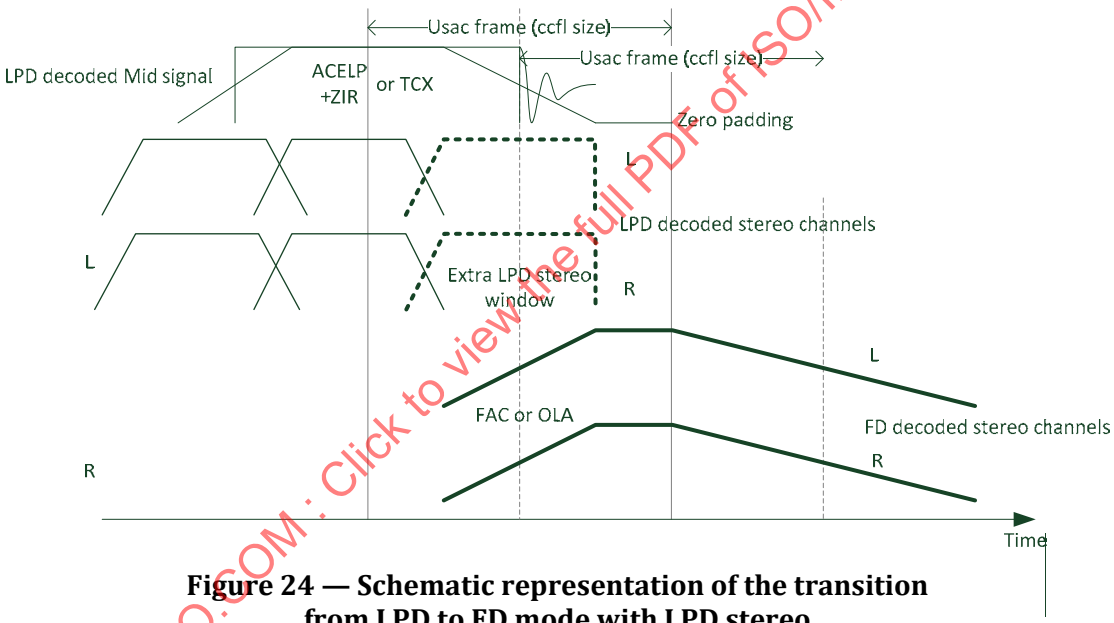


Figure 24 — Schematic representation of the transition from LPD to FD mode with LPD stereo

5.5.10 Multichannel coding tool

5.5.10.1 Tool description

The multichannel coding tool (MCT) is a method for joint coding of multiple channels for more efficient coding of time-variant horizontally and vertically distributed channels.

5.5.10.2 Definitions

Help elements:

mctChanMask[chan]	Indicates the use of the tool for a certain channel according to Table 108. The value of mctChanMask[chan] shall be 0 for any LFE channel.

Table 108 — mctChanMask

mctChanMask	Meaning
0	Multichannel coding tool not applied
1	Multichannel coding tool applied

channelPairIndex — A list of channel pair indices for each pair of channels processed by the MCT. The channelPairIndex is decoded to two channel indices with `decode_channel_pair_index()`.

hasMctMask — Indicates the transmission of a mask that indicates the use of the tool for certain scale factor bands.

Table 109 — hasMctMask

hasMctMask	Meaning
0	MCT is applied to all bands
1	A mask indicating the usage of MCT per band is transmitted

hasBandwiseAngles — Indicates whether a single angle or multiple angles are transmitted.

hasBandwiseCoeff — Indicates whether a single prediction coefficient or multiple prediction coefficients are transmitted.

isMCTShort — Indicates whether the current processing is applied to a frame containing eight sub-windows.

Table 110 — isMCTShort

isMCTShort	Meaning
0	Stereo parameters applied to one MDCT frame
1	Stereo parameters applied to eight MDCT sub-windows

numMaskBands — The number of processing bands that are processed by MCT.

pair — The index of the currently processed stereo processing box.

band — A stereo processing band containing two scalefactor bands.

mctMask [band] — Indicates the activity of the MCT for a certain parameter band within a certain parameter pairing.

mct_delta_time — Indicates the coding scheme used for the MCT parameters:

Table 111 — mct_delta_time

mct_delta_time	Meaning
0	frequency differential coding of MCT parameters
1	time differential coding of MCT parameters

hcod_angle[ ]	The Huffman code book for angles.
dpcm_beta[band]	The differentially encoded angle to be applied.
dpcm_alpha_q_re[band]	The differentially encoded prediction coefficient to be applied.
pred_dir	Indicates the direction of prediction according to ISO/IEC 23003-3:2012, Table 120.
DEFAULT_ALPHA	Initialization value for stereo prediction, equal to 0.
DEFAULT_BETA	Initialization value for rotation angle, equal to 48.
MAX_NUM_MC_BANDS	Maximum number of MCT bands, equal to 64.
MCTSignalingType	The type of signaling MCT data.
keepTree	Indicates whether to use the same tree of channel pairs as in the previous frame.
numPairs	The number of MCT channel pairs. The maximum number of MCT channel pairs per signal group shall not exceed (nMCTChannels · (nMCTChannels -1) / 2) of that signal group.
nMCTChannels	The number of active MCT channels, where mctChanMask[chan] ==1.

5.5.10.3 Decoding process

5.5.10.3.1 General

In case an element with `usacExtElementType ID_EXT_ELE_MCT` belongs to the currently processed signal group, the affected channels according to `mctChanMask[]` shall be decoded by the MCT. Here, the extension element with `usacExtElementType ID_EXT_ELE_MCT` shall be written before any audio element of a certain signal group. Further, the extension element properties **usacExtElementPayloadFrag** shall be zero, **usacExtElementPresent** shall be 1 and the transmitted data shall conform to the syntax element MultichannelCodingFrame() as described in Table 62.

The decoding of the multichannel coding tool (MCT) is performed in multiple steps as follows:

5.5.10.3.2 Decoding of channel pair index

Channel pairs are efficiently signalled using a unique index `channelPairIndex` for each pair, dependent on the sum `nMCTChannels` of active channels in the vector `mctChanMask[]`. The decoding process is described in the function `decode_channel_pair_index()` as follows:

```
decode_channel_pair_index(channelPairIndex, channelPair[2])
{
  maxNumPairIdx = nMCTChannels*(nMCTChannels-1)/2 - 1;
  numBits = floor(log2(maxNumPairIdx))+1;
  pairCounter = 0;

  for (chan1=1; chan1 < nMCTChannels; chan1++) {
      for (chan0=0; chan0 < chan1; chan0++) {
        if (pairCounter == channelPairIndex) {
          channelPair[0] = chan0;
          channelPair[1] = chan1;
          return;
        }
        else
          pairCounter++;
      }
    }
  }
}
```

For instance, all possible channel pairs when using 6 channels can be indexed according to Table 112.

Table 112 — Coding of channelPairIndex for a setup with 6 MCT channels

ch nr	0	1	2	3	4	5
0		0	1	2	3	4
1			5	6	7	8
2				9	10	11
3					12	13
4						14
5						

5.5.10.3.3 Decoding process for rotation angles

In case `MCTSignalingType = 1`, the `MultichannelCodingBoxRotation()` bitstream element is used. For all rotation angles the difference to a preceding (in time or frequency) value is coded using the Huffman code book specified in subclause 5.5.10.3.6. See ISO/IEC 14496-3:2009, 4.6.3, for a detailed description of the Huffman decoding process. Rotation angles are not transmitted for `mctMask[band] = 0`. The following pseudo code describes how to decode the rotation angles `pairBeta[band]`.

```
decode_rotation()
{
  for(pair=0; pair<numPairs; pair++) {
    mctBandsPerWindow = numMaskBands[pair]/windowsPerFrame;
    for(band=0; band<numMaskBands[pair]; band++) {
      if(mct_delta_time[pair] > 0) {
        lastVal = beta_prev_frame[pair][band%mctBandsPerWindow];
      }
      else {
        if ((band % mctBandsPerWindow) == 0) {
          lastVal = DEFAULT_BETA;
        }
```

```
      }
      if (mctMask[pair][band] > 0 ) {

        newBeta = lastVal + dpcm_beta[pair][band];
        if(newBeta >= 65) {
          newBeta -= 65;
        }
        pairBeta[pair][band] = newBeta;
        beta_prev_frame[pair][band%mctBandsPerWindow] = newBeta;
        lastVal = newBeta;
      }
      else {
        beta_prev_frame[pair][band%mctBandsPerWindow] = DEFAULT_BETA; /* -45° */
      }

      /* reset fullband angle */
      beta_prev_fullband[pair] = DEFAULT_BETA;
    }
    for(band=bandsPerWindow; band<MAX_NUM_MC_BANDS; band++) {
      beta_prev_frame[pair][band] = DEFAULT_BETA;
    }
  }
}
```

`beta_prev_frame[pair][sfb]` contains the decoded rotation angles of the corresponding stereo channel pair of the last sub-window of the previous frame. If no differential coding was used for the previous frame or for the respective scale factor band in the previous frame, `beta_prev_frame[sfb]` is set to `DEFAULT_BETA`.

All rotation angles shall be reset to `DEFAULT_BETA` upon a transform length change and for all cases the memory is not used for the current frame.

5.5.10.3.4 Decoding process for real-valued stereo prediction

In case `MCTSignalingType == 0`, the `MultichannelCodingBoxPrediction()` bitstream element is used. Decoding is performed similar to the decoding of prediction coefficients as defined in ISO/IEC 23003-3:2012, 7.7.2.3.2. The following pseudo code describes how to decode the prediction coefficients `pairAlpha[band]`.

```
decode_prediction()
{
  for(pair=0; pair<numPairs; pair++) {
    mctBandsPerWindow = numMaskBands[pair]/windowsPerFrame;
    for(band=0; band<numMaskBands[pair]; band++) {
      if(mct_delta_time[pair] > 0) {
        lastVal = alpha_prev_frame[pair][band%mctBandsPerWindow];
      }
      else {
        if ((band % mctBandsPerWindow) == 0) {
           lastVal = DEFAULT_ALPHA;
        }
      }
      if (mctMask[pair][band] > 0 ) {

        dpcm_alpha = -decode_huffman() + 60; /* function returns dpcm_alpha_[sfb]*/
        newAlpha = lastVal + dpcm_alpha;

        pairAlpha[pair][band] = newAlpha;
```

```
        alpha_prev_frame[pair][band%mctBandsPerWindow] = newAlpha;
        lastVal = newAlpha;
      }
      else {
        alpha_prev_frame[pair][band%mctBandsPerWindow] = DEFAULT_ALPHA;
      }

      /* reset fullband angle */
      alpha_prev_fullband[pair] = DEFAULT_ALPHA;
    }
    for(band=bandsPerWindow; band<MAX_NUM_MC_BANDS; band++) {
      alpha_prev_frame[pair][band] = DEFAULT_ALPHA;
    }
  }
}
```

All prediction coefficients shall be reset to DEFAULT_ALPHA upon a transform length change and for all cases the memory is not used for the current frame.

5.5.10.3.5 Decoding of quantized rotation angles

To avoid floating point differences of trigonometric functions on different platforms, the following lookup-tables for converting rotation angle indices directly to sin/cos shall be used.

```
tabIndexToSinAlpha[65] = {
  -1.000000f,-0.998795f,-0.995185f,-0.989177f,-0.980785f,
  -0.970031f,-0.956940f,-0.941544f,-0.923880f,-0.903989f,
  -0.881921f,-0.857729f,-0.831470f,-0.803208f,-0.773010f,
  -0.740951f,-0.707107f,-0.671559f,-0.634393f,-0.595699f,
  -0.555570f,-0.514103f,-0.471397f,-0.427555f,-0.382683f,
  -0.336890f,-0.290285f,-0.242980f,-0.195090f,-0.146730f,
  -0.098017f,-0.049068f, 0.000000f, 0.049068f, 0.098017f,
   0.146730f, 0.195090f, 0.242980f, 0.290285f, 0.336890f,
   0.382683f, 0.427555f, 0.471397f, 0.514103f, 0.555570f,
   0.595699f, 0.634393f, 0.671559f, 0.707107f, 0.740951f,
   0.773010f, 0.803208f, 0.831470f, 0.857729f, 0.881921f,
   0.903989f, 0.923880f, 0.941544f, 0.956940f, 0.970031f,
   0.980785f, 0.989177f, 0.995185f, 0.998795f, 1.000000f
};


tabIndexToCosAlpha[65] = {
  0.000000f, 0.049068f, 0.098017f, 0.146730f, 0.195090f,
  0.242980f, 0.290285f, 0.336890f, 0.382683f, 0.427555f,
  0.471397f, 0.514103f, 0.555570f, 0.595699f, 0.634393f,
  0.671559f, 0.707107f, 0.740951f, 0.773010f, 0.803208f,
  0.831470f, 0.857729f, 0.881921f, 0.903989f, 0.923880f,
  0.941544f, 0.956940f, 0.970031f, 0.980785f, 0.989177f,
  0.995185f, 0.998795f, 1.000000f, 0.998795f, 0.995185f,
  0.989177f, 0.980785f, 0.970031f, 0.956940f, 0.941544f,
  0.923880f, 0.903989f, 0.881921f, 0.857729f, 0.831470f,
  0.803208f, 0.773010f, 0.740951f, 0.707107f, 0.671559f,
  0.634393f, 0.595699f, 0.555570f, 0.514103f, 0.471397f,
  0.427555f, 0.382683f, 0.336890f, 0.290285f, 0.242980f,
  0.195090f, 0.146730f, 0.098017f, 0.049068f, 0.000000f
};
```

5.5.10.3.6 Huffman tables for differential rotation angles

The following Huffman tables `huff_ctabAngle[]` and `huff_ltabAngle[]` for the code words and the code word lengths, respectively, shall be used for decoding the rotation angle differences.

```
huff_ctabAngle[65] = {
  0x00000000, 0x0000000B, 0x00000012, 0x0000001B, 0x0000001F,
  0x00000031, 0x0000003A, 0x00000043, 0x00000065, 0x00000073,
  0x00000082, 0x0000009A, 0x000000CE, 0x000000EE, 0x00000106,
  0x0000013A, 0x000001D9, 0x000001DE, 0x00000202, 0x00000261,
  0x0000020F, 0x0000020E, 0x00000263, 0x00000266, 0x00000272,
  0x00000271, 0x00000277, 0x00000276, 0x00000334, 0x00000325,
  0x00000326, 0x00000327, 0x00000324, 0x00000323, 0x00000335,
  0x00000322, 0x00000320, 0x00000321, 0x00000273, 0x00000270,
  0x00000267, 0x00000260, 0x000004C4, 0x000004C5, 0x00000203,
  0x000001DF, 0x000001DA, 0x000001D8, 0x0000019B, 0x000001DB,
  0x00000132, 0x00000100, 0x000000CF, 0x000000CC, 0x0000009B,
  0x00000081, 0x00000072, 0x0000004F, 0x00000042, 0x00000038,
  0x00000030, 0x0000001E, 0x0000001A, 0x00000011, 0x0000000A
};


huff_ltabAngle[65] = {
  0x00000001, 0x00000004, 0x00000005, 0x00000005, 0x00000005,
  0x00000006, 0x00000006, 0x00000007, 0x00000007, 0x00000007,
  0x00000008, 0x00000008, 0x00000008, 0x00000008, 0x00000009,
  0x00000009, 0x00000009, 0x00000009, 0x0000000A, 0x0000000A,
  0x0000000A, 0x0000000A, 0x0000000A, 0x0000000A, 0x0000000A,
  0x0000000A, 0x0000000A, 0x0000000A, 0x0000000A, 0x0000000A,
  0x0000000A, 0x0000000A, 0x0000000A, 0x0000000A, 0x0000000A,
  0x0000000A, 0x0000000A, 0x0000000A, 0x0000000A, 0x0000000A,
  0x0000000A, 0x0000000A, 0x0000000B, 0x0000000B, 0x0000000A,
  0x00000009, 0x00000009, 0x00000009, 0x00000009, 0x00000009,
  0x00000009, 0x00000009, 0x00000008, 0x00000008, 0x00000008,
  0x00000008, 0x00000007, 0x00000007, 0x00000007, 0x00000006,
  0x00000006, 0x00000005, 0x00000005, 0x00000005, 0x00000004
};
```

5.5.10.3.7 Application of multichannel coding tool

5.5.10.3.7.1 General

Reconstruct the spectral coefficients of all channels by iteratively looping over all transmitted stereo boxes and frequency bands as follows.

```
decode_mct()
{
  for (pair=0; pair < numPairs; pair++) {

    mctBandOffset = 0;
    alphaSfb = pairAlpha[pair];
    betaSfb = pairBeta[pair];

    /* inverse MCT application */
    for (win = 0, group = 0; group <num_window_groups; group++) {

      for (groupwin = 0; groupwin < window_group_length[group]; groupwin++, win++) {
        *dmx = spectral_data[ch1][win];
```

```
        *res = spectral_data[ch2][win];
        apply_mct_wrapper(self,dmx,res,
                          &alphaSfb[mctBandOffset], &betaSfb[mctBandOffset],
                          &mctMask[mctBandOffset],mctBandsPerWindow, alpha,
                          pair,nSamples);
      }
      mctBandOffset += mctBandsPerWindow;
    }
  }
}
```

Thereby `spectral_data[ch1]` and `spectral_data[ch2]` represent the two input and output channels of the channel pair that is currently processed in the MCT stereo processing box.

Further processing of every MCT stereo processing box is achieved as follows.

```
apply_mct_wrapper(self, *dmx, *res,
                  *alphaSfb, *betaSfb,
                  *mctMask, mctBandsPerWindow, alpha,
                  pair, nSamples)
{
  sfb = 0;

  if (MCTSignalingType == 0) {
    if (!bHasBandwiseCoeff[pair] && !bHasMctMask[pair]) {
      apply_mct_prediction(dmx, res, alphaSfb[0], nSamples);
    }
    else {
      /* apply bandwise processing */
      for (i = 0; i< mctBandsPerWindow; i++) {
        if (mctMask[i] == 1) {
          startLine = swb_offset [sfb];
          stopLine  = (sfb+2<num_swb)? swb_offset [sfb+2] : swb_offset [sfb+1];
          nSamples  = stopLine-startLine;

          apply_mct_prediction(&dmx[startLine], &res[startLine],
                               alphaSfb[i], nSamples, pred_dir);
        }
        sfb += 2;

        /* break condition */
        if (sfb >= num_swb) {
          break;
        }
      }
    }
  }
  else if (MCTSignalingType == 1) {

    /* apply fullband box */
    if (!bHasBandwiseAngles[pair] && !bHasMctMask[pair]) {
      apply_mct_rotation(dmx, res, betaSfb[0], nSamples);
    }
    else {
      /* apply bandwise processing */
      for (i = 0; i< mctBandsPerWindow; i++) {
        if (mctMask[i] == 1) {
          startLine = swb_offset [sfb];
          stopLine  = (sfb+2<num_swb)? swb_offset [sfb+2] : swb_offset [sfb+1];
```

```
          nSamples  = stopLine-startLine;

          apply_mct_rotation(&dmx[startLine], &res[startLine],
                             betaSfb[i], nSamples);
        }
        sfb += 2;

        /* break condition */
        if (sfb >= num_swb) {
          break;
        }
      }
    }
  }
  else if (MCTSignalingType == 2) {
    /* reserved */
  }
  else if (MCTSignalingType == 3) {
    /* reserved */
  }
}
```

5.5.10.3.7.2 Application of rotation angles

```
apply_mct_rotation(*dmx, *res, aIdx, nSamples)
{
  for (n=0;n<nSamples;n++) {
    L =  dmx[n] * tabIndexToCosAlpha [aIdx] - res[n] * tabIndexToSinAlpha [aIdx];
    R =  dmx[n] * tabIndexToSinAlpha [aIdx] + res[n] * tabIndexToCosAlpha [aIdx];
    dmx[n] = L;
    res[n] = R;
  }
}
```

5.5.10.3.7.3 Application of real-valued stereo prediction coefficients

Real-valued stereo prediction is performed like the upmixing process described in ISO/IEC 23003-3:2012, subclause 7.7.2.3.4 under the assumption that `ms_mask_present = 3; num_window_groups = windowsPerFrame; window_group_length = 1; cplx_pred_used[g][sfb] = mctMask[sfb]; alpha_im = 0;`

Thus, in the context of the MCT, the prediction upmixing process can be calculated using the following pseudo code.

```
apply_mct_prediction(*dmx, *res, alpha_q, nSamples, pred_dir)
{
  alpha_re = alpha_q * 0.1;

  for (n=0;n<nSamples;n++) {
    if (pred_dir == 0) {
      L = dmx[n] + alpha * dmx[n] + res[n];
      R = dmx[n] - alpha * dmx[n] - res[n];

    }
    else {
      L =   dmx[n] + alpha * dmx[n] + res[n];
      R = - dmx[n] + alpha * dmx[n] + res[n];
```

```
      }
      dmx[n] = L;
      res[n] = R;
    }
}
```

5.5.10.4 Stereo filling in the MCT

Like stereo filling for IGF in a channel pair element, described in subclause 5.5.5.4.9, stereo filling in the multichannel coding Tool (MCT) fills "empty" scale factor bands (which are fully quantized to zero) at and above the noise filling start frequency using a downmix of the previous frame's output spectra.

5.5.10.4.1 Tool description

When stereo filling is active in a MCT joint-channel pair (**hasStereoFilling**[pair] ≠ 0 in Table 62), all "empty" scale factor bands in the noise filling region (i.e. starting at or above noiseFillingStartOffset) of the pair's second channel are filled to a specific target energy using a downmix of the corresponding output spectra (after MCT application) of the previous frame. This is performed after the FD noise filling (see ISO/IEC 23003-3:2012, 7.2) and prior to scale factor and MCT joint-stereo application. All output spectra after completed MCT processing are saved for potential Stereo Filling in the next frame.

5.5.10.4.2 Operational constraints

Cascaded execution of stereo filling algorithm (**hasStereoFilling**[pair] ≠ 0) in empty bands of the second channel is not supported for any following MCT stereo pair with **hasStereoFilling**[pair] ≠ 0 if the second channel is the same. In a channel pair element, active IGF stereo filling in the second (residual) channel according to subclause 5.5.5.4.9 takes precedence over – and, thus, disables – any subsequent application of MCT stereo filling in the same channel of the same frame.

5.5.10.4.3 Definitions

hasStereoFilling[pair]	indicates usage of stereo filling in currently processed MCT channel pair.
ch1, ch2	indices of channels in currently processed MCT channel pair.
spectral_data[ ][ ]	spectral coefficients of channels in currently processed MCT channel pair.
spectral_data_prev[ ][ ]	output spectra after completed MCT processing in previous frame.
downmix_prev[ ][ ]	estimated downmix of previous frame's output channels with indices given by currently processed MCT channel pair.
num_swb	total number of scale factor bands, see ISO/IEC 23003-3:2012, subclause 6.2.9.4.
ccfl	coreCoderFrameLength, transform length, see ISO/IEC 23003-3:2012, subclause 6.1.
noiseFillingStartOffset	Noise filling start line, defined depending on ccfl in ISO/IEC 23003-3:2012, Table 109.
igf_WhiteningLevel	Spectral whitening in IGF, see subclause 5.5.5.4.7.

seed[ ]	Noise filling seed used by randomSign(), see ISO/IEC 23003-3:2012, subclause 7.2.

5.5.10.4.4 Decoding process

MCT stereo filling is performed using four consecutive operations, which are described in the following steps.

Step 1: Preparation of second channel's spectrum for stereo filling algorithm

If the stereo filling indicator for the given MCT channel pair, **hasStereoFilling**[pair], equals zero, stereo filling is not used and the following steps are not executed. Otherwise, scale factor application is reversed if it was previously applied to the pair's second channel spectrum, spectral_data[ch2].

Step 2: Generation of previous downmix spectrum for given MCT channel pair

The previous downmix is estimated from the previous frame's output signals spectral_data_prev[ ][ ] that was stored after application of MCT processing. If a previous output channel signal is not available, e.g. due to an independent frame (indepFlag>0), a transform length change or core_mode == 1 , the previous channel buffer of the corresponding channel shall be set to zero.

For prediction stereo pairs, i.e. MCTSignalingType == 0, the previous downmix is calculated from the previous output channels as downmix_prev[ ][ ] defined in step 2 of subclause 5.5.5.4.9.4, whereby spectrum[window][ ] is represented by spectral_data[ ][window].

For rotation stereo pairs, i.e. MCTSignalingType == 1, the previous downmix is calculated from the previous output channels by inverting the rotation operation defined in subclause 5.5.10.3.7.1.

```
apply_mct_rotation_inverse(*R, *L, *dmx, aIdx, nSamples)
{
  for (n=0; n<nSamples; n++) {
    dmx =  L[n] * tabIndexToCosAlpha[aIdx] + R[n] * tabIndexToSinAlpha[aIdx];
  }
}
```

using L = spectral_data_prev[ch1][ ], R = spectral_data_prev[ch2][ ], dmx = downmix_prev[ ] of the previous frame and using aIdx, nSamples of current frame and MCT pair.

Step 3: Execution of stereo filling algorithm in empty bands of second channel

Stereo filling is applied in the MCT pair's second channel as in step 3 of subclause 5.5.5.4.9.4, whereby spectrum[window] is represented by spectral_data[ch2][window] and max_sfb_ste is given by num_swb.

Step 4: Scale factor application and adaptive synchronization of Noise Filling seeds

As after step 3 of subclause 5.5.5.4.9.4, the scale factors are applied on the resulting spectrum as in ISO/IEC 23003-3:2012, 7.3, with the scale factors of empty bands being processed like regular scale factors. In case a scale factor is not defined, e.g. because it is located above max_sfb, its value shall equal zero. If IGF is used, igf_WhiteningLevel equals 2 in any of the second channel's tiles, and both channels do not employ eight-short transformation, the spectral energies of both channels in the MCT pair are computed in the range from index noiseFillingStartOffset to index ccfl/2 – 1 before executing decode_mct( ). If the computed energy of the first channel is more than eight times greater than the

energy of the second channel, the second channel's seed[ch2] is set equal to the first channel's seed[ch1].

5.5.11 Filterbank and block switching

The frequency-to-time transformation, windowing, block switching, and overlap-and-add operations are carried out as specified in ISO/IEC 23003-3:2012, subclause 7.9. The only exception is the analytical expression for the inverse lapped transform $x_{i,n}$ of the spectral coefficients $spec[i]$ for the window index i, which is now given by:

$$x_{i,n} = \frac{2}{N}\sum_{k=0}^{\frac{N}{2}-1} spec[i][k] \cdot \mathrm{cs}\left(\frac{2\pi}{N}(n+n_0)(k+k_0)\right) \qquad \text{for } 0 \le n < N,$$

with i, k, n, N, and n_0 defined as in ISO/IEC 23003-3:2012, 7.9.3.1, and with cs() and k_0 as tabulated, using the prev_aliasing_symmetry and curr_aliasing_symmetry values, in Table 113 below. Note that for the 7 last transforms (at $i > 0$) of an EIGHT_SHORT_SEQUENCE, prev_aliasing_symmetry is set to curr_aliasing_symmetry.

Table 113 — Mapping of aliasing symmetry values to parameters of the inverse lapped transform $x_{i,n}$

last frame *i*-1	**current frame *i***	
	right-side symmetry even ($symm_i$=0)	**right-side symmetry odd ($symm_i$=1)**
right-side symmetry even ($symm_{i-1}$=0)	cs(...) = cos(...) k_0 = 0.5	cs(...) = sin(...) k_0 = 1.0
right-side symmetry odd ($symm_{i-1}$=1)	cs(...) = cos(...) k_0 = 0.0	cs(...) = sin(...) k_0 = 0.5
NOTE $symm_{i-1}$ = value of prev_aliasing_symmetry, $symm_i$ = value of curr_aliasing_symmetry.		

In channels and frames with LPD coding (core_mode[ch] ≠ 0), prev_aliasing_symmetry and curr_aliasing_symmetry shall be zero.

5.5.12 Frequency domain prediction

5.5.12.1 Tool description

The frequency domain prediction (FDP) tool can be utilized for subjective quality improvement of low-frequency harmonic signal components. It is largely designed using fixed point arithmetic in order to ensure consistent operation across different platforms. FDP is applied individually for each channel of the given element in the TNS filtered (and in the case of channel pair elements, the joint-stereo coded) MDCT spectral domain, as obtained after the entropy decoding and noise filling steps, and is supported in both the MDCT based TCX and FD coding modes.

5.5.12.2 Operational constraints

The FDP tool is only available in dependently coded channels/frames (i. e. indepFlag == 0) which are transform coded using the maximum MDCT length (i. e. largest available mod[k] in case of TCX and window_sequence != EIGHT_SHORT_SEQUENCE in case of FD coding) and for which no transition from TCX to FD coding, or vice versa, occurred between the last and current frame. If these requirements are not satisfied, the FDP indicator, **fdp_data_present**, should equal zero, and all FDP helper states (see 5.5.12.3) shall be set to zero.

5.5.12.3 Definitions

fdp_data_present	binary flag indicating whether the FDP tool is active (1) or disabled (0) in the channel.
fdp_spacing_index	eight-bit integer holding the harmonic spacing index used during the FDP processing.
ccfl	coreCoderFrameLength, the transform length, see ISO/IEC 23003-3:2012, subclause 6.1.
g	re-scaling gain, based on global_gain value, see ISO/IEC 23003-3:2012, subclause 7.15.
lg	number of quantized MDCT bins, see ISO/IEC 23003-3:2012, subclauses 6.2.9.2 and subclause 7.15.
noiseFillingStartOffset	noise filling start line, defined depending on ccfl in ISO/IEC 23003-3:2012, Table 109.
samplingFrequency	core-coder sample rate defined by usacSamplingFrequency or usacSamplingFrequencyIndex as defined in Table 12.
harmonicSpacing	helper element, unsigned integer holding a harmonic spacing value for FDP decoding.
predictionBandwidth	helper element, unsigned integer holding the maximum line count for FDP decoding.
quantSpecPrev[ ][ ]	helper array, internal signed-integer MDCT line memory for the inter-frame prediction.
fdp_exp[ ]	constant array holding the integer line-expansion data $NINT\left(64 \cdot i^{4/3}\right)$, with 0 ≤ i ≤ 181.
fdp_scf[ ]	constant array holding the integer scale-factor power data $NINT\left(2^{(s+1)/4}\right)$, with 0 ≤ s ≤ 63.
fdp_sin[ ]	constant array holding the floating-point sine values of $\sin\left(\pi \cdot {}^{i}/_{256}\right)$, with 0 ≤ i ≤ 128.
fdp_int[ ]	output array holding the signed-integer predictor values derived during FDP decoding.

5.5.12.4 Decoding process

The FDP decoding procedure is performed in four consecutive operations, which are described in the following steps.

Step 1: Derivation of harmonic spacing value

If **fdp_data_present** == 0, this step is skipped. Otherwise, harmonicSpacing is derived from **fdp_spacing_index**:

$$\text{harmonicSpacing} = (894 \cdot 512 + \text{fdp\_spacing\_value}) / (2 \cdot \text{fdp\_spacing\_value})$$

with fdp_spacing_value = 894 / 3 – **fdp_spacing_index**. The division in the above equation is an integer division.

Step 2: Determination of prediction bandwidth

If ccfl ≠ 768 and samplingFrequency ≥ 44 100 Hz (i.e. usacSamplingFrequencyIndex < 5), predictionBandwidth equals 132. Otherwise, predictionBandwidth equals the long-window-sequence value of noiseFillingStartOffset. Also, predictionBandwidth is limited to lg, the number of quantized MDCT lines given by the arithmetic decoder:

$$\text{predictionBandwidth} = \min(\text{lg}, \text{predictionBandwidth}).$$

Step 3: Execution of MDCT-domain prediction

The FDP decoding process, which returns the predictor values fdp_int[i], 0 ≤ i < predictionBandwidth, depends on the mode of the current frame and channel. If **fdp_data_present** == 0, all fdp_int[i] = 0. Otherwise, in case of FD coding, FDP decoding is applied to the expanded, scaled MDCT values outputSpecCurr[i] after noise filling.

```
harmIndex = -128, compIndex = 256; /* harmonic and compare indices */
s1 = 0; s2 = 0; /* LPC coefficients, adapt both for each harmonic */

if (fdp_data_present) { /* FDP active and allowed, obtain estimate */
  for (i = 0; i < predictionBandwidth; i++) {
    if (abs(i * 256 - harmIndex) >= 384) {  /* bin is not harmonic */
      fdp_int[i] = 0;
    } else {  /* bin is part of the currently active harmonic line */
      reg32 = s1 * quantSpecPrev[0][i] + s2 * quantSpecPrev[1][i];
      fdp_int[i] = sign(reg32) * (((unsigned int)abs(reg32) + 16384) >> 15);
      outputSpecCurr[i] += i_gain * fdp_int[i]; /* actual decoding */
    }
    if (i * 256 == compIndex) {   /* update indices and LPC coeffs */
      harmIndex += harmonicSpacing;
      compIndex = harmIndex & 255;
      if (compIndex > 128) {
        compIndex = 256 - compIndex; /* exploit trigonom. symmetry */
      }
      s1 = NINT( 768*min(82-18*fdp_sin[compIndex]^2, 77-6*fdp_sin[compIndex]^2)*
           fdp_sin[compIndex]);
      s2 = NINT(-4.5*min(82-18*fdp_sin[compIndex]^2, 77-
6*fdp_sin[compIndex]^2)^2);
      compIndex = harmIndex >> 8;  /* integer unscaled harm. index */
      if ((compIndex & 1) == 0) {
        s1 *= -1; /* negate first LPC coeff for even harm. indices */
```

```
        }
        compIndex = 256 + ((harmIndex + 128) >> 8) * 256;  /* update */
      }
    }
  }
```

Note that, in case of MDCT based TCX coding in the given frame and channel (core_mode[ch] == 1), inverse gain `i_gain` = 64 / g, i.e. an amplified version of the re-scaling gain, and in case of FD coding, `i_gain` = 512.

Step 4: Update of spectral prediction memory

For each line at index 0 ≤ i < predictionBandwidth an integer representation x_int[i] of the expanded, scaled line value is computed. In case of FD coding, this depends on the quantized value x_ac_quant[i] and its associated scale factor scf[sfb] for band sfb (see ISO 23003-3:2012, subclauses 7.1 and 7.2). If scf[sfb] < 21, x_int[i] = fdp_int[i]. Otherwise,

$$x[i] = fdp\_exp[\min(abs(x\_ac\_quant[i]), 181)] \cdot fdp\_scf[\min(scf[sfb] - 21, 63)],$$

$$x\_int[i] = sign(x\_ac\_quant[i]) \cdot ((x[i] + 512) >> 10) + fdp\_int[i] \quad (\text{“>>” is a binary shift}).$$

In case of MDCT based TCX coding, x_int[i] is derived from the x_tcx_invquant[i] coefficients and the gain g:

$$x\_int[i] = x\_tcx\_invquant[i] \cdot NINT(g / 64) + fdp\_int[i].$$

For all 0 ≤ i < predictionBandwidth, the update is finalized using quantSpecPrev[1][i] = quantSpecPrev[0][i] and, afterwards,

$$quantSpecPrev[0][i] = \min(\max(x\_int[i], -31775), 31775).$$

All x_int[ ] associated with uncoded scale factor bands (i.e. bands whose sfb ≥ max_sfb) shall equal zero.

5.5.13 Long-term postfilter

5.5.13.1 Tool description

The long-term postfilter (LTPF) tool can be utilized for subjective quality improvement of low-frequency harmonic signal components. LTPF is applied individually for each channel of the given element in the time domain as obtained after FD/LPD decoding. More specifically, it is applied on the time-domain signal obtained after the overlap-and-add operation in case of the FD mode (see ISO/IEC 23003-3:2012, 7.9.3.3) and after the bass postfilter in case of the LPD mode (see ISO/IEC 23003-3:2012, 7.17).

5.5.13.2 Operational constraints

The LTPF tool is supported in both the FD mode and in the longest MDCT based TCX mode, but not in the shorter MDCT based TCX modes nor in the ACELP mode. However, to avoid any discontinuities that could be introduced when switching from a frame where the tool is supported to a frame where the tool is not supported, the LTPF decoding process is still applied in the modes where the tool is not supported but with **ltpf_data_present** equal zero.

5.5.13.3 Definitions

ltpf_data_present binary flag indicating whether the LTPF tool is active (1) or disabled (0) in the channel.

ltpf_pitch_lag_index nine-bit integer holding the pitch lag index used during the LTPF decoding process.

ltpf_gain_index two-bit integer holding the gain index used during the LTPF decoding process.

pit_int integer part of the pitch lag used during the LTPF decoding process.

pit_fr fractional part of the pitch lag used during the LTPF decoding process.

gain gain used during the LTPF decoding process.

Fs the sampling frequency at which the core coder operates.

ccfl core coder frame length in samples.

5.5.13.4 Decoding process

5.5.13.4.1 General

The LTPF tool processes the output signal of the FD/LPD core decoder with an IIR filter, whose coefficients are derived from three parameters that are decoded from the bitstream. These parameters are estimated at the encoder side on a frame of length ccfl whose middle point coincides with the middle point of the MDCT window. The frame of output signal coming from the FD/LPD core decoder is however delayed by ccfl/2. At the decoder side, the LTPF tool then filters the first half of the current frame using the parameters decoded in the previous frame and filters the second half of the current frame using the parameters decoded in the current frame. To avoid any discontinuities that could be introduced when the filter parameters change between the previous and the current frame, the beginning portion of the second half of the current frame is processed with a transition filter.

The decoding of the filter parameters is described in subclause 5.5.13.4.2. The IIR filtering of the core decoder output signal is described in subclause 5.5.13.4.3. The transition filter used to remove possible discontinuities is described in subclause 5.5.13.4.4.

5.5.13.4.2 Decoding of the filter parameters

5.5.13.4.2.1 General

There are three parameters per frame: the integer part of the pitch lag, the fractional part of the pitch lag, and the gain. If ltpf_data_present equals zero, then the three parameters are set to zero. Otherwise, the parameters are decoded as described in the following subclauses.

5.5.13.4.2.2 Decoding of the integer and fractional parts of the pitch lag

A fractional pitch delay is used with resolutions 1/2 in the range [pit_min, pit_fr2-1/2], integers only in the range [pit_fr2, pit_fr1-1], and integers with increments by 2 in the range [pit_fr1, pit_max]. pit_min, pit_fr2, pit_fr1 and pit_max are the boundaries of the segments of the quantizers which depend on Fs and they are determined as follows.

```
pit_min = round( 34 * ( Fs / 2 ) / 12800 ) * 2;
pit_fr2 = 324 - pit_min;
```

```
pit_fr1 = 320;
pit_max = 54 + 6 * pit_min;
```

The integer and fractional parts of the pitch lag are then decoded as follows.

```
if ( ltpf_pitch_lag_index < (pit_fr2-pit_min)*2 )
{
  pit_int = pit_min + (ltpf_pitch_lag_index/2);
  pit_fr = ltpf_pitch_lag_index – (pit_int - pitmin)*2;
}
else if ( ltpf_pitch_lag_index < (pit_fr2-pit_min)*2 + (pit_fr1-pit_fr2) )
{
  pit_int = pit_fr2 + ltpf_pitch_lag_index - (pit_fr2-pit_min)*2;
  pit_fr = 0;
}
else
{
  pit_int = (ltpf_pitch_lag_index-(pit_fr2-pit_min)*2-(pit_fr1-pit_fr2))*2 +
pit_fr1;
  pit_fr = 0;
}
```

5.5.13.4.2.3 Decoding of the gain

The gain is decoded as follows.

```
gain = (ltpf_gain_index + 1) * 0.0625;
```

5.5.13.4.3 IIR filtering

The LTPF processes the core decoder output with an IIR filter whose coefficients are derived from the integer and fractional parts of the pitch lag and from the gain. The IIR filter is implemented with the function given below, assuming filtering a portion of signal, where *x points to the first sample of the portion of input signal, *y points to the first sample of the portion of output signal, and N is the length of the portion of signal.

```
function ltpf_filter( *x, *y, N, gain, ltpf_gain_index, pit_int, pit_fr )

if ( gain == 0 )
{
  for ( n = 0; n < N; n++ )
  {
    y[n] = x[n];
  }
}
else
{
  for ( n = 0; n < N; n++ )
  {
    s1 = 0;
    for ( k = 0; k < 8; k++ )
    {
      s1 += y[n-pit_int+k-4] * ltpf_filter_coef1[pit_fr][k];
    }
    s2 = 0;
    for ( k = 0; k < 7; k++ )
    {
      s2 += x[n-k] * ltpf_filter_coef2[ltpf_gain_index][k];
```

```
    }
    y[n] = x[n] + gain * s1 - 0.95 * gain * s2;
  }
}
```

The two tables ltpf_filter_coef1 and ltpf_filter_coef2 are given below.

```
ltpf_filter_coef1[2][8] =
{
  {0.0000000,0.0304386,0.1162701,0.2195613,0.2674597,0.2195613,0.1162701,0.0304386},
  {0.0076226,0.0676508,0.1700032,0.2547232,0.2547232,0.1700032,0.0676508,0.0076226}
}

ltpf_filter_coef2[4][7] =
{
  {0.27150189,0.44286013,0.23027992,0.05759155,-0.00172290,-0.00045168,-0.00005891},
  {0.27581838,0.44682277,0.22783915,0.05410054,-0.00353758,-0.00092331,-0.00011995},
  {0.28044685,0.45103979,0.22519192,0.05037740,-0.00545541,-0.00141719,-0.00018336},
  {0.28543320,0.45554676,0.22230634,0.04638935,-0.00749011,-0.00193612,-0.00024943}
}
```

5.5.13.4.4 Transition filtering

The first half of the current frame is always filtered with the function `ltpf_filter` using the parameters of the previous frame and N=ccfl/2.

If the parameters of the current frame are the same as the ones from the previous frame, the second half of the current frame is also filtered with the function `ltpf_filter` using the same parameters and N=ccfl/2. However, if the parameters change between the previous and the current frame, a discontinuity can be introduced. In that case, the beginning portion (ccfl/8 samples) of the second half of the current frame is processed with a transition filter described in the following. The remaining 3*ccfl/8 samples are then processed with the function `ltpf_filter` using the parameters from the current frame and N=3*ccfl/8.

Three cases are considered:

a) The gain of the previous frame is equal to zero and the gain of the current frame is not equal to zero.

A simple fade-in mechanism is used where the function `ltpf_filter` is applied using the filter parameters of the current frame, N=ccfl/8, and changing the line of code:

```
y[n] = x[n] + gain * s1 - 0.95 * gain * s2;
```

by the following lines of code:

```
y[n] = x[n] + alpha * ( gain * s1 - 0.95 * gain * s2 );
alpha += 1/N;
```

and by setting alpha to zero at the beginning of the function.

b) The gain of the previous frame is not equal to zero and the gain of the current frame is equal to zero.

A simple fade-out mechanism is used where the function `ltpf_filter` is applied using the filter parameters of the previous frame, N=ccfl/8, and changing the line of code


```
y[n] = x[n] + gain * s1 - 0.95 * gain * s2;
```

by the following lines of code

```
y[n] = x[n] + alpha * ( gain * s1 - 0.95 * gain * s2 );
alpha -= 1/N;
```

and by setting alpha to one at the beginning of the function.

c) The gain of the previous frame is not equal to zero and the gain of the current frame is not equal to zero.

The discontinuity is removed using the zero-impulse-response (ZIR) of a LPC synthesis filter estimated on the previous frame, and with memories computed using the filter parameters of the current frame.

The coefficients of the LPC synthesis filter are estimated using the classic autocorrelation and levinson-durbin approach and is implemented using the function given below, where x[] is the portion of LTPF output signal corresponding to the last ccfl/4 samples, a[] are the LPC coefficients, N=ccfl/4 is the length of the portion of signal, and M=24 is the order of the LPC synthesis filter.

```
function ltpf_get_lpc( x[], a[], N, M )

for ( m = 0; m <= M; m++ )
{
  s = 0.0;
  for ( n = 0; n < N-m; n++ )
  {
    s += x[n] * x[n+m];
  }
  r[m] = s;
}
if (r[0] < 100.0)
{
  r[0] = 100.0;
}
r[0] *= 1.0001;
a[0] = 1.0;
rc[0] = -r[1] / r[0];
a[1] = rc[0];
sigma2 = r[0] + r[1] * rc[0];
for ( m = 2; m <= M; m++ )
{
  sum = 0.0f;
  for ( i = 0; i < m; i++ )
  {
    sum += r[m-i] * a[i];
  }
  rc[m-1] = -sum / sigma2;
  sigma2 = sigma2 * ( 1.0 - rc[m-1] * rc[m-1] );
  if ( sigma2 <= 1.0E-09 )
  {
    sigma2 = 1.0E-09;
    for ( i = m; i <= M; i++ )
    {
      rc[i-1] = 0.0;
      a[i] = 0.0;
    }
    break;
```

```
  }
  for ( i = 1; I <= (m/2); i++ )
  {
    value = a[i] + rc[m-1] * a[m-i];
    a[m-i] += rc[m-1] * a[i];
    a[i] = value;
  }
  a[m] = rc[m-1];
}
```

Then, the ZIR of the LPC synthesis filter is computed using the following function, where *x points to the first sample of the beginning portion of the second half of the current frame of input signal, *y points to the first sample of the beginning portion of the second half of the current frame of output signal, and Lz=ccfl/8 is the length of the ZIR.

```
function ltpf_get_zir( *x, *y, a[], zir[], M, Lz )

for ( m = 0; m < M; n++ )
{
  s1 = 0;
  for ( k = 0; k < 8; k++ )
  {
    s1 += y[m-M-pit_int+k-4] * ltpf_filter_coef1[pit_fr][k];
  }
  s2 = 0;
  for ( k = 0; k < 7; k++ )
  {
    s2 += x[m-M-k] * ltpf_filter_coef2[ltpf_gain_index][k];
  }
  buf[n] = ( x[m-M] - 0.95 * gain * s2 ) - ( y[m-M] - gain * s1 );
}
for ( i = 0; i < Lz; i++ )
{
  for ( j = 1; j <= M; j++ )
  {
    buf[M+i] -= a[M] * buf[M+i-j];
  }
}
for ( i = 0; i < Lz/2; i++ )
{
  zir[i] = buf[M+i];
}
alpha = 1;
for ( i = Lz/2; i < Lz; i++ )
{
  zir[i] = buf[M+i]*alpha;
  alpha -= 2/Lz;
}
```

And finally, the function `ltpf_filter` is applied using the filter parameters of the current frame, N=ccfl/8, and changing the line of code:

```
    y[n] = x[n] + gain * s1 - 0.95 * gain * s2;
```

by the following line of code:

```
    y[n] = x[n] + gain * s1 - 0.95 * gain * s2 - zir[n];
```

5.5.14 Tonal component coding

5.5.14.1 Tool description

The tonal component coding (TCC) is a tool for coding of selected high frequency tonal components using an approach based on sinusoidal modelling. Tonal components are represented as sinusoidal trajectories – data vectors with varying amplitude and frequency values. The trajectories are divided into segments and encoded with technique based on discrete cosine transform.

Each individually encoded sinusoidal component is uniquely represented by its parameters: frequency and amplitude, one pair of values per component per each output data frame containing H = 256 samples. The parameters describing one tonal component are linked into so called sinusoidal trajectories. The original sinusoidal trajectories constructed in the encoder may have an arbitrary length. For the purpose of coding, these trajectories are partitioned into segments. The length of the segments into which each trajectory is split are individually adjusted in time for each trajectory. Finally, segments of different trajectories starting within a particular time are grouped into groups of segments (GOS).

Data values within each segment are encoded jointly. All segments of a trajectory can have lengths in the range from TCC_MIN_SEG_LENGTH=GOS_LENGTH to TCC_MAX_SEG_LENGTH = 32 and they are always multiples of 8. So, the possible segment length values are: 8, 16, 24, and 32. During encoding, the lengths of the segments are adjusted by an extrapolation process. Thanks to this the partitioning of the trajectory into segments is synchronized with the endpoints of GOS structure, i.e. each segment always starts and ends at the endpoints of the GOS structure.

Upon decoding, each segment may continue to the next GOS (or even further), as shown in Figure **25**. After decoding, the segmented trajectories are joined together in the trajectory buffer.

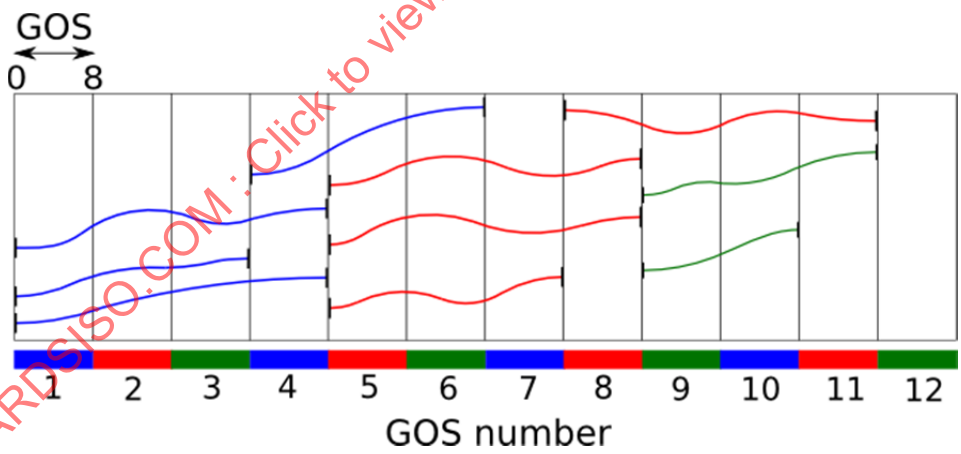


Figure 25 — Partitioning of sinusoidal trajectories into segments and their relation to GOS

The encoding algorithm also has an ability to jointly encode clusters of segments belonging to the harmonic structure of the sound source, i.e. clusters represent the fundamental frequency of each harmonic structure and its integer multiplications. It can exploit the fact that each segment is characterized with a very similar FM and AM modulations.

For stereo and multichannel signals each channel is encoded independently. The TCC tool is optional and may be active only for some of the audio channels. The TCC payload is transmitted in USAC extension element. It is possible to send additional information related to trajectory panning as illustrated in Figure **26** below to further save some bits. However, due to low bitrate overhead introduced by TCC each channel can also be encoded independently as illustrated in Figure 27.

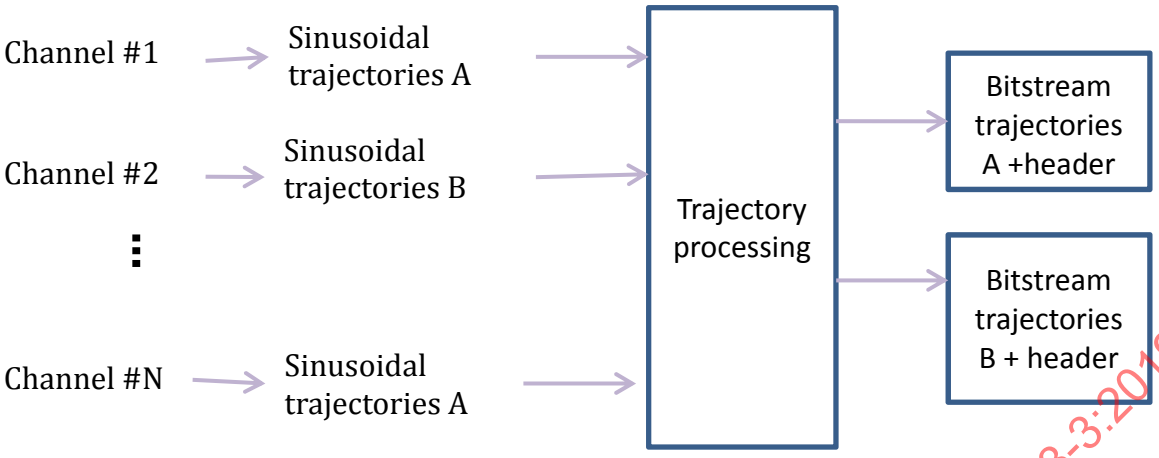


Figure 26 — Sending additional information related to trajectory panning

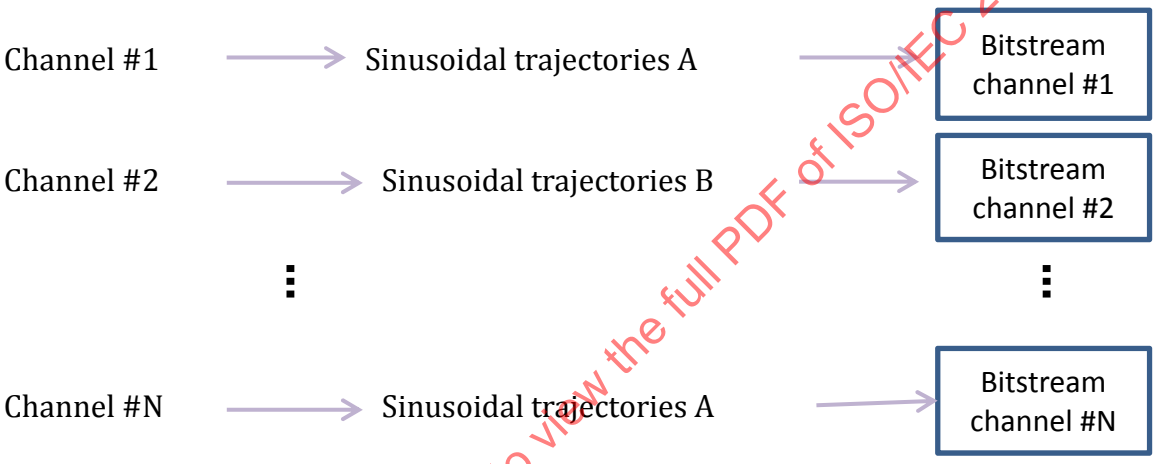


Figure 27 — Independent encoding for each channel

5.5.14.2 Definitions

Help elements:

tccMode[elemIdx] Indicates the use of the tool for a certain group of audio channel elements (SCEs and CPEs):

Table 114 — tccMode

tccMode	Meaning
0	TCC tool not applied
1	One TCC frame for corresponding SCE/CPE
2	Two TCC frames for corresponding CPE
3	Reserved

TccGroupOfSegments () Syntactic element that contains TCC Group Of Segment data.

tccDataPresent Indicates if TCC data are transmitted in current group of segments (GOS).

numSegments Indicates the number of trajectory segments transmitted in current GOS.

isContinued Indicates whether this particular segment will have its continuation in next GOS.

Table 115 — isContinued

isContinued	Meaning
0	Segment will not be continued
1	Segment will be continued

segLength Indicates the length of the currently decoded segment.

Table 116 — segLength

segLength	Trajectory segment length
00	8
01	16
10	24
11	32

amplQuant Quantization step for amplitude coefficients.

Table 117 — amplQuant

amplQuant	Amplitude quantization step in dB
0	0.5
1	1

freqQuant Quantization step for frequency coefficients.

Table 118 — freqQuant

freqQuant	Frequency quantization step in cents
0	2
1	4

huffWord Huffman codeword.

amplTransformCoeffDC Amplitude DCT transform DC coefficient.

freqTransformCoeffDC Frequency DCT transform DC coefficient.

numAmplCoeffs Number of decoded amplitude AC coefficients.

numFreqCoeffs Number of decoded frequency AC coefficients.

amplTransformCoeffAC Array with amplitude DCT transform AC coefficients.

freqTransformCoeffAC Array with frequency DCT transform AC coefficients.

amplTransformIndex Array with amplitude DCT transform AC indices.

freqTransformIndex Array with frequency DCT transform AC indices.

amplOffsetDC	Constant integer added to each decoded amplitude DC coefficient, equal to 32.
freqOffsetDC	Constant integer added to each decoded frequency DC coefficient, equal to 600.
offsetAC	Constant integer added to each decoded amplitude and frequency AC coefficient, equal to 1.
amplSgn	Bit indicating the sign of decoded amplitude AC coefficient, 1 indicates negative value.
freqSgn	Bit indicating the sign of decoded frequency AC coefficient, 1 indicates negative value.
MAX_NUM_TRJ	Maximum number of processed trajectories, equal to 8.
TCC_BUFFER_LENGTH	Length of buffer for storing decoded trajectory amplitude and frequency data, equal to 32.
TCC_SYNTH_LENGTH	Length of buffer for storing synthesized TCC samples, equal to 2048.
TCC_FS	Nominal sampling frequency for TCC sinusoidal trajectory data, equal to 48 000 Hz.

5.5.14.3 Decoding process

5.5.14.3.1 General

Elements of usacExtElementType ID_EXT_ELE_TCC according to *tccMode[elemIndex]* contain TCC data (TCC Groups of Segments - GOS) corresponding to the currently processed channel elements, i.e. SCE or CPE, in the currently processed signal group. The number of transmitted GOS structures for a particular type of channel element is defined in Table 114.

The very first step for decoding of each GOS starts with reading the number of transmitted segments:

$$K = numSegments + 1$$

Next, the decoding of the individual *k-th* segment starts with decoding its length *segLength[k]* and *isContinued[k]* flag according to Table 115 and Table 116.

5.5.14.3.2 Decoding of segment amplitude data

The following procedures are performed for decoding of the *k-th* segment of amplitude data.

1) The amplitude quantization *stepA* step is calculated according to the formula:

$$stepA[k] = \log\left(10^{\frac{amplQuant[k]}{20}}\right)$$

where *amplQuant[k]* is expressed in dB.

2) The *amplTransformCoeffDC[k]* is decoded according to the formula:

$$amplDC[k] = -\left(amplTransformCoeffDC[k] + amplOffset\ DC\right) \times stepA[k]$$

3) The amplitude AC indices *amplIndex[k][j]* are decoded by starting with *j=0* and decoding consecutive *amplTransformIndex[k][j]* Huffman code words and incrementing *j*, until a codeword representing *0* is encountered. The Huffman code words are listed in *huff_idxTab[]* table. The number of decoded indices indicates the number of further transmitted coefficients – *numCoeff[k]*. After decoding, each index should be incremented by o*ffsetAC*.

4) The amplitude AC coefficients are also decoded by means of Huffman code words specified in *huff_acTab[]* table. The AC coefficients are signed values, and so an additional sign bit *amplSgn[k][j]* after each Huffman code word is transmitted, where 1 indicates negative value. Finally, the value of the AC coefficient is decoded according to the formula:

$$amplAC[k][\ j] = amplSgn[k]\ \ [j]\ \left(\left|amplTransf\ \ ormCoeffAC\ \ [k][j]\ + offsetAC\ \right| - 0.25\right) \times stepA[k]$$

Decoded amplitude transform DC and AC coefficients are placed into vector *amplCoeff* of length equal to *segLength[k]*. The *amplDC[k]* coefficient is placed at index 0 and *amplAC[k][j]* coefficients are placed according to decoded *amplIndex[k][j]* indices.

5) The sequence of trajectory amplitude data on a logarithmic scale is reconstructed from the inverse discrete cosine transform and moved into $segAmpl_{log}[k][i]$ buffer according to:

$$segAmpl_{log}[k][i] = \sum_{r=0}^{segLength[k]-1} amplCoeff[k][r]\ w[r] \cos\left(\frac{(2i+1)\pi}{2segLength\ [k]}\right)$$

where:

$$w[r] = \begin{cases} (segLength[k])^{-0.5} & ,\text{for} \quad r = 0 \\ \sqrt{2}(segLength[k])^{-0.5} & ,\text{for} \quad r > 0 \end{cases}$$

The amplitude data is placed in $segAmpl_{log}$ buffer of length equal to *TCC_BUFFER_LENGTH*, beginning with the index *i = 1*. Setting the value with index *i = 0* is explained in subclause 5.5.14.3.5.

6) The linear values of amplitudes in *segAmpl[k][i]* are calculated by:

$$segAmpl[k]\ \ [i]\ = exp\left(segAmpl_{\ log}[k][i]\ \right)$$

5.5.14.3.3 Decoding of segment frequency data

The following procedures are performed for *k-th* segment frequency data decoding.

1) The frequency quantization *stepF[k]* is calculated according to formula:

$$stepF[k] = freqQuant[k] \times log\left(2^{\frac{1}{1200}}\right)$$

where *freqQuant[k]* is expressed in cents.

2) The *freqTransformCoeffDC[k]* is decoded according to formula:

$$freqDC[k]\ \ = -\left(freqTransf\ \ ormCoeffDC\ \ [k]\ + freqOffset\ \ DC\ \right) \times stepF[k]$$

3) Decoding process of frequency AC indices is the same as for amplitude AC indices. The resulting data vector is *freqIndex[k][j].*

4) Decoding process of frequency AC coefficients is the same as for amplitude AC coefficients. The resulting data vector is *freqAC[k][j]*.

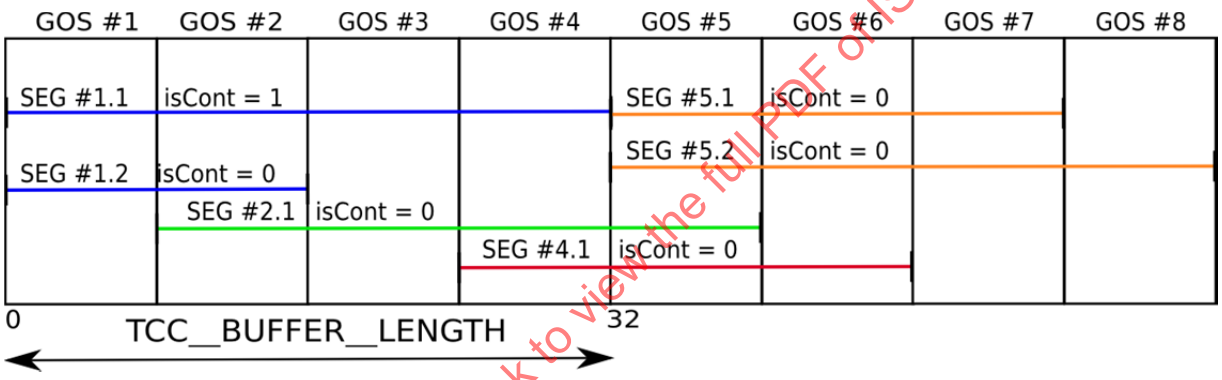
5) Decoded frequency transform DC and AC coefficients are placed into vector *freqCoeff* of length equal to *segLength[k]*. The *freqDC[k]* coefficient is placed in position *j=0* and *freqAC[k][j]* coefficients are placed according to decoded *freqIndex[k][j]* indices.

6. The reconstruction of the sequence of trajectory frequency data in logarithmic scale and further transformation to linear scale is performed in the same manner as for amplitude data. The resulting vector is *segFreq[k][i]*. The linear values of frequency data are stored in the normalized frequency range from 0.07 – 0.5. In order to obtain proper frequency values in Hz, they shall be multiplied by *TCC_FS*:

$$segFreq_{Hz}[k][i] = segFreq[k][i] \times TCC\_FS$$

5.5.14.3.4 Ordering and linking of trajectory segments

The original sinusoidal trajectories built in the encoder are partitioned into an arbitrary number of segments. The length of currently processed segment *segLength[k]* and continuation flag *isContinued[k]* is used to determine when (i.e. in which of the following GOS) the continuation segment will be received. Linking of segments relies on the particular order the trajectories are transmitted. The order of decoding and linking segments is presented and explained in Figure 28.



NOTE Segments decoded within one GOS are marked with the same colour. Each segment is marked with a number (e.g. SEG #5.1) which determines the order of decoding (i.e. order of receiving the segment data from bitstream in given GOS). In this example SEG #1.1 has length of 32 data points and is marked to be continued (isCont = 1). Therefore, SEG #1.1 is going to be continued in GOS #5, where there are two new segments received (SEG #5.1 and SEG #5.2). Decoding order of these segments indicates that the continuation for SEG #1.1 is SEG #5.1.

Figure 28 — Scheme of linking trajectory segments

5.5.14.3.5 Synthesis of decoded trajectories

The received representation of trajectory segments is temporarily stored in data buffers *segAmpl[k][i]* and *segFreq[k][i]*, where *k* represents the index of segment not greater than *MAX_NUM_TRJ = 8*, and *i* represents the trajectory data index within a segment, *0<= i < TCC_BUFFER_LENGTH*. The index *i=0* of buffers *segAmpl* and *segFreq* is filled with data depending on the one of two possible scenarios for further processing of particular segments.

1) The received segment is starting a new trajectory, then the *i*=0 index amplitude and frequency data are provided by simple extrapolation process:

   segFreq[k][0] = segFreq[k][1],

   segAmpl[k][0] = 0.

2) The received segment is recognized as a continuation for the segment processed in the previously received GOS structure, then the *i*=0 index amplitude and frequency data are copy of the last data points from the segment being continued.

The output signal is synthesized from sinusoidal trajectory data stored in the synthesis region of *segAmpl[k][l]* and *segFreq[k][l]*, where each column corresponds to one synthesis frame and *l=0, 1,...,8*. For the purpose of synthesis, these data are to be interpolated on a sample basis, taking into account the synthesis frame length H = 256. The samples of the output signal are calculated according to

$$y_{TCC}[n] = \sum_{k=1}^{K[n]} A_k[n] \cos\left(\varphi_k[n]\right)$$

where

n = *0... TCC_SYNTH_LENGTH-1*;

K[n] denotes the number of currently active trajectories, i.e. the number of rows synthesis region of *segAmpl[k][l]* and *segFreq[k][l]* which have valid data in the frame *l = floor(n/H)* and *l* = floor(*n*/H)+1;

$A_k[n]$ denotes the interpolated instantaneous amplitude of *k*-th partial;

$\varphi_k[n]$ denotes the interpolated instantaneous phase of *k*-th partial.

The instantaneous phase $\varphi_k[n]$ is calculated from the instantaneous frequency $F_k[n]$ according to:

$$\varphi_k[n] = \varphi_k[0] + 2\pi \sum_{m=0}^{n} F_k[m]$$

The initial value of phase $\varphi_k[0]$ is not transmitted and should be stored between consecutive buffers, so that the evolution of phase is continuous. For this purpose the final value of $\varphi_k[TCC\_SYNTH\_LENGTH-1]$ is written to a vector *segPhase[k]*. This value is used as $\varphi_k[0]$ during the synthesis in the next buffer. At the beginning of each new trajectory, $\varphi_k[0] = 0$ is set.

The instantaneous parameters $A_k[n]$ and $F_k[n]$ are interpolated on a sample basis from trajectory data stored in trajectory buffers *segAmpl[k][h]* and *segFreq[k][h]*. The values between two consecutive (*h* and *h+1*) trajectory nodes are calculated by linear interpolation:

$$A_k[n] = segAmpl[k][h] + \left(segAmpl[k][h+1] - segAmpl[k][h]\right) \times \frac{n \bmod H}{H}$$

$$F_k[n] = segFreq[k][h] + \left(segFreq[k][h+1] - segFreq[k][h]\right) \times \frac{n \bmod H}{H}$$

where

$$h = \left\lfloor \frac{n}{H} \right\rfloor$$

$$n = 0,1,2,\ldots,H-1$$

In order to reduce the computational complexity, the *cos* function should be calculated according to the Taylor series expansion. This process is described in the function `get_taylor_cos(phase,N)` as follows.

```
get_taylor_cos(phase,N){
   sign = 1;
   /* convert phase if phase > PI/2 */
```

```
    phase = abs(phase);
    if (phase > (PI/2)) {
        sign = -1;
        phase = PI - phase;
    }
    output = 1;
    phase2 = phase * phase;
    phase_pow = phase2;
    for(i = 1; i<=N; i++){
        output += (phase_pow * inv_factorial_tab[i-1]) ;
        phase_pow *= phase2;
    }
    return sign*output;
}
```

The input phase values should be kept in the `[-PI,PI]` range and *N* is equal to 3. The consecutive inverted factorial values are stored in the following table:

```
inv_factorial_tab[] = {-5.0000000000e-01, 4.1666666667e-02, -1.3888888889e-03}
```

Once the group of TCC_SYNTH_LENGTH samples is synthesized, the content of segAmpl[k][l] and segFreq[k][l] is shifted by 8 trajectory data points and updated with new data from incoming GOS. The synthesized samples are passed to the output buffer, where the data is mixed with the contents produced by the core decoder with appropriate scaling to the output data range through multiplication by 2^{15}. The TCC tool does not introduce any delay during decoding process, so there should be a proper buffer management performed, depending on the current core decoder configuration (delays introduced by SBR and/or MPS/MPS212 decoding).

5.5.14.3.6 Output signal domain switching

Tonal component tool always generates signal in the time domain. If the core decoder output signal representation is in QMF domain, an additional QMF analysis of the TCC output signal should be performed according to ISO/IEC 14496-3:2009, subclause 4.B.18.2. Another option can be direct synthesis of sinusoidal partials to the frequency domain (e.g., MDCT/QMF).

5.5.14.3.7 Huffman tables for AC indices

The following Huffman table `huff_idxTab[]` shall be used for decoding the DCT AC indices:

Table 119 — Huffman table for decoding the DCT AC indices

```
huff_idxTab[] =
{
    /* index, length/bits, deccode, bincode  */
    {       0,       1,         0},  //                0
    {       1,       3,         6},  //              110
    {       2,       3,         7},  //              111
    {       3,       4,         9},  //             1001
    {       4,       4,        11},  //             1011
    {       5,       5,        17},  //            10001
    {       6,       6,        32},  //           100000
    {       7,       6,        40},  //           101000
    {       8,       6,        42},  //           101010
    {       9,       7,        67},  //          1000011
    {      10,       7,        83},  //          1010011
    {      11,       8,       133},  //         10000101
    {      12,       8,       132},  //         10000100
```

```
    {       13,       8,       165},  //         10100101
    {       14,       8,       173},  //         10101101
    {       15,       8,       175},  //         10101111
    {       16,       9,       329},  //        101001001
    {       17,       9,       344},  //        101011000
    {       18,       9,       348},  //        101011100
    {       19,      10,       656},  //       1010010000
    {       20,      10,       698},  //       1010111010
    {       21,      10,       699},  //       1010111011
    {       22,      11,      1380},  //      10101100100
    {       23,      11,      1382},  //      10101100110
    {       24,      11,      1383},  //      10101100111
    {       25,      12,      2628},  //     101001000100
    {       26,      12,      2763},  //     101011001011
    {       27,      12,      2629},  //     101001000101
    {       28,      12,      2631},  //     101001000111
    {       29,      13,      5525},  //    1010110010101
    {       30,      12,      2630},  //     101001000110
    {       31,      13,      5524},  //    1010110010100
};
```

5.5.14.3.8 Huffman tables for AC coefficients

The following Huffman table *huff_acTab[]* shall be used for decoding the DCT AC values. Before further processing and dequantization, the decoded AC values need to be increased by adding the *offsetAC* value.

The encoding of AC coefficients with absolute value higher than 49 uses the escape code, which is the last codeword in the table. After the escape code is put in the bitstream, the actual AC value is transmitted with 7 bit unsigned integer.

Table 120 — Huffman table for decoding the DCT AC values

```
huff_acTab[] =
{
    /* index, length/bits, deccode, bincode  */
    {       0,          6,        31},  //            011111
    {       1,          3,         5},  //               101
    {       2,          3,         1},  //               001
    {       3,          3,         2},  //               010
    {       4,          3,         4},  //               100
    {       5,          3,         7},  //               111
    {       6,          4,         6},  //              0110
    {       7,          4,        13},  //              1101
    {       8,          5,         2},  //             00010
    {       9,          5,        14},  //             01110
    {      10,          6,         0},  //            000000
    {      11,          6,         2},  //            000010
    {      12,          6,         7},  //            000111
    {      13,          6,        30},  //            011110
    {      14,          6,        50},  //            110010
    {      15,          7,         2},  //           0000010
    {      16,          7,         6},  //           0000110
    {      17,          7,        96},  //           1100000
    {      18,          7,        98},  //           1100010
    {      19,          7,        99},  //           1100011
    {      20,          8,         6},  //          00000110
```

```
    {       21,       8,        27},  //          00011011
    {       22,       8,         7},  //          00000111
    {       23,       8,        15},  //          00001111
    {       24,       8,        26},  //          00011010
    {       25,       8,       206},  //          11001110
    {       26,       9,        50},  //         000110010
    {       27,       9,        49},  //         000110001
    {       28,       9,        28},  //         000011100
    {       29,       9,        48},  //         000110000
    {       30,       9,       390},  //         110000110
    {       31,       9,       389},  //         110000101
    {       32,       9,        51},  //         000110011
    {       33,      10,        59},  //        0000111011
    {       34,      10,       783},  //        1100001111
    {       35,       9,       408},  //         110011000
    {       36,      10,       777},  //        1100001001
    {       37,      10,        58},  //        0000111010
    {       38,      10,       782},  //        1100001110
    {       39,       8,       205},  //          11001101
    {       40,       9,       415},  //         110011111
    {       41,      10,       829},  //        1100111101
    {       42,      10,       819},  //        1100110011
    {       43,      10,       828},  //        1100111100
    {       44,      11,      1553},  //       11000010001
    {       45,      11,      1637},  //       11001100101
    {       46,      12,      3105},  //      110000100001
    {       47,      14,     12419},  //   11000010000011
    {       48,      11,      1636},  //       11001100100
    {       49,      14,     12418},  //   11000010000010
    {       50,      13,      6208},  //    1100001000000
};
```

5.5.15 Internal channel on MPS212 for low-complexity format conversion

5.5.15.1 General

For the stereo reproduction layouts, internal channel is chosen to reduce the required complexity by removing redundant processing of upmixing by MPS212 and downmixing by format conversion. When a CPE has two output channels signal whose mix matrix $M_{Mix}(i,j)$ equals one, both the decorrelation and residual processing blocks are switched off by setting $ICC^{l,m}$=1 as introduced in subclause 5.5.4. In this case, the stereo output signal of MPEG Surround 212 upmixer has no phase difference but implementation of the MPEG Surround upmixing as in subclause 5.5.4 results in unnecessary complexity with an increased number of input channels in following format conversion as shown in Figure **29**. When the reproduction layout is stereo, internal channel shall be chosen to reduce required complexity in MPEG-H 3D audio decoding process.

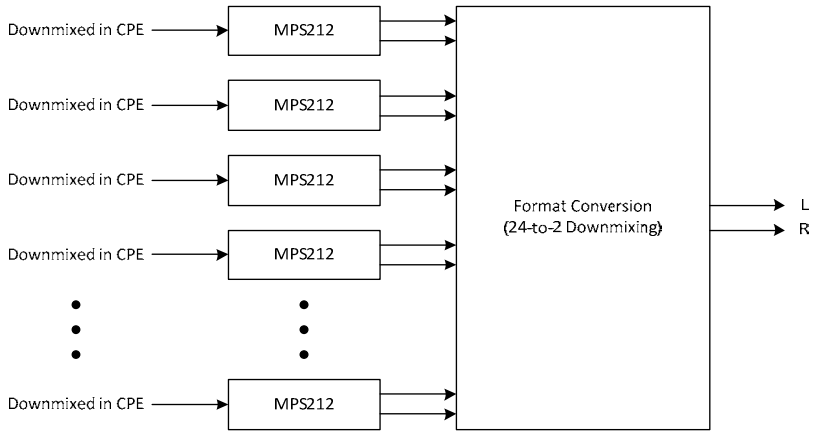


Figure 29 — Redundant decoding structure of MPS212 and format conversion to stereo which results in 24 input channels for 22.2-to-stereo format conversion

Internal channel is defined as an intermediate imaginary channel which corresponds to the input of the format converter. As shown in Figure 30, each internal channel processing block generates an internal channel signal using MPS212 payloads and rendering parameters of EQ and gain value defined in format converter rules table corresponding to the output channel of MPS212 block.

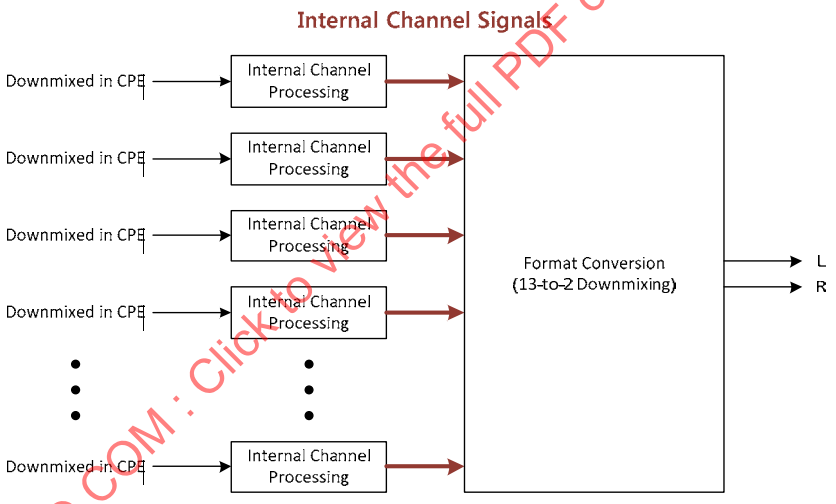


Figure 30 — Decoding process of internal channel and format conversion to stereo which results in 13 input channels for 22.2-to-stereo format conversion

In format conversion, the internal channel signal uses additional converter rules in generic format converter with the gain value of 1.0 and EQ index of 0 as shown in Table 121 because the internal channel signal is created with the consideration of the gain and EQ values for the format conversion.

Table 121 — Additional converter rules

Source	Destination	Gain	EQ index
CH_I_CNTR	CH_M_L030, CH_M_R030	1.0	0 (off)
CH_I_LFE	CH_M_L030, CH_M_R030	1.0	0 (off)
CH_I_LEFT	CH_M_L030	1.0	0 (off)
CH_I_RIGHT	CH_M_R030	1.0	0 (off)

The internal channel shall be implemented providing two types of processing; pre-processing at the MPEG-H 3D audio encoder and post-processing at the MPEG-H 3D audio decoder, depending on the

configuration parameters. As QCE is another form of a pair of CPEs, internal channel for QCE shall be implemented as defined in subclauses 5.5.15.4.4 and 5.5.15.4.5.

5.5.15.2 Definitions

Internal channel — An intermediate imaginary channel with the consideration of stereo output in the format conversion replacing redundant upmixing in MPS212 and downmixing in format converter.

Internal channel signal — A mono signal to be mixed in the format converter to provide the stereo signal. This signal is generated using internal channel gain.

Internal channel processing — A processing block that creates internal channel signal based on the MPS212 decoding block.

Internal channel gain — A gain calculated from CLD value and format conversion parameters of EQ and gain in the Internal channel processing.

Internal channel group — One of the four values CH_I_LEFT, CH_I_RIGHT, CH_I_CNTR, and CH_I_LFE, depending on the core codec output channel position as defined in Table 122. Note that the internal channel group is correspondent to the additional converter rules defined in Table 121.

Table 122 — Groups of internal channel

Group	Core codec output channels	Panning (L,R)
CH_I_LFE	CH_LFE1, CH_LFE2, CH_LFE3	(0.707, 0.707)
CH_I_CNTR	CH_M_000, CH_L_000, CH_U_000, CH_T_000, CH_M_180, CH_U_180	(0.707, 0.707)
CH_I_LEFT	CH_M_L022, CH_M_L030, CH_M_L045, CH_M_L060, CH_M_L090, CH_M_L110, CH_M_L135, CH_M_L150, CH_L_L045, CH_U_L045, CH_U_L030, CH_U_L045, CH_U_L090, CH_U_L110, CH_U_L135, CH_M_LSCR, CH_M_LSCH	(1, 0)
CH_I_RIGHT	CH_M_R022, CH_M_R030, CH_M_R045, CH_M_R060, CH_M_R090, CH_M_R110, CH_M_R135, CH_M_R150, CH_L_R045, CH_U_R045, CH_U_R030, CH_U_R045, CH_U_R090, CH_U_R110, CH_U_R135, CH_M_RSCR, CH_M_RSCH	(1, 0)

5.5.15.3 Variable definitions

For a CPE encoded by MPS212:

cplx_out_dmx [ ] — Donwmix signal in a CPE after complex prediction stereo decoding.

cplx_out_dmx_preICG [ ] — ICG Pre-applied mono signal in hybrid QMF domain by decoding complex prediction stereo decoding and hybrid QMF analysis filterbank.

cplx_out_dmx_postICG [ ] — ICG Post-applied mono signal in hybrid QMF domain by decoding complex prediction stereo decoding and internal channel processing.

cplx_out_dmx_ICG [] — Fullband internal channel signal in hybrid QMF domain.

For a QCE as a pair of CPE encoded by MPS212:

cplx_out_dmx_L[ ]	First channel of first CPE after complex prediction stereo decoding.
cplx_out_dmx_R[ ]	Second channel of first CPE after complex prediction stereo decoding.
cplx_out_dmx_L_preICG [ ]	First ICG pre-applied internal channel signal in hybrid QMF domain.
cplx_out_dmx_R_preICG []	Second ICG pre-applied internal channel signal in hybrid QMF domain.
cplx_out_dmx_L_postICG [ ]	First ICG post-applied internal channel signal in hybrid QMF domain.
cplx_out_dmx_R_postICG []	Second ICG post-applied internal channel signal in hybrid QMF domain.
cplx_out_dmx_L_ICG_SBR	First fullband decoded internal channel signal with high frequency components generated by SBR with downmixed parameters as defined in subclauses 5.5.15.4.5 and 5.5.15.4.6 for the 22.2-to-2 format conversion.
cplx_out_dmx_R_ICG_SBR	Second fullband decoded internal channel signal with high frequency components generated by SBR with downmixed parameters as defined in subclauses 5.5.15.4.5 and 5.5.15.4.6 for the 22.2-to-2 format conversion.

5.5.15.4 Decoding process

When an element of immersive input signal is encoded using CPE or QCE with MPS212 and the output layout is stereo, an internal channel signal shall be generated in the core codec decoding so that the following format conversion uses a reduced number of input channels for efficient covariance analysis except for the CPEs for which **ICinCPE** equals zero. The internal channel processing is simply achieved by multiplying the internal channel gain, calculated from CLD and format conversion parameters, to the decoded mono signal.

5.5.15.4.1 Internal channel calculation conditions and applying ICG

The **ICinCPE**[n] describes whether internal channel processing for the n^{th} CPE is possible. When two output channels in the CPE are in the same output internal channel group as defined in Table 122, **ICinCPE**[n] shall be encoded to 1 at the encoder. For example, **ICinCPE**[n] shall be encoded as 1 when a CPE carries CH_M_L060 and CH_T_L045, which results in the internal channel group of CH_I_LEFT, and **ICinCPE**[n] shall be 0 when a CPE carries CH_M_L060 and CH_M_000, which is not able to result in one internal channel group.

For a QCE as a pair of CPEs, both **ICinCPE**[n] and **ICinCPE**[n+1] shall be encoded to 1 at the encoder for two conditions: (1) if a QCE contains four channels in one group, (CH_M_000, CH_L_000, CH_U_000, and CH_T_000 resulting in CH_I_CNTR), or (2) if a QCE contains two channels in one group and the other two channels in the other group, (CH_M_L060, CH_U_L045, CH_M_R060, and CH_U_R045 resulting in CH_I_LEFT and CH_I_RIGHT). In other cases, both **ICinCPE**[n] and **ICinCPE**[n+1] shall be 0.

While the internal channel gain can be applied at the decoder, applying the gain at the encoder will efficiently reduce the required complexity in the decoder. The **ICGPreAppliedCPE**[n] in ICGConfig indicates whether the internal channel gain for the n^{th} CPE is applied in the encoder side or not. If it is true, internal channel processing block bypasses the downmix signal cplx_out_dmx for the stereo reproduction. If it is false, internal channel processing shall apply the internal channel gain to the cplx_out_dmx at the decoder. Note that **ICGPreAppliedCPE**[n] shall be encoded as 0 if **ICinCPE**[n] is 0 because the calculation of the internal channel gain for the CPE or QCE is not possible. For a QCE as a

pair of CPEs, both **ICGPreAppliedCPE**[n] and **ICGPreAppliedCPE**[n+1] shall have the same value. For convenience, different decoding scenarios are explained in different subclauses as shown in Table 123.

Table 123 — Decoding scenario and explained subclause

Reproduction layout	Element	Order of MPS and SBR	Subclause
Stereo	CPE	An MPS after mono SBR	5.5.15.4.2
Stereo	CPE	An MPS before Stereo SBR	5.5.15.4.3
Stereo	QCE	Two MPS before two Stereo SBR	5.5.15.4.4
Non-stereo	CPE/QCE	Independent of the order	5.5.15.4.5

5.5.15.4.2 Internal channel processing for decoding MPS212 after mono SBR for stereo reproduction

As shown in Figure 31, if the **ICGinCPE**[n] is false, the element shall be decoded as defined in subclauses 5.5.1 to 5.5.6. If it is true, after creating downmixed signal cplx_out_dmx by stereo decoding (ISO/IEC 23003-3:2012, subclause 7.7) and SBR defined in ISO/IEC 23003-3 if required, internal channel processing creates the ICG post-applied downmixed signal cplx_out_dmx_postICG in hybrid QMF domain when the **ICGPreAppliedCPE**[n] is false. If **ICGPreAppliedCPE**[n] equals 1, cplx_out_dmx is analysed so that the ICG pre-applied downmixed signal cplx_out_dmx_preICG is in the hybrid QMF domain. In the internal channel processing, the dequantized linear CLD value for the CPE is calculated according to subclause 5.5.4 and the internal channel gain calculated by the equation given below is multiplied by the cplx_out_dmx to create cplx_out_dmx_postICG.

$$G_{ICH}^{l,m} = \sqrt{\left(c_{left}^{l,m} \times G_{left} \times G_{EQ,left}^{m}\right)^2 + \left(c_{right}^{l,m} \times G_{right} \times G_{EQ,right}^{m}\right)^2}$$

where

$c_{left}^{l,m}$ and $c_{right}^{l,m}$ dequantized linear CLD value of l^{st} time slot and m^{th} hybrid QMF band for the CPE

G_{left} and G_{right} gain column value of Table 164 for the output channels corresponding to the MPS212

$G_{EQ,left}^{m}$ and $G_{EQ,right}^{m}$ gain of m^{th} band in the EQ defined in Table 164 for the output channels corresponding to the MPS212

As a result, either cplx_out_dmx_preICG or dplx_out_dmx_postICG in hybrid QMF domain is used as the internal channel signal cplx_out_dmx_ICG.

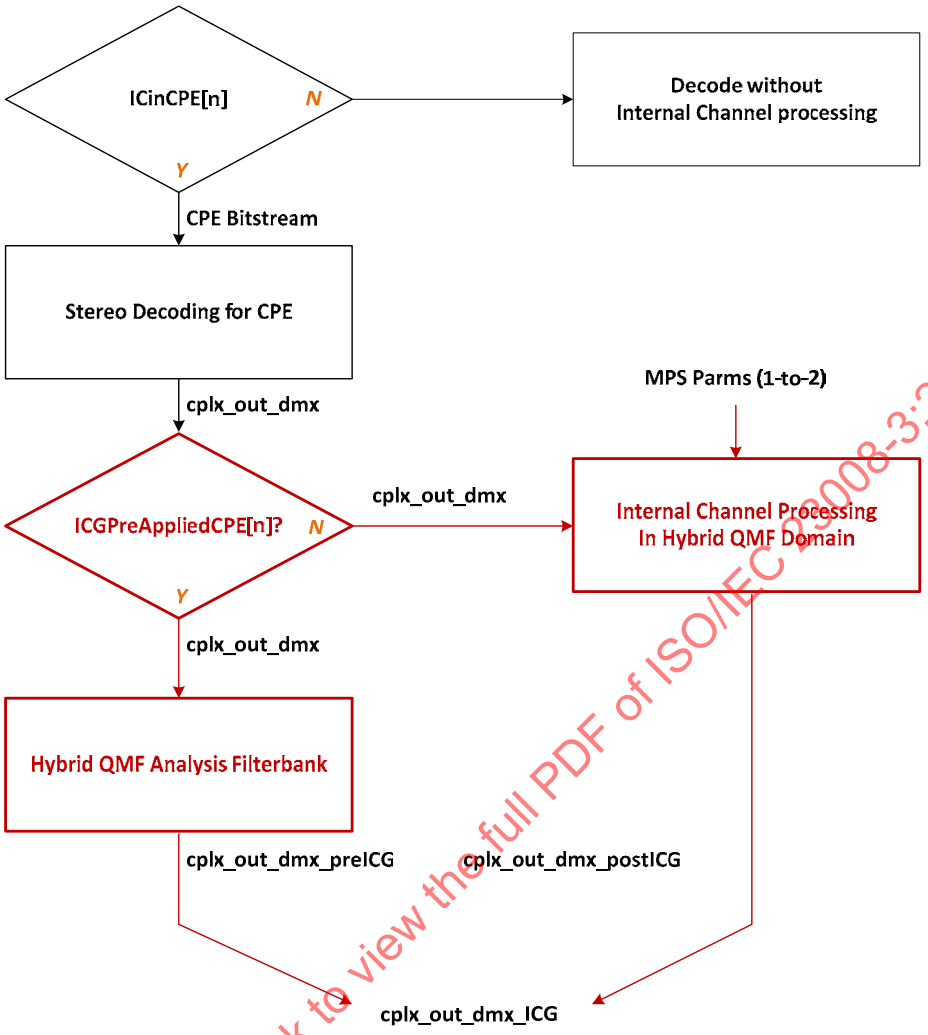


Figure 31 — Internal channel processing for decoding CPE with MPS212 after mono SBR

5.5.15.4.3 Internal channel processing for decoding MPS212 with stereo SBR for stereo reproduction

In this operation configuration, the ICG applied signal of either cplx_out_dmx_preICG or cplx_out_dmx_postICG, which is calculated by multiplying internal channel gain calculated using equation given in subclause 5.5.15.4.2, depending on the configuration by ICGConfig as described in subclause 5.5.15.4.2 is bandlimited because the high frequency components shall be extended by SBR after the internal channel processing. A pair of SBR parameters designed for the bandlimited upmixed stereo signal by MPS212 from the bandlimited mono signal of cplx_out_dmx shall be downmixed into mono SBR parameters in the parametric domain in the SBR parameter downmixer. The SBR parameter downmixer shall include multiplication of the EQ and gain parameters in the format converter. The mono SBR block in Figure 31 generates a full band internal channel signal from the bandlimited internal channel signal extending high frequency components using the downmixed mono SBR parameters.

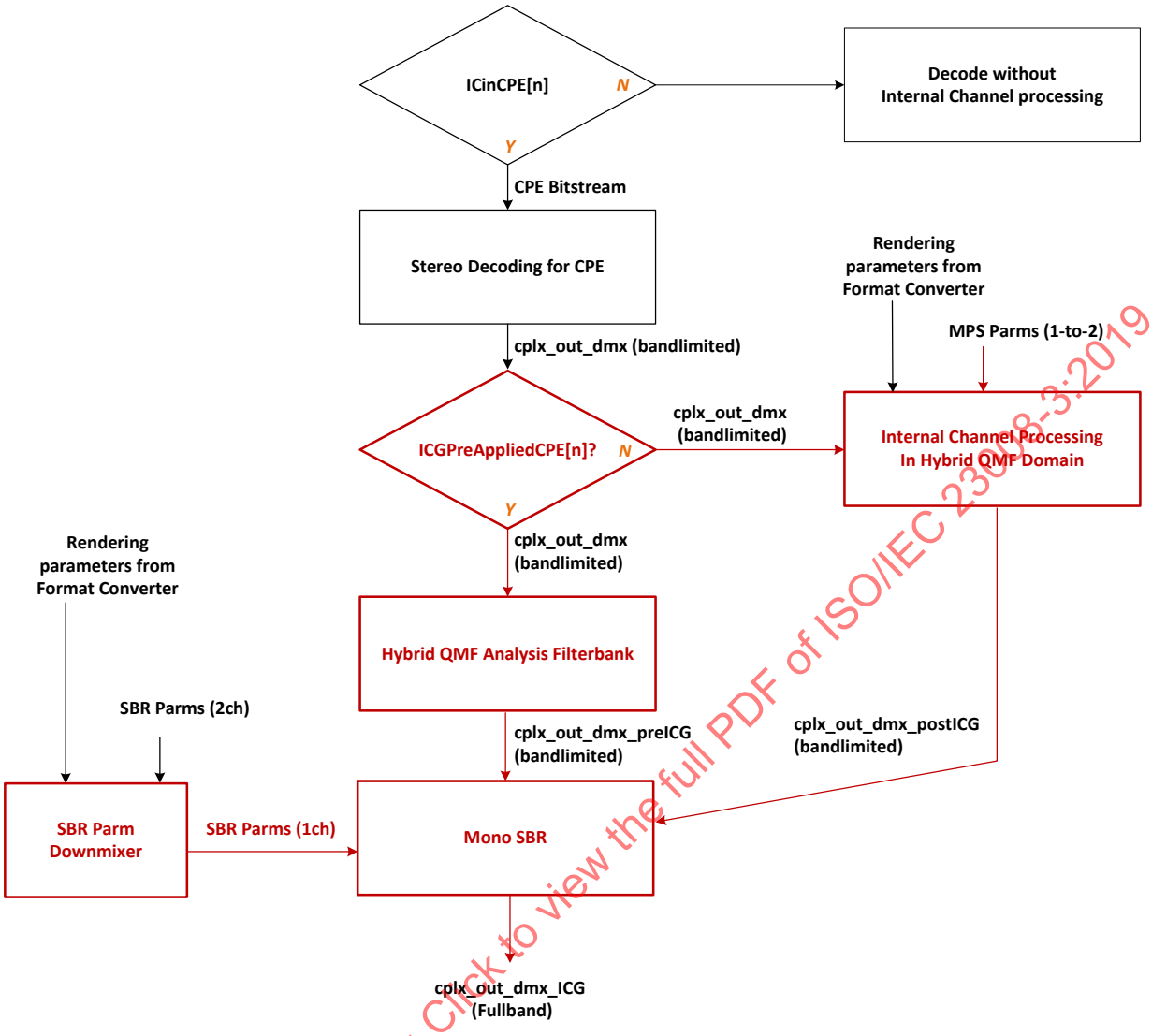


Figure 32 — Internal channel processing for decoding CPE with MPS212 and stereo SBR

5.5.15.4.4 Internal channel processing for decoding a QCE with two stereo SBR for stereo reproduction

When a pair of CPEs, CPE1 and CPE2, has both **ICGinCPE**[n] and **ICGinCPE**[n+1] set to 0, the decoding process shall follow the procedure described in subclause 5.5.2. When a pair of CPEs for a QCE has both **ICGinCPE**[n] and **ICGinCPE**[n+1] set to 1, internal channel signal is created.

If both **ICGPreAppliedCPE**[n] and **ICGPreAppliedCPE**[n+1] are equal to 0, each stereo decoded signal of cplx_dmx_L and cplx_dmx_R are sent to Internal Channel processing block in order to apply bandlimited internal channel signals of cplx_dmx_L_PostICG and cplx_dmx_R_PostICG by using the equation given in subclause 5.5.15.4.2 in the hybrid QMF domain. Then using stereo SBR, two fullband internal channel signals of cplx_dmx_L_ICG_SBR and cplx_dmx_R_ICG_SBR in the hybrid QMF domain shall be generated as shown in Figure 33.

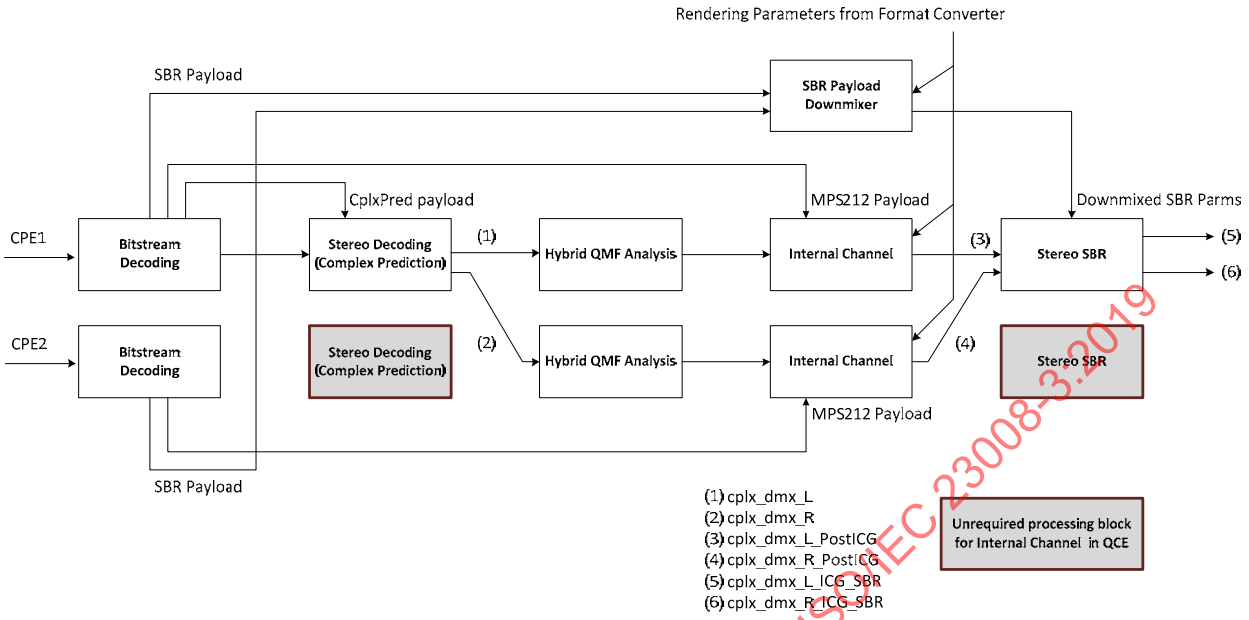


Figure 33 — Internal channel processing or decoding a QCE with two stereo SBR when ICGPreAppliedCPE[n] and ICGPreAppliedCPE[n+1] equal 0

If both **ICGPreAppliedCPE**[n] and **ICGPreAppliedCPE**[n+1] are equal to 1, each stereo decoded signal of cplx_dmx_L and cplx_dmx_R are sent to the stereo SBR block directly because the internal channel gain is applied at the encoder. After the stereo SBR block, the stereo fullband internal channel signals of cplx_dmx_L_ICG and cplx_dmx_R_ICG shall be generated as shown in Figure 34.

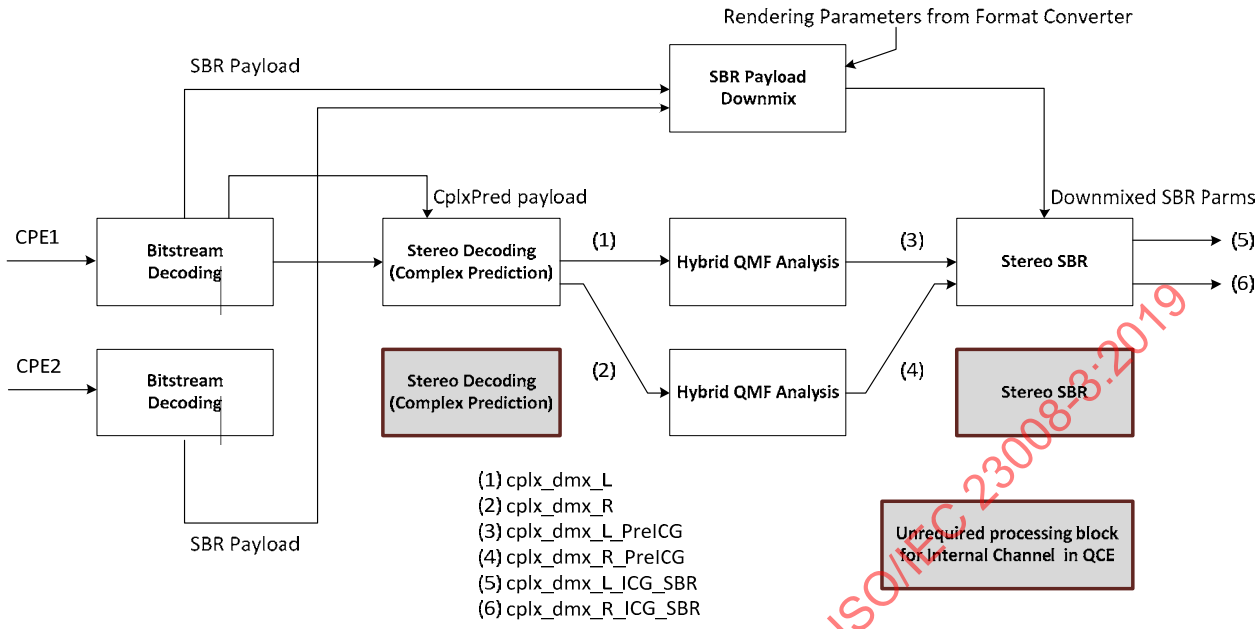


Figure 34 — Internal channel processing for decoding a QCE with two stereo SBR when ICGPreAppliedCPE[n] and ICGPreAppliedCPE[n+1] equal 1

5.5.15.4.5 Internal channel processing for decoding MPS212 after mono SBR for non-stereo reproduction

For the non-stereo layout reproduction, inverse ICG calculation is performed using MPS parameters and format conversion parameters as given in the equation below. If **ICGPreAppliedCPE**[n] is **TRUE**, n^{th} cplx_dmx shall be multiplied by the inverse ICG before MPS block and the rest of the decoding process shall follow this document.

$$IG_{ICH}^{l,m} = \frac{1}{\sqrt{\left(c_{left}^{l,m} \times G_{left} \times G_{EQ,left}^{m}\right)^2 + \left(c_{right}^{l,m} \times G_{right} \times G_{EQ,right}^{m}\right)^2}}$$

where

$c_{left}^{l,m}$ and $c_{right}^{l,m}$ dequantized linear CLD value for the CPE

G_{left} and G_{right} gain column value of Table 164 for the output channels corresponding to the MPS212

$G_{EQ,left}^{m}$ and $G_{EQ,right}^{m}$ gain of m^{th} band in the EQ defined in Table 164 for the output channels corresponding to the MPS212

5.5.15.4.6 Parameter downmixing for stereo SBR for internal channel

5.5.15.4.6.1 General

When the internal channel processing block is used or internal channel gain is pre-processed at the encoder and the output layout is stereo, before the SBR block as introduced in subclauses 5.5.15.4.3 and 5.5.15.4.4, the bandlimited internal channel signals for a CPE/QCE are generated instead of MPS upmixed stereo/quad channel signals for the CPE/QCE. While the SBR payload is encoded for the MPS upmixed stereo/quad channel signal, it shall be downmixed in the parametric domain. For that, the gain and EQ in format converter shall be multiplied during the parameter downmixing for stereo SBR.

5.5.15.4.6.2 Inverse filtering

The inverse filtering mode is selected by taking the maximum value from stereo SBR parameters for each noise floor band.

$$\text{for } \left(\text{i} = 0; \text{i} < N_Q; i++\right)$$
$$bs\_invf\_mode_{Downmixed}(i) = MAX(bs\_invf\_mode_{ch1}(i), bs\_invf\_mode_{ch2}(i))$$

$$\binom{ch1}{ch2} = \begin{cases} \binom{Left\ of\ CPE1}{Left\ of\ CPE2} & \text{in case of Cplx\_out\_dmx\_L} \\ \binom{Right\ of\ CPE1}{Right\ of\ CPE2} & \text{in case of Cplx\_out\_dmx\_R} \end{cases}'$$

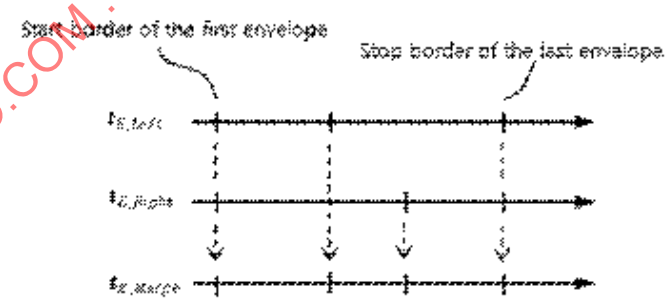
5.5.15.4.6.3 Additional harmonics

The additional harmonics are the union of the additional sinusoids present in stereo channels as shown below.

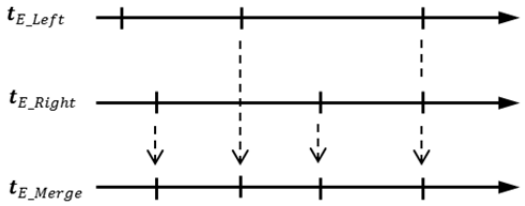
$$\text{for } \left(\text{i} = 0; \text{i} < N_{High}; i++\right)$$
$$bs\_add\_harmonic_{Downmixed}(i) = OR(bs\_add\_harmonic_{ch1}(i), bs\_add\_harmonic_{ch2}(i))$$

5.5.15.4.6.4 Envelope time borders

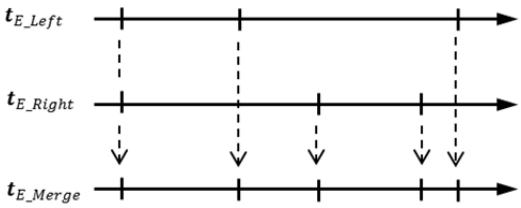
The time envelope grid t_{E_Merged} for internal channel SBR is generated from stereo SBR time grids to divide into the smallest pieces with the highest resolution. The start border value for t_{E_Merged} is set to the largest of the start border values for the stereo channels. Envelope between time grid 0 and start border is already processed in the previous frame. The stop border of the last envelope is selected by taking the maximum value from stop borders of the last envelopes for both channels. Start/Stop borders between the first and the last envelope are determined to provide the maximum segment resolution by taking intersection of time borders of both channels as shown in Figure 35. If there are more than 5 envelopes, the number of envelopes shall be reduced by starting from the end of t_{E_Merged} and searching towards the beginning of t_{E_Merged} for the length of envelope smaller than 4 and removing the start border of that envelope. This continues until there are 5 envelopes left.



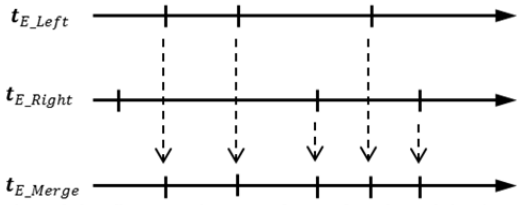
(a) Case 1: start border of the first envelope and stop of the last envelope are same



(b) Case 2 : start border of the first envelope are different and stop border of the last envelope are same



(c) Case 3 : start border of the first envelope are same and stop border of the last envelope are different



(d) Case 4 : start border of the first envelope and stop border of the last envelope are all different

Figure 35 — Merging of envelope time borders

5.5.15.4.6.5 Noise time borders

The number of downmixed noise time borders L_{Q_Merged} is determined by taking the larger value between noise time borders of both channels. The first grid and the last grid of merged noise time borders t_{Q_Merged} is determined by taking the first grid and the last grid of envelope time borders t_{E_Merged}. If the number of the noise time borders L_{Q_Merged} is larger than $t_{Q\_Merged}(1)$ is selected as $t_Q(1)$ of the channel whose number of the noise time border L_Q is larger than 1. If both channels have L_Q larger than 1, the minimum value of $\boldsymbol{t}_Q(1)$ is selected as a $\boldsymbol{t}_{Q\_Merged}(1)$.

5.5.15.4.6.6 Envelope data

Frequency resolution $\boldsymbol{r}_{Merged}$ of the merged envelope time borders for each envelope is selected. The maximum value between frequency resolution $\boldsymbol{r}_{ch1}, \boldsymbol{r}_{ch2}$ corresponding to each section of $\boldsymbol{r}_{Merged}$ is selected as shown in Figure 36.

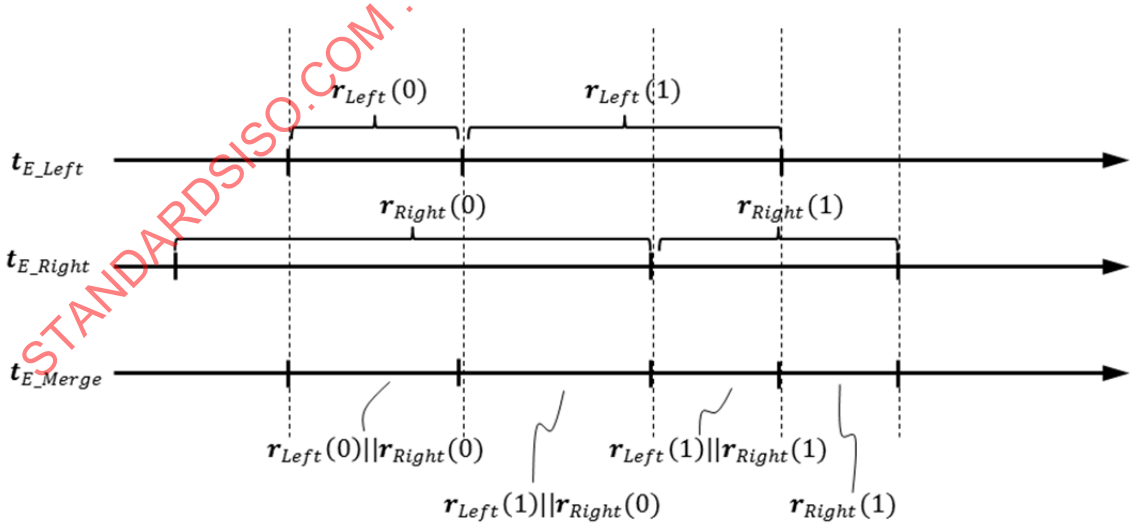


Figure 36 — Merging frequency resolution

The envelope data $\boldsymbol{E}_{Orig\_Merged}$ for every envelope shall be calculated from envelope data $\boldsymbol{E}_{Orig}$ with the considerations of format conversion parameters as shown below.

$$\boldsymbol{E}_{Orig_{Merged}}(\mathrm{k},\mathrm{l}) =$$
$$\boldsymbol{E}_{ch1Orig}\left(g_{ch1}(k),h_{ch1}(l)\right)\times\left(EQ_{ch1}\left(k,h_{ch1}(l)\right)\right)^2 +$$
$$\boldsymbol{E}_{ch2Orig}\left(g_{ch2}(k),h_{ch2}(l)\right)\times\left(EQ_{ch2}(k,h_{ch2}(l)\right)^2$$
$$\text{with } 0 \le \mathrm{k} < \mathbf{n}\left(\boldsymbol{r}_{Merged}(l)\right), 0 \le l < L_{E\_Merged}$$

where

$EQ_{ch1}(k,l) = \frac{\sum m(G_{left}\times G^{m}_{EQ,left})}{\mathbf{F}\left(k+1,\boldsymbol{r}_{Merged}(l)\right)-\mathbf{F}(k,\boldsymbol{r}_{Merged}(l))}, \mathbf{F}(k,\boldsymbol{r}_{Merged}(l)) \le \mathrm{m} < \mathbf{F}(k+1,\boldsymbol{r}_{Merged}(l))$,

$EQ_{ch2}(k,l) = \frac{\sum m(G_{right}\times G^{m}_{EQ,right})}{\mathbf{F}\left(k+1,\boldsymbol{r}_{Merged}(l)\right)-\mathbf{F}(k,\boldsymbol{r}_{Merged}(l))}, \mathbf{F}(k,\boldsymbol{r}_{Merged}(l)) \le \mathrm{m} < \mathbf{F}(k+1,\boldsymbol{r}_{Merged}(l))$,

$h_{ch1}(l)$ is defined by $\boldsymbol{t}_{E\_ch1}(h_{ch1}(l)) \le \boldsymbol{t}_{E\_Merged}(l) < \boldsymbol{t}_{E\_ch1}(h_{ch1}(l)+1)$,

$h_{ch2}(l)$ is defined by $\boldsymbol{t}_{E\_ch2}(h_{ch2}(l)) \le \boldsymbol{t}_{E\_Merged}(l) < \boldsymbol{t}_{E\_ch2}(h_{ch2}(l)+1)$,

$g_{ch1}(k)$ is defined by $\mathrm{F}(g_{ch1}(k),\boldsymbol{r}_{ch1}(h_{ch1}(l))) \le \mathrm{F}(\mathrm{k},\boldsymbol{r}_{Merged}(l)) < \mathrm{F}(g_{ch1}(k)+1,\boldsymbol{r}_{ch1}(h_{ch1}(l)))$,

$g_{ch2}(k)$ is defined by $\mathrm{F}(g_{ch2}(k),\boldsymbol{r}_{ch2}(h_{ch2}(l))) \le \mathrm{F}(\mathrm{k},\boldsymbol{r}_{Merged}(l)) < \mathrm{F}(g_{ch2}(k)+1,\boldsymbol{r}_{ch2}(h_{ch2}(l)))$,

$G_{left}, G_{right}, G^{m}_{EQ,left}$, and $G^{m}_{EQ,right}$ are the rendering parameters from the format converter as introduced in subclause 5.5.15.4.2

5.5.15.4.6.7 Noise floor data

The merged noise floor data $\boldsymbol{Q}_{OrigMerged}$ is determined as the sum of data from both channel according to the function below.

$$\boldsymbol{Q}_{OrigMerged}(k,l) = \boldsymbol{Q}_{Origch1}(k,h_{ch1}(l)) + \boldsymbol{Q}_{Origch2}(k,h_{ch2}(l))$$

$,0 \le \mathrm{k} < N_Q, 0 \le l < L_{Q\_Merged},$

where

$h_{ch1}(l)$ is defined by $\boldsymbol{t}_{Q\_ch1}(h_{ch1}(l)) \le \boldsymbol{t}_{Q\_Merged}(l) < \boldsymbol{t}_{Q\_ch1}(h_{ch1}(l)+1)$ and

$h_{ch2}(l)$ is defined by $\boldsymbol{t}_{Q\_ch2}(h_{ch2}(l)) \le \boldsymbol{t}_{Q\_Merged}(l) < \boldsymbol{t}_{Q\_ch2}(h_{ch2}(l)+1)$.

5.5.15.4.7 Spectral band replication for internal channel

Spectral band replication part is identical to ISO/IEC 23003-3 except for one change. While the SBR in ISO/IEC 23003-3 is defined in the QMF domain, the internal channel processes are in the hybrid QMF domain. For that, the frequency indexes for the whole SBR processes for internal channels are updated from k to k+7. Note that the downmixed SBR parameters described in subclause 5.5.15.4.3 shall be used to generate the fullband internal channel signal for the bandwidth extension of the bandlimited internal channel signal in the cases introduced in subclauses 5.5.15.4.3 and 5.5.15.4.4.

5.5.15.4.8 Interface with the format conversion

As the output of the core codec with internal channel processing is in the hybrid QMF domain, the process defined in subclause 10.3.5.2 shall be discarded. In order to assign each channel of the core coder, the following additional channel assignment and downmix rules shall be used for each internal channel signal as shown in Table 121 and Table 124.

Table 124 — Additional channels definitions for internal channel

Channel	Azimuth [deg]	Elevation [deg]	Azimuth start angle of sector [deg]	Azimuth end angle of sector [deg]	Elevation start angle of sector [deg]	Elevation end angle of sector [deg]	Ch. is LFE	Position is relative
CH_I_CNTR	0	0	0	0	0	0	0	0
CH_I_LFE	0	n/a	n/a	n/a	n/a	n/a	1	0
CH_I_LEFT	30	0	30	30	0	0	0	0
CH_I_RIGHT	-30	0	-30	-30	0	0	0	0

5.5.16 High resolution envelope processing (HREP) tool

5.5.16.1 Tool description

The HREP tool provides improved coding performance for signals that contain densely spaced transient events, such as applause signals as they are an important part of live recordings. Similarly, sound of raindrops or other sounds such as fireworks can show such characteristics. Unfortunately, this class of sounds presents difficulties to existing audio codecs, especially when coded at low bitrates and/or with parametric coding tools.

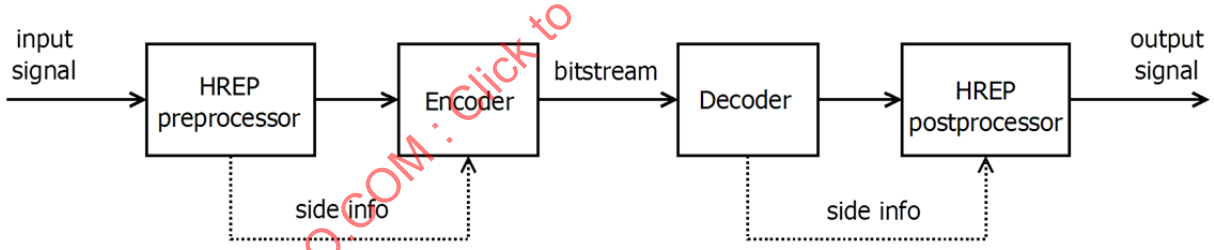


Figure 37 — Overview of signal flow in an HREP equipped codec

Figure 37 depicts the signal flow in an HREP equipped codec. At the encoder, the tool preprocesses the input signal to temporally flatten high frequencies while generating a small amount of side information (1-4 kbps for stereo signals). An exemplary description of an HREP encoder can be found in Annex L. At the decoder, the tool post-processes the output signal to temporally restore the high frequencies, making use of the side information. The benefits of applying HREP are two-fold: HREP relaxes the bitrate demand imposed on the encoder by reducing short time dynamics of the input signal; additionally, HREP ensures proper envelope restoration in the decoder's (up-)mixing stage, which is all the more important if parametric multi-channel coding techniques have been applied within the codec. The HREP tool works for all input channel configurations (mono, stereo, multi-channel including 3D) and also for audio objects.

5.5.16.2 Data and help elements

current_signal_group	The current_signal_group parameter is based on the Signals3d() syntax element and the mpegh3daDecoderConfig() syntax element.
signal_type	The type of the current signal group, used to differentiate between channel signals and object, HOA, and SAOC signals.
signal_count	The number of signals in the current signal group.
channel_layout	In case the current signal group has channel signals, it contains the properties of loudspeakers for each channel, used to identify LFE loudspeakers.
extendedGainRange	Indicates whether the gain indexes use 3 bits (8 values) or 4 bits (16 values), as computed by nBitsGain.
extendedBetaFactorPrecision	Indicates whether the beta factor indexes use 3 bits or 4 bits, as computed by nBitsBeta.
isHREPActive[sig]	Indicates whether the tool is active for the signal on index sig in the current signal group.
lastFFTLine[sig]	The position of the last non-zero line used in the low-pass procedure implemented using FFT.
transitionWidthLines[sig]	The width in lines of the transition region used in the low-pass procedure implemented using FFT.
defaultBetaFactorIdx[sig]	The default beta factor index used to modify the gains in the gain compensation procedure.
outputFrameLength	The equivalent number of samples per frame, using the original sampling frequency, as defined in ISO/IEC 23003-3.
gain_count	The number of gains per signal in one frame.
useRawCoding	Indicates whether the gain indexes are coded raw, using nBitsGain each, or they are coded using arithmetic coding.
gainIdx[pos][sig]	The gain index corresponding to the block on position pos of the signal on position sig in the current signal group. If extendedGainRange = 0, the possible values are in the range {0, …, 7}, and if extendedGainRange = 1, the possible values are in the range {0, …, 15}.
GAIN_INDEX_0dB	The gain index offset corresponding to 0 dB, with a value of 4 being used if extendedGainRange = 0, and with a value of 8 being used if extendedGainRange = 1. The gain indexes are transmitted as unsigned values by adding GAIN_INDEX_0dB to their original signed data ranges.

all_zero	Indicates whether all the gain indexes in one frame for the current signal are having the value GAIN_INDEX_0dB.
useDefaultBetaFactorIdx	Indicates whether the beta factor index for the current signal has the default value specified by defaultBetaFactor[sig].
betaFactorIdx[sig]	The beta factor index used to modify the gains in the gain compensation procedure.

5.5.16.3 Decoding process

5.5.16.3.1 General

In the syntax element *mpegh3daExtElementConfig()* the field *usacExtElementPayloadFrag* shall be zero in the case of an ID_EXT_ELE_HREP element. The HREP tool is applicable to signal groups of any type: SignalGroupTypeChannels, SignalGroupTypeObject, SignalGroupTypeSAOC, or SignalGroupTypeHOA as defined by SignalGroupType[grp] in the Signals3d() syntax element.

The block size and correspondingly the FFT size used is $N = 128$.

The entire processing is performed independently on each signal in the current signal group. Therefore, to simplify notation, the decoding process is described just for one signal having index position sig.

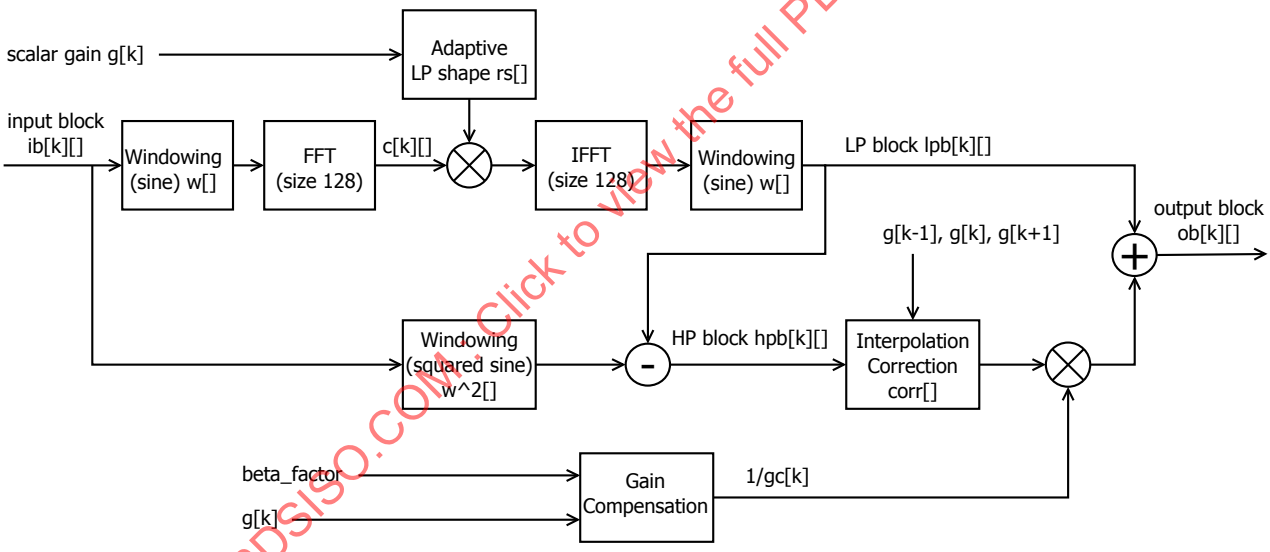


Figure 38 — High resolution envelope processing (HREP) tool decoder

5.5.16.3.2 Decoding of gains using arithmetic coding

The helper function HREP_decode_ac_data(gain_count, signal_count) describes the reading of the gain values into the array gainIdx using the following USAC low-level arithmetic coding functions as defined in ISO/IEC 23003-3. Corresponding encoder helper functions are provided in L.4.

```
arith_decode(*ari_state, cum_freq, cfl),
arith_start_decoding(*ari_state),
arith_done_decoding(*ari_state).
```

Two additional helper functions are introduced,

```
ari_decode_bit_with_prob(*ari_state, count_0, count_total),
```

which decodes one bit with p_0 = count_0/total_count and $p_1 = 1 - p_0$, and

```
ari_decode_bit(*ari_state),
```

which decodes one bit without modelling, with $p_0 = 0.5$ and $p_1 = 0.5$.

```
ari_decode_bit_with_prob(*ari_state, count_0, count_total)
{
    prob_scale = 1 << 14;
    tbl[0] = probScale - (count_0 * prob_scale) / count_total;
    tbl[1] = 0;
    res = arith_decode(ari_state, tbl, 2);
    return res;
}

ari_decode_bit(*ari_state)
{
    prob_scale = 1 << 14;
    tbl[0] = prob_scale >> 1;
    tbl[1] = 0;
    res = arith_decode(ari_state, tbl, 2);
    return res;
}

HREP_decode_ac_data(gain_count, signal_count)
{
    cnt_mask[2] = {1, 1};
    cnt_sign[2] = {1, 1};
    cnt_neg[2] = {1, 1};
    cnt_pos[2] = {1, 1};

    arith_start_decoding(&ari_state);

    for (pos = 0; pos < gain_count; pos++) {
        for (sig = 0; sig < signal_count; sig++) {
            if (!isHREPActive[sig]) {
                continue;
            }
            mask_bit = ari_decode_bit_with_prob(&ari_state, cnt_mask[0],
                                                cnt_mask[0] + cnt_mask[1]);
            cnt_mask[mask_bit]++;

            if (mask_bit) {
                sign_bit = ari_decode_bit_with_prob(&ari_state, cnt_sign[0],
                                                    cnt_sign[0] + cnt_sign[1]);
                cnt_sign[sign_bit] += 2;

                if (sign_bit) {
                    large_bit = ari_decode_bit_with_prob(&ari_state, cnt_neg[0],
                                                         cnt_neg[0] + cnt_neg[1]);
                    cnt_neg[large_bit] += 2;
                    last_bit = ari_decode_bit(&ari_state);
                    gainIdx[pos][sig] = -2 * large_bit - 2 + last_bit;
                } else {
                    large_bit = ari_decode_bit_with_prob(&ari_state, cnt_pos[0],
                                                         cnt_pos[0] + cnt_pos[1]);
                    cnt_pos[large_bit] += 2;
                    if (large_bit) {
                        gainIdx[pos][sig] = 3;
```

```
                    } else {
                        last_bit = ari_decode_bit(&ari_state);
                        gainIdx[pos][sig] = 2 - last_bit;
                    }
                }
            } else {
                gainIdx[pos][sig] = 0;
            }

            if (extendedGainRange) {
                prob_scale = 1 << 14;
                esc_cnt = prob_scale / 5;
                tbl_esc[5] = {prob_scale-esc_cnt, prob_scale-2*esc_cnt,
                              prob_scale-3*esc_cnt, prob_scale-4*esc_cnt, 0};
                sym = gainIdx[pos][sig];
                if (sym <= -4) {
                    esc = arith_decode(ari_state, tbl_esc, 5);
                    sym = -4 - esc;
                } else if (sym >= 3) {
                    esc = arith_decode(ari_state, tbl_esc, 5);
                    sym = 3 + esc;
                }
                gainIdx[pos][sig] = sym;
            }

            gainIdx[pos][sig] += GAIN_INDEX_0dB;
        }
    }

    arith_done_decoding(&ari_state);
}
```

5.5.16.3.3 Decoding of quantized beta factors

The following lookup tables for converting beta factor index betaFactorIdx[*sig*] to beta factor beta_factor should be used, depending on the value of extendedBetaFactorPrecision.

```
tab_beta_factor_dequant_coarse[8] = {
  0.000f, 0.035f, 0.070f, 0.120f, 0.170f, 0.220f, 0.270f, 0.320f
}

tab_beta_factor_dequant_precise[16] = {
  0.000f, 0.035f, 0.070f, 0.095f, 0.120f, 0.145f, 0.170f, 0.195f,
  0.220f, 0.245f, 0.270f, 0.295f, 0.320f, 0.345f, 0.370f, 0.395f
}
```

If extendedBetaFactorPrecision = 0, the conversion is computed as

beta_factor = tab_beta_factor_dequant_coarse[betaFactorIndex[sig]]

If extendedBetaFactorPrecision = 1, the conversion is computed as

beta_factor = tab_beta_factor_dequant_precise[betaFactorIndex[sig]]

5.5.16.3.4 Decoding of quantized gains

One frame is processed as gain_count blocks consisting of N samples each, which are half-overlapping. The scalar gains for each block are derived, based on the value of extendedGainRange.

$g[k] = 2^{\frac{\text{gainIdx}[k][sig]-\text{GAIN\_INDEX\_0dB}}{4}}$, for $0 \le k <$ gain_count

5.5.16.3.5 Computation of the LP part and the HP part

The input signal s is split into blocks of size N, which are half-overlapping, producing input blocks $\text{ib}[k][i] = s[k \cdot \frac{N}{2} + i]$, where k is the block index and i is the sample position in the block k. A window $w[i]$ is applied to $\text{ib}[k]$, in particular the sine window, defined as

$$w[i] = \sin\frac{\pi(i+0.5)}{N}, \text{ for } 0 \le i < N,$$

and after computing an FFT, the complex coefficients $c[k][f]$ are obtained as

$c[k][f] = \text{FFT}(\text{w[i]}\times\text{ib}[k])$, for $0 \le f \le \frac{N}{2}$.

On the encoder side, as further described in L.2, to obtain the low-pass (LP) filtered part, an element-wise multiplication of $c[k]$ with the processing shape $\text{ps}[f]$ may be applied, which consists of the following.

$$\text{ps}[f] = \begin{cases} 1, \text{ for } 0 \le f < \text{lp\_size} \\ 1 - \frac{f - \text{lp\_size}+1}{\text{tr\_size} + 1}, \text{ for lp\_size} \le f < \text{lp\_size} + \text{tr\_size} \\ 0, \text{ for lp\_size} + \text{tr\_size} \le f \le \frac{N}{2} + 1 \end{cases}$$

The $\text{lp\_size} = \text{lastFFTLine}[sig] + 1 - \text{transitionWidthLines}[sig]$ parameter represents the width in FFT lines of the low-pass region, and the $\text{tr\_size} = \text{transitionWidthLines}[sig]$ parameter represents the width in FFT lines of the transition region.

On the decoder side, in order to get perfect reconstruction in the transition region, an adaptive reconstruction shape $\text{rs}[f]$ in the transition region shall be used, instead of the processing shape $\text{ps}[f]$ used at the encoder side, depending on the processing shape $\text{ps}[f]$ and $g[k]$ as

$$\text{rs}[f] = 1 - (1 - \text{ps}[f]) \times \frac{g[k]}{1 + (g[k] - 1) \cdot (1 - \text{ps}[f])}$$

The LP block $\text{lpb}[k]$ is obtained by applying IFFT and windowing again as:

$$\text{lpb}[k][i] = w[i] \times IFFT(\text{rs}[f] \times c[k][f]), \text{ for } 0 \le i < N$$

The high-pass (HP) filtered block $\text{hpb}[k]$ is then obtained by simple subtraction in the time domain as:

$$\text{hpb}[k][i] = \text{in}[k][i] \times w^2[i] - \text{lpb}[k][i], \text{ for } 0 \le i < N$$

5.5.16.3.6 Computation of the interpolation correction

The gains $g[k\text{-}1]$ and $g[k]$ to be applied on the encoder side to blocks on positions k-1 and k, as suggested in L.1, are implicitly interpolated due to the windowing and overlap-add operations. In order to achieve perfect reconstruction in the HP part above the transition region, an interpolation correction factor is needed as:

$$\text{corr}[j] = 1 + \left(\frac{g[k-1]}{g[k]} + \frac{g[k]}{g[k-1]} - 2\right) \cdot w^2[j] \cdot (1 - w^2[j]), \text{ for } 0 \le j < \frac{N}{2}.$$

$$\text{corr}\left[j + \frac{N}{2}\right] = 1 + \left(\frac{g[k]}{g[k+1]} + \frac{g[k+1]}{g[k]} - 2\right) \cdot w^2[j] \cdot (1 - w^2[j]), \text{ for } 0 \le j < \frac{N}{2}.$$

5.5.16.3.7 Computation of the compensated gains

The core encoder and decoder introduce additional attenuation of transient events, which is compensated by adjusting the gains $g[k]$ using the previously computed beta_factor as:

$$gc[k] = (1 + \text{beta\_factor})g[k] - \text{beta\_factor}$$

5.5.16.3.8 Computation of the output signal

Based on $gc[k]$ and corr[i], the value of the output block ob[k] is computed as:

$$\text{ob}[k][i] = \text{lpb}[k][i] + \frac{1}{gc[k]} \cdot \frac{1}{\text{corr}[i]} \cdot \text{hpb}[k][i], \text{ for } 0 \le i < N$$

Finally, the output signal is computed from the output blocks using overlap-add processing as:

$$o\left[k \cdot \frac{N}{2} + j\right] = \text{ob}[k-1]\left[j + \frac{N}{2}\right] + \text{ob}[k][j], \text{ for } 0 \le j < \frac{N}{2}$$

$$o\left[(k+1) \cdot \frac{N}{2} + j\right] = \text{ob}[k]\left[j + \frac{N}{2}\right] + \text{ob}[k+1][j], \text{ for } 0 \le j < \frac{N}{2}$$

5.6 Buffer requirements

5.6.1 Minimum decoder input buffer

The following rules are used to calculate the maximum number of bits in the input buffer for any of the following: the bitstream as a whole, for any given program, or for any given elements of type ID_USAC_SCE (SCE) and ID_USAC_CPE (CPE).

The input buffer size is 6144 bits per SCE plus 12288 bits per CPE ($6144 \cdot NCC$), irrespective of whether these elements are used for channels, objects, SAOC transport channels or HOA transport channels. Both the total buffer and the individual buffer sizes are limited, so that the buffering limit can be calculated for either the entire bitstream payload or the individual audio elements permitting unnecessary bitstream elements to be stripped from the 3D audio bitstream payload to form a new bitstream payload with reduced buffer requirements. All bits for LFEs shall be supplied from the total buffer requirements based on the SCEs and CPEs.

Furthermore, all bits contained in an access unit, including any extension element payloads and MHAS transport overhead, shall also be considered in the total buffer requirements.

For the definition of NCC (number of considered channels), see ISO/IEC 14496-3:2009, 1.3.

5.6.2 Bit reservoir

The bit reservoir is controlled at the encoder. The maximum bit reservoir in the encoder depends on the NCC and the mean bitrate. The maximum bit reservoir size for constant rate channels can be calculated by subtracting the mean number of bits per frame from the minimum decoder input buffer size.

For example, at 32 kbit/s with two 48 kHz sampled mono objects and a frame length of 1024 samples the mean number of bits per frame (*mean_aulengh*) is:

$$mean\_aulength = \frac{32\,000\,\frac{bits}{sec} \cdot 1024\,\frac{samples}{frame}}{48\,000\,\frac{samples}{sec}} = 682.666\ldots\,\frac{bits}{frame}$$

This leads to a maximum bit reservoir size (*max_bit_reservoir*) of:

$$max\_bit\_reservoir = FLOOR\left(12288\,\frac{bits}{frame} - 682.666\ldots\,\frac{bits}{frame}\right) = 11605\frac{bits}{frame}.$$

For variable bitrate channels the encoder should operate in a way that the input buffer requirements do not exceed the minimum decoder input buffer.

For a delay optimized decoder start up the state of the bit reservoir (*bit_reservoir_state*) can be transmitted in a *buffer_fullness* field. If the buffer fullness is known at the decoder, the decoder does not have to wait until the input buffer is fully filled. Instead, the decoder only needs to buffer the actual frame and a number of additional bits as indicated in *bit_reservoir_state* before starting to decode.

The *bit_reservoir_state* of subsequent frames can be derived as follows:

$$bit\_reservoir\_state[frame] = bit\_reservoir\_state[frame-1] + mean\_aulength - aulength[frame]$$

aulength has to be adjusted such that the following restriction is met:

$$0 \leq bit\_reservoir\_state[frame] \leq max\_bit\_reservoir$$

5.6.3 Maximum bit rate

The maximum bitrate depends on the audio sampling rate (as given by **usacSamplingFrequencyIndex** and **usacSamplingFrequency**) and the output frame length (as given by **coreSbrFrameLengthIndex**). It can be calculated based on the minimum input buffer size according to the formula:

$$\frac{6144\,\frac{bits}{block} \cdot usacSamplingFrequency}{outputFrameLength} \cdot NCC$$

To give an example the maximum bitrate per channel for a sampling rate of 48 kHz and outputFrameLength of 768 samples is 384 $kbit/_{sec}$ per *NCC*.

5.7 Stream access point requirements and inter-frame dependency

For the sake of increased coding efficiency the MPEG-H 3D audio coding scheme may exploit inter-frame redundancy. This coding strategy introduces inter-frame dependencies, which means that the decoding process of one frame requires the knowledge of the previously decoded frame.

To allow for (random) stream access MPEG-H 3D audio features the concept of the independently decodable frame in which no such inter-frame dependencies exist. Decoders can easily identify and use this independently decodable frame for start-up or as stream access points.

The bitstream field which indicates that a frame is independently decodable is usacIndependencyFlag. The flag is conveniently located at the very beginning of each mpegh3daFrame(). If usacIndependencyFlag == 1 then the present frame shall be independently decodable. See also the definition of usacIndependencyFlag in ISO/IEC 23003-3.

Consequently, an independently decodable frame shall have the following properties.

— No processing block may operate in a state in which it requires information from the previous frame to perform the decoding process. In particular:

— The context of the arithmetic coder shall be reset (arith_reset_flag == 1).

— if eSBR is used:

  — it shall not employ delta coding in the time direction over SBR frame borders;

  — sbrInfo() and sbrHeader() shall be present (sbrInfoPresent == 1, sbrHeaderPresent == 1);

  — re-use of PVC IDs is not allowed (reuse_pvcID == 0).

— if IGF is used:

  — it shall not employ delta coding in time direction over frame borders;

  — it shall not make use of previous tile indices nor previous whitening levels.

— if complex prediction stereo is used:

  — it shall not employ delta coding in time direction (delta_code_time == 0) over frame borders;

  — it shall not make use of previous frame information (use_prev_frame==0).

— if MPS212 is used, bsIndependencyFlag shall be 1.

— if SAOC 3D is used, bsIndependencyFlag shall be 1.

— if HOA is used, hoaIndependencyFlag shall be 1.

— If object_metadata (OAM) is used, independently decodable oam_metadata() has to be available (intracoded_object_metadata_efficient() in case of efficient object metadata coding or intracoded_object_metadata_low_delay() in case of object metadata coding with low delay).

Note that most of the above requirements are fulfilled intrinsically by the design of the bitstream syntax.

An independently decodable frame may contain audio pre-roll information as described in subclause 5.5.6.

In an MHAS stream an independently decodable frame may be signalled by means of a previously transmitted MHAS packet of type PACTYP_MARKER as defined in subclause 14.4.6.3.

In order to decode the frame the information from mpegh3daConfig() is required to be known to the decoder as well.

6 Dynamic range control and loudness processing

6.1 General

This clause describes the decoding process of loudness metadata and dynamic range control (DRC) metadata. These are needed for different tasks including loudness monitoring and normalization, dynamic range control in noisy and quiet playback environments, or for other audio enhancement scenarios.

6.2 Description

Coding, transmission and application of loudness information and dynamic range control gains shall be based on ISO/IEC 23003-4.

6.3 Syntax

6.3.1 Loudness metadata

The loudness metadata is located in an mpegh3daConfigExtension as defined in Table 24. The content of mpegh3daLoudnessInfoSet() is listed in Table 125. loudnessInfo() and loudnessInfoSetExtension() are defined in ISO/IEC 23003-4. Other syntax elements are either defined in subclause 6.3.3 or in other parts of ISO/IEC 23003.

Table 125 — Syntax of mpegh3daLoudnessInfoSet()

Syntax	No. of bits	Mnemonic
mpegh3daLoudnessInfoSet ()		
{		
loudnessInfoCount;	**6**	**uimsbf**
for (i=0; i<loudnessInfoCount; i++) {		
loudnessInfoType;	**2**	**uimsbf**
if (loudnessInfoType == 1 || loudnessInfoType == 2) {		
mae_groupID;	**7**	**uimsbf**
} else if (loudnessInfoType == 3) {		
mae_groupPresetID;	**5**	**uimsbf**
}		
loudnessInfo();		
}		
loudnessInfoAlbumPresent;	**1**	**uimsbf**
if (loudnessInfoAlbumPresent) {		
loudnessInfoAlbumCount;	**6**	**uimsbf**
for (i=0; i< loudnessInfoAlbumCount; i++) {		
loudnessInfoType = 0;		
loudnessInfo();		
}		
}		
loudnessInfoSetExtensionPresent;	**1**	**uimsbf**
if (loudnessInfoSetExtensionPresent == 1) {		
loudnessInfoSetExtension();		
}		
}		

6.3.2 Dynamic range control metadata

The dynamic range control (DRC) metadata is located in an mpegh3daExtElementConfig and an mpegh3daExtElement as defined in Table 23 and Table 76. The static DRC metadata is defined by mpegh3daUniDrcConfig() listed in Table 126. The dynamic DRC metadata is defined by uniDrcGain()

listed in ISO/IEC 23003-4. drcCoefficientsUniDrc(), drcInstructionsUniDrc(), uniDrcConfigExtension(), drcSampleRate, and baseChannelCount are defined in ISO/IEC 23003-4. Other syntax elements are either defined in subclause 6.3.3 or in other parts of ISO/IEC 23008.

Table 126 — Syntax of mpegh3daUniDrcConfig()

Syntax	No. of bits	Mnemonic
mpegh3daUniDrcConfig()		
{		
drcSampleRate = sampling frequency as determined from mpegh3daConfig();		
drcCoefficientsUniDrcCount;	**3**	**uimsbf**
drcInstructionsUniDrcCount;	**6**	**uimsbf**
mpegh3daUniDrcChannelLayout();		
for (i=0; i< drcCoefficientsUniDrcCount; i++) {		
drcCoefficientsUniDrc();		
}		**uimsbf**
for (i=0; i< drcInstructionsUniDrcCount; i++) {		
drcInstructionsType;	**1..2**	**vlclbf**
if (drcInstructionsType == 2) {		
mae_groupID;	**7**	**uimsbf**
} else if (drcInstructionsType == 3) {		
mae_groupPresetID;	**5**	**uimsbf**
}		
drcInstructionsUniDrc();	[a]	
}		
uniDrcConfigExtPresent;	**1**	**uimsbf**
if (uniDrcConfigExtPresent == 1) {		
uniDrcConfigExtension();		
}		
loudnessInfoSetPresent;	**1**	**uimsbf**
if (loudnessInfoSetPresent == 1) {		
mpegh3daLoudnessInfoSet();		
}		
}		

[a] Parsing of the drcInstructionsUniDrc() structure might require information on present *downmixId* definitions as specified by the downmixConfig() structure (see Table 26).

Table 127 — Syntax of mpegh3daUniDrcChannelLayout()

Syntax	No. of bits	Mnemonic
mpegh3daUniDrcChannelLayout ()		
{		
baseChannelCount;	**7**	**uimsbf**
}		

6.3.3 Data elements

loudnessInfoType This field signals whether the following loudnessInfo() block refers to a fixed audio scene (default), to a specific audio element (mae_groupID), to an audio scene which includes a specific audio element (mae_groupID), or to an audio scene defined by a combination of audio elements (mae_groupPresetID). The encoding of loudnessInfoType is specified in Table 128. Note that a value of 0 represents the default audio scene selected by the decoder. If mae_numGroupPresets > 0, the group preset with the lowest mae_groupPresetID value is selected as the default audio scene (see subclause 15.3).

Table 128 — Coding and meaning of loudnessInfoType

loudnessInfoType (value)	binary encoding	codeword size [bits]	meaning
0	'00'	2	loudnessInfo() for fixed audio scene (default audio scene)
1	'01'	2	loudnessInfo() for single audio element defined by mae_groupID
2	'10'	2	loudnessInfo() for audio scene which includes a specific mae_groupID
3	'11'	2	loudnessInfo() for audio scene defined by mae_groupPresetID

drcInstructionsType This field signals whether the following drcInstructionsUniDrc() block refers to a fixed audio scene (default), to an audio scene which includes a specific audio element (mae_groupID), or to an audio scene defined by a combination of audio elements (mae_groupPresetID).

Table 129 — Coding and meaning of drcInstructionsType

drcInstructionsType (value)	binary encoding	codeword size [bits]	meaning
0	'0'	1	drcInstructionsUniDrc() for fixed audio scene (default)
2	'10'	2	drcInstructionsUniDrc() for audio scene which includes a specific mae_groupID
3	'11'	2	drcInstructionsUniDrc() for audio scene defined by mae_groupPresetID

baseChannelCount Sum of all present channels, objects, internal SAOC 3D channels/objects and HOA coefficient channels. baseChannelCount shall have a value as determined by the following pseudo code:

```
baseChannelCount = numAudioChannels;     NOTE: from Signals3d()
baseChannelCount += numAudioObjects;     NOTE: from Signals3d()
if (numSAOCTransportChannels > 0) {
    baseChannelCount += NumInputSignals;     NOTE: from SAOC3DSpecificConfig()
}
if (numHOATransportChannels > 0) {
    baseChannelCount += NumOfHoaCoeffs   NOTE: from HoaConfig()
}
```

6.4 Decoding process

6.4.1 General

The decoding and application of dynamic range control gains and loudness normalization gains is in general identical to the specification in ISO/IEC 23003-4. Therefore, the following subclauses only provide differences and specific MPEG-H related configuration details. A high-level block diagram of the complete MPEG-H decoder processing chain including blocks for dynamic range control, loudness normalization and peak limiting is depicted in Figure 39. Note that the definitions and descriptions for loudnessInfoSet() and uniDrcConfig() as specified in ISO/IEC 23003-4 equally hold for mpegh3daLoudnessInfoSet() and mpegh3daUniDrcConfig() in MPEG-H unless stated otherwise.

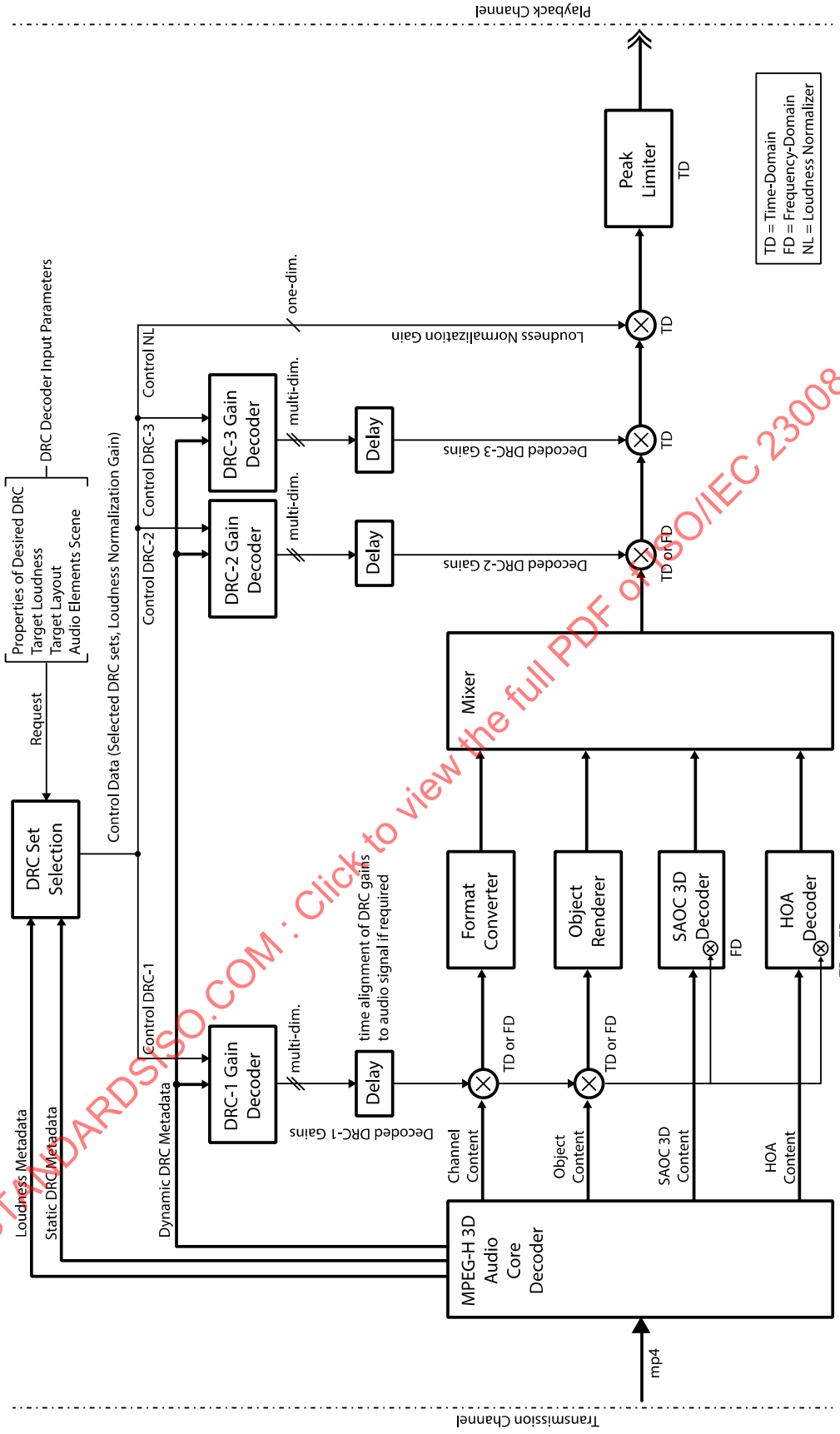


Figure 39 — MPEG-H decoder processing chain including blocks for dynamic range control, loudness normalization and peak limiting

6.4.2 Dynamic range control

There are three DRC gain decoders (denoted by DRC-1, DRC-2, and DRC-3) available in the MPEG-H decoder processing chain. All three decoders are identical and conform to ISO/IEC 23003-4. Each decoder addresses different tasks and scenarios. The decoded gains are applied by multiplication of the audio signal in the time- or frequency-domain as shown in Figure 39. Note that all three DRC gain decoders are driven by the same DRC metadata stream.

DRC-1 addresses dynamic range control for individual channels, individual objects, SAOC 3D content and HOA content. Additionally, DRC-1 shall support multi-band DRC. Dependent on the availability of the QMF or STFT representation of the audio signal, single-band DRC gains should be either applied in the time-domain or in the frequency-domain. Further details on the DRC application to SAOC 3D and HOA content are defined in subclauses 6.4.5 and 6.4.6, respectively.

DRC-2 addresses dynamic range control for the entire audio scene. DRC-2 shall also support multi-band DRC. Dependent on the availability of the QMF or STFT representation of the audio signal, single-band DRC gains should be either applied in the time-domain or in the frequency-domain. DRC sets designed for DRC-2 are restricted to one DRC channel group to be compatible to all target rendering layouts after the mixer module. Thus, DRC-2 is also best suited for simple DRC tasks especially in case of low-complexity requirements.

Note that in MPEG-H multi-band DRC gains shall always be applied in the QMF-domain, STFT-domain or in the SAOC processing band domain. The usage of the multi-band DRC filterbank defined in ISO/IEC 23003-4 is not permitted.

DRC-3 addresses dynamic range control and/or playback level dependent guided clipping prevention for specific target layouts (channel configurations). DRC-3 shall always operate in the time-domain. Thus, DRC-3 only supports single-band DRC. However, it shall support multiple DRC channel groups. Note that the gain application of DRC-2 and DRC-3 can be only combined if both operate in the time-domain.

Due to general restrictions in ISO/IEC 23003-4, either one or two DRC decoders are active simultaneously to achieve one specific DRC effect. Note that the DRC-3 gain decoder is only suited for a restricted number of target layouts. In all cases, the DRC gain decoders shall provide gains suitable for the present audio signal representation according to the downsampling and frequency mapping rules defined in ISO/IEC 23003-4. Note that if one DRC gain decoder is used to apply a non-DRC effect, such as fading or ducking, all three DRC gain decoders can be active simultaneously if the others additionally serve a specific DRC effect.

If binaural rendering is enabled, DRC-2 shall be applied to the output of the binaural renderer (not shown in Figure 39). The same holds for the loudness normalization gain. Note that DRC-3 gains are not supported in case of binaural rendering.

6.4.3 Usage of downmixId in MPEG-H

According to ISO/IEC 23003-4, a non-zero identifier, termed downmixId, can be used to externally refer to a specific downmix or rendering layout. In ISO/IEC 23003-4, downmixId is defined in the downmixInstructions() payload. If a DRC set contains a specific downmixId, the DRC set shall be applied to the output of the downmix or rendering which downmixId refers to. If loudness metadata contains a specific downmixId, this indicates that the loudness metadata was obtained from the output of the downmix or rendering which downmixId refers to.

In MPEG-H, downmixInstructions() is replaced by DownmixMatrixSet() as defined in Table 27, which permits the definition of a downmixId not only for transmitted downmix matrices, but also for default downmix matrices available on the decoder side (format converter). At the decoder a downmixId matching algorithm determines which downmixId is present as outlined in subclause 10.3.1.2.

According to ISO/IEC 23003-4, downmixId has two reserved values, namely 0x0 and 0x7F, which are also reserved values in MPEG-H.

A downmixId of 0x0 indicates that a DRC set shall be applied to the unmodified MPEG-H 3D audio core decoder output by DRC-1. On the encoder side, the DRC gain sequences have to be sequentially assigned to the mixed-content multi-channel output of the MPEG-H 3D audio core decoder. For combined channel-, object-, SAOC- and HOA-content, the size of the DRC assignment loop is defined by the baseChannelCount, which is the sum of all channels, all objects, all internal SAOC 3D channels/objects and the square of HoaOrder+1, NumOfHoaCoeffs (see definition in Table 127). In contrast to ISO/IEC 23003-4, DRC sets with ducking effect are only permitted for DRC sets with a downmixId of 0x0. Furthermore, in MPEG-H it is permitted to define more than one DRC set with ducking effect for the same downmixId if those DRC sets are differentiated by unique audio element group identifiers (mae_groupIDs/mae_groupPresetIDs) (see Table 126 and Clause 15). In case of loudness metadata a downmixId of 0x0 shall refer to the reference layout (referenceLayout) specified in mpegh3daConfig().

A downmixId of 0x7F indicates that a DRC set shall be applied by DRC-2. In contrast to ISO/IEC 23003-4, it is not allowed to freely choose the application position of DRC sets coded with a downmixId of 0x7F. For example, the clipping behaviour for one specific downmix configuration will be different if a DRC set is applied before or after the format converter in MPEG-H. Note that DRC-2 sets can always be applied irrespective of the actual target layout. In case of loudness metadata a downmixId of 0x7F is not permitted.

All other downmixId values (except 0x0 and 0x7F) indicate that a DRC set shall be applied to a specific target layout by DRC-3. In case of loudness metadata all other downmixId values shall refer to the target layout specified by downmixId.

The MPEG-H meaning of downmixId is again summarized in Table 130.

Table 130 — Meaning of downmixId in MPEG-H

Metadata type	downmixId == 0x0	downmixId == 0x7F	downmixId != 0x0 && downmixId != 0x7F
drcInstructionsUniDrc() (DRC set)	Shall be applied by DRC-1.	Shall be applied by DRC-2.	Shall be applied by DRC-3.
loudnessInfo() (loudness metadata)	Measurement of reference layout.	Not permitted.	Measurement of target layout specified by downmixId.

6.4.4 DRC set selection process

6.4.4.1 General

The DRC set selection process shall be implemented according to ISO/IEC 23003-4. Note that dmxInstructions() is replaced by DownmixMatrixSet() in MPEG-H. DownmixMatrixSet() is defined in Table 27.

If information on present audio elements is available (mae_groupID/mae_groupPresetID) (see Clause 15), the pre-selection process specified in ISO/IEC 23003-4 shall be extended by two additional requirements as listed in Table 131. Note that if in pre-selection step #8 the requirement of clipping

prevention needs to be relaxed, at least one DRC set has to remain that either matches the present audio elements or has to be independent of any audio elements (drcInstructionsType==0). The final selection based on present audio elements shall be performed after the final selection based on output peak value (see subclauses 6.4.4.2, 6.4.4.3 and ISO/IEC 23003-4). If information on present audio elements is not available to the DRC selection process, DRC sets and loudness metadata with defined audio element group identifiers (mae_groupID/mae_groupPresetID) shall be ignored by the DRC selection process.

Note that the applicable loudnessInfo() block for loudness normalization according to ISO/IEC 23003-4 is also dependent on the present audio elements. If loudnessInfo() blocks with matching group identifiers for present audio elements are not available, loudnessInfo() blocks with undefined group identifiers shall be used instead (loudnessInfoType==0).

A DRC set with ducking effect is automatically selected if the group identifiers of the present audio elements match the group identifier of the DRC set with ducking effect. Note that in MPEG-H a DRC set with ducking effect is always automatically selected if it doesn't define an audio element group identifier (drcInstructionsType==0).

The selection of DRC sets with fading effect shall be done according to ISO/IEC 23003-4.

Table 131 — Requirements for DRC pre-selection (additional MPEG-H requirements)

#	Requirement	Applicability	Comment
9	mae_groupPresetID matches or is undefined	If mae_groupPresetID is present.	See subclause 6.4.4.2
10	mae_groupID matches or is undefined	If mae_groupID is present and no match was found in step #9.	See subclause 6.4.4.3

6.4.4.2 Pre-selection based on present mae_groupPresetID (#9)

This pre-selection step addresses the case when a specific audio scene defined by a combination of audio elements (mae_groupPresetID) is present. All DRC sets, which define a matching mae_groupPresetID (see Table 126) shall be selected in this step. All other DRC sets with defined mae_groupPresetID shall be discarded. If at least one DRC set has a matching mae_groupPresetID, pre-selection step #10 shall be omitted and all DRC sets with defined mae_groupID shall be discarded. If no DRC set has a matching mae_groupPresetID, all DRC sets with defined mae_groupPresetID are discarded and the pre-selection shall continue with step #10. If several mae_groupPresetIDs are present, the matching shall be done separately for each identifier. Note that DRC sets which don't define a mae_groupPresetID shall also pass this pre-selection step.

If in the final selection according to ISO/IEC 23003-4 multiple DRC sets with different mae_groupPresetIDs are selected, the DRC set, which defines the largest number of members in its associated mae_groupPresetID, shall be selected. If there are still multiple DRC sets with different mae_groupPresetIDs selected, select the DRC set with the lowest mae_groupPresetID. Note that if the user has actively chosen a specific mae_groupPresetID, all corresponding DRC sets shall be preferred in this final selection step.

If in the final selection according to ISO/IEC 23003-4 multiple DRC sets with defined and undefined mae_groupPresetID are selected, all DRC sets with undefined mae_groupPresetID shall be discarded.

6.4.4.3 Pre-selection based on present mae_groupID (#10)

This pre-selection step addresses the case when an audio scene is present which includes a specific audio element (mae_groupID). All DRC sets which define a matching mae_groupID (see Table 126) shall

be selected in this step. All other DRC sets with defined mae_groupID shall be discarded. If no DRC set has a matching mae_groupID, all DRC sets with defined mae_groupID shall be discarded. If several mae_groupIDs are present, the matching shall be done separately for each identifier. Note that DRC sets which don't define a mae_groupID shall also pass this pre-selection step.

If in the final selection according to ISO/IEC 23003-4 multiple DRC sets with defined and undefined mae_groupID are selected, all DRC sets with undefined mae_groupID shall be discarded.

6.4.5 DRC-1 for SAOC 3D Content

For SAOC 3D, the DRC-1 decoder shall produce one DRC gain $\mathrm{drc}_{\mathrm{ch/obj}}(n,m,i)$ per QMF time slot n, per SAOC 3D processing band m and per internal SAOC 3D channel/object i. Appropriate down-sampling and frequency mapping rules are specified in ISO/IEC 23003-4.

The decoded DRC gains are always applied to the SAOC 3D processing bands. More precisely, the gains are applied on the SAOC 3D rendering matrix. The corresponding SAOC 3D rendering coefficients are defined for each parameter time slot l and processing band m. The SAOC 3D decoder shall provide the number of processing bands to the DRC decoder. The corresponding centre frequencies shall be derived in accordance with ISO/IEC 23003-1.

The final SAOC 3D DRC rendering matrix is computed by replacing the SAOC 3D unmodified rendering matrix **R** defined in subclause 9.5.3.4 by:

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_{\mathrm{ch}} & \mathbf{R}_{\mathrm{obj}} \end{bmatrix} \begin{bmatrix} \mathbf{DRC}_{\mathrm{ch}} & 0 \\ 0 & \mathbf{DRC}_{\mathrm{obj}} \end{bmatrix},$$

where $\mathbf{DRC}_{\mathrm{ch}}$ of size $N_{\mathrm{ch}} \times N_{\mathrm{ch}}$ represents the DRC gain matrix associated with the channel-content input and $\mathbf{DRC}_{\mathrm{obj}}$ of size $N_{\mathrm{obj}} \times N_{\mathrm{obj}}$ represents the DRC gain matrix associated with the object-content input.

The DRC gain matrices are defined for each parameter time slot l and processing band m and given by:

$$\mathbf{DRC}_{\mathrm{ch}}^{l,m}(i,j) = \begin{cases} \mathrm{drc}_{\mathrm{ch}}(n_0,m,i), & \text{if } i = j \\ 0, & \text{otherwise} \end{cases},$$

and

$$\mathbf{DRC}_{\mathrm{obj}}^{l,m}(i,j) = \begin{cases} \mathrm{drc}_{\mathrm{obj}}(n_0,m,i), & \text{if } i = j \\ 0, & \text{otherwise} \end{cases},$$

where n_0 represents the last time slot in parameter set l.

6.4.6 DRC-1 for HOA content

6.4.6.1 General

The DRC-1 gains for HOA are applied to the HOA signal before rendering and may be combined with rendering. The DRC-1 gains for HOA are either applied in the time-domain or in the QMF-domain.

6.4.6.2 Application of DRC-1 gains in the time-domain

The DRC decoder shall provide $(N+1)^2$ gain values $\boldsymbol{g}_{drc} = [g_1, .., g_{(N+1)^2}]^T$ according to the number of HOA coefficient channels of the HOA signal $\boldsymbol{c}$. N is the HOA order.

Application of DRC gains to the HOA signals:

$$\boldsymbol{c_{drc}} = \boldsymbol{D}_{DSHT}^{-1} diag(\boldsymbol{g}_{drc}) \boldsymbol{D}_{DSHT}\, \boldsymbol{c},$$

where $\boldsymbol{c}$ is a vector of one time sample of HOA coefficients ($\boldsymbol{c} \in \mathbb{R}^{(N+1)^2 \mathrm{x}\, 1}$), and $\boldsymbol{D}_{DSHT} \in \mathbb{R}^{(N+1)^2 \mathrm{x}\, (N+1)^2}$ and its inverse $\boldsymbol{D}_{DSHT}^{-1}$ are matrices related to a discrete spherical harmonics transform (DSHT) optimized for DRC purposes.

Informative remark: To decrease the computational load by $(N+1)^4$ operations per sample, it may be found advantageous to include the rendering step and to calculate the loudspeaker signals directly by: $\boldsymbol{w_{drc}} = (\boldsymbol{D}\, \boldsymbol{D}_{DSHT}^{-1})\ (diag(\boldsymbol{g}_{drc}) \boldsymbol{D}_{DSHT})\, \boldsymbol{c}$, where $\boldsymbol{D}$ is the rendering matrix and $(\boldsymbol{D}\, \boldsymbol{D}_{DSHT}^{-1})$ can be pre-computed.

If all gains $g_1, .., g_{(N+1)^2}$ have the same value of g_{drc}, then a single DRC channel group has been used. This case shall be flagged by the DRC-1 decoder because in this case the calculation of the spatial filter is not necessary and the calculation simplifies to $\boldsymbol{c_{drc}} = g_{\mathrm{drc}}\, \boldsymbol{c}$.

6.4.6.3 Calculation of DSHT matrices for DRC-1 gains

The matrix coefficients and operations required to determine the spatial filter $\boldsymbol{D}_{DSHT}$ and its inverse $\boldsymbol{D}_{DSHT}^{-1}$ are calculated as follows:

A set of spherical positions $\mathfrak{D}_{DSHT} = [\, \boldsymbol{\Omega}_1, \ldots, \boldsymbol{\Omega}_{\mathrm{l}}, .., \boldsymbol{\Omega}_{(\mathrm{N}+1)^2}]$ with $\boldsymbol{\Omega}_{\mathrm{l}} = [\theta_l, \phi_l]^T$ and related quadrature gains $\boldsymbol{q} \in \mathbb{R}^{(N+1)^2 \mathrm{x}\, 1}$ are selected from the Table in Annex F.25 based upon the HOA order N as an index.

A mode matrix $\boldsymbol{\Psi}_{DSHT}$ related to these positions is calculated. For details see Annex F.1. A first prototype matrix is calculated by $\widetilde{\boldsymbol{D}}_1 = diag(\boldsymbol{q}) \frac{\boldsymbol{\Psi}_{DSHT}}{(N+1)^2}$. A compact singular value decomposition is performed $\widetilde{\boldsymbol{D}}_1 = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$ and a new prototype matrix is calculated by: $\widehat{\widetilde{\boldsymbol{D}}}_2 = \boldsymbol{U}\boldsymbol{V}^T$. This matrix is normalized by: $\breve{\boldsymbol{D}}_2 = \frac{\widehat{\widetilde{\boldsymbol{D}}}_2}{\left\| \widehat{\widetilde{\boldsymbol{D}}}_2 \right\|_{fro}}$. A row-vector $\boldsymbol{e}$ is calculated by $= \frac{\boldsymbol{1}_L^T \breve{\boldsymbol{D}}_2 - [1,0,0,..,0]}{(\mathrm{N}+1)^2}$, where $[1,0,0,..,0]$ is a row vector of $(N+1)^2$ all zero elements except for the first element with a value of one. $\boldsymbol{1}_L^T \breve{\boldsymbol{D}}_2$ denotes the sum of rows of $\breve{\boldsymbol{D}}_2$.

The optimized DSHT matrix $\boldsymbol{D}_{DSHT}$ is now derived by $\boldsymbol{D}_{DSHT} = \breve{\boldsymbol{D}}_2 - [\, \boldsymbol{e}^T, \boldsymbol{e}^T, \boldsymbol{e}^T, ..]^T$.

6.4.6.4 Application of DRC-1 gains in the QMF-domain

The DRC-1 decoder provides a gain value $g_{ch}(n,m)$ for every time frequency tile (n,m) for $(N+1)^2$ spatial channels. The gains for time slot n and frequency band m are arranged in $\boldsymbol{g}(n,m) \in \mathbb{R}^{(N+1)^2 \mathrm{x}\, 1}$.

Multi-band DRC gains are applied in the QMF-domain. The processing steps are shown in Figure 40. The reconstructed HOA signal is transformed into the spatial domain by (inverse DSHT): $W_{DSHT} = D_{DSHT} C$, where $\boldsymbol{C} \in \mathbb{R}^{(N+1)^2 \mathrm{x}\, \tau}$ is a block of τ HOA samples and $\boldsymbol{W}_{DSHT} \in \mathbb{R}^{(N+1)^2 \mathrm{x}\, \tau}$ is a block of spatial samples matching the input time granularity of the QMF filter bank. Then the QMF analysis filter bank is applied. Let $\widehat{\boldsymbol{w}}_{DSHT}(n,m) \in \mathbb{C}^{(N+1)^2 \mathrm{x}\, 1}$ denote the vector of spatial channels per time frequency tile (n,m). Then the DRC gains are applied: $\breve{\boldsymbol{w}}_{DRC}(n,m) = diag\big(\boldsymbol{g}(n,m)\big)\, \widehat{\boldsymbol{w}}_{DSHT}(n,m)$. To minimize the computational complexity the DSHT and rendering to loudspeaker channels are combined:

$\boldsymbol{w}(n,m) = \boldsymbol{D}\,\boldsymbol{D}_{DSHT}^{-1}\,\breve{\boldsymbol{w}}_{DRC}(n,m)$, where $\boldsymbol{D}$ denotes the HOA rendering matrix. The QMF signals shall then be fed to the mixer for further processing.

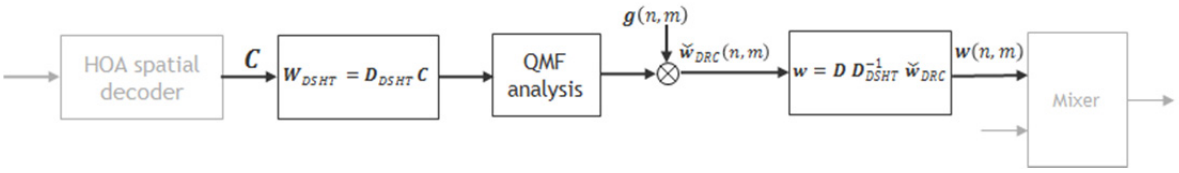


Figure 40 — DRC-1 gain application for HOA in the QMF-domain combined with rendering step

If only a single DRC channel group is present, this should be flagged by the DRC-1 decoder since again computational simplifications are possible (Figure 41). In this case the gains in vector $g(n,m)$ all share the same value of $g_{drc}(n,m)$. The QMF filter bank may be directly applied to the HOA signal and the gain $g_{drc}(n,m)$ shall then be multiplied in the QMF-domain. The resulting loudspeaker channels in the QMF-domain after rendering are obtained by $\boldsymbol{w}(n,m) = g_{drc}(n,m)\boldsymbol{D}\boldsymbol{c}_{DRC}(n,m)$.

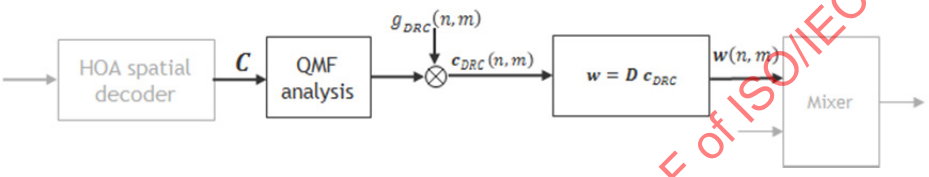


Figure 41 — DRC-1 gain application for HOA in the QMF-domain combined with rendering step for the simple case of a single DRC channel group

Tables for the application of DRC in HOA can be found in Annex F.25

6.4.7 Loudness normalization

Loudness normalization shall be performed in the time-domain after DRC-3. The processing shall be conducted according to ISO/IEC 23003-4. Note that the normalization gain also depends on the result of the DRC set selection process as shown in Figure 39. Configuration changes restricted to the mpegh3daLoudnessInfoSet() structure shall not re-initialize the decoder but update the normalization gain by smooth interpolation over one DRC frame.

In case of binaural rendering, loudness normalization shall be applied after DRC-2.

6.4.8 Peak limiter

The application of a peak limiter is mandatory. Further details are specified in Annex D. The peak limiter shall be placed at the very end of the audio processing chain.

6.4.9 Time-synchronization of DRC gains

DRC gains shall be sent aligned to the IMDCT output waveforms as specified in subclause 4.6. If the waveforms encounter delays in the processing pipeline from the IMDCT output until reaching the DRC application block, e.g. due to an analysis filterbank for DRC application in the frequency domain, the DRC gains shall be delayed accordingly at the decoder before application to the audio signal.

6.4.10 Default parameters

The following default parameters as defined in ISO/IEC 23003-4 shall be applied to the DRC decoder:

— loudnessDeviationMax = 0 dB;

— outputPeakLevelMax = 0 dBFS.

The following default parameters as defined in ISO/IEC 23003-4 should be applied to the DRC decoder:

— numDrcEffectTypeRequests >= 6 (see ISO/IEC 23003-4:2015, Annex E.2.2 for recommended drcEffectTypeRequest fallback settings).

7 Object metadata decoding

7.1 General

This clause describes the decoding process of object metadata, i.e. geometrical data for audio objects. These are needed to apply the object rendering specified in Clause 8.

7.2 Description

Metadata is conveyed for every audio object providing its spatial positions (azimuth, elevation, and radius) and a linear gain at defined timestamps. In addition to that, either a uniform spread value of three distinct spread values (spread in width, height and depth dimension), as well as a dynamic object priority value may also be defined.

The units for each of the different metadata components are given in Table 132. A visualization of the polar coordinate system for azimuth and elevation is shown in Figure 42.

Table 132 — Units of the decoded object metadata components

Component	Unit	Value range
azimuth	° (degrees)	-180; 180
elevation	° (degrees)	-90; 90
radius	m (meter)	0.5; 16
gain	none (linear)	0.004; 5.957
spread (uniform)	° (degrees)	0; 180
spread width	° (degrees)	0; 180
spread height	° (degrees)	0; 90
spread depth	m (metre)	0; 15.5
dynamic object priority	none	0; 7

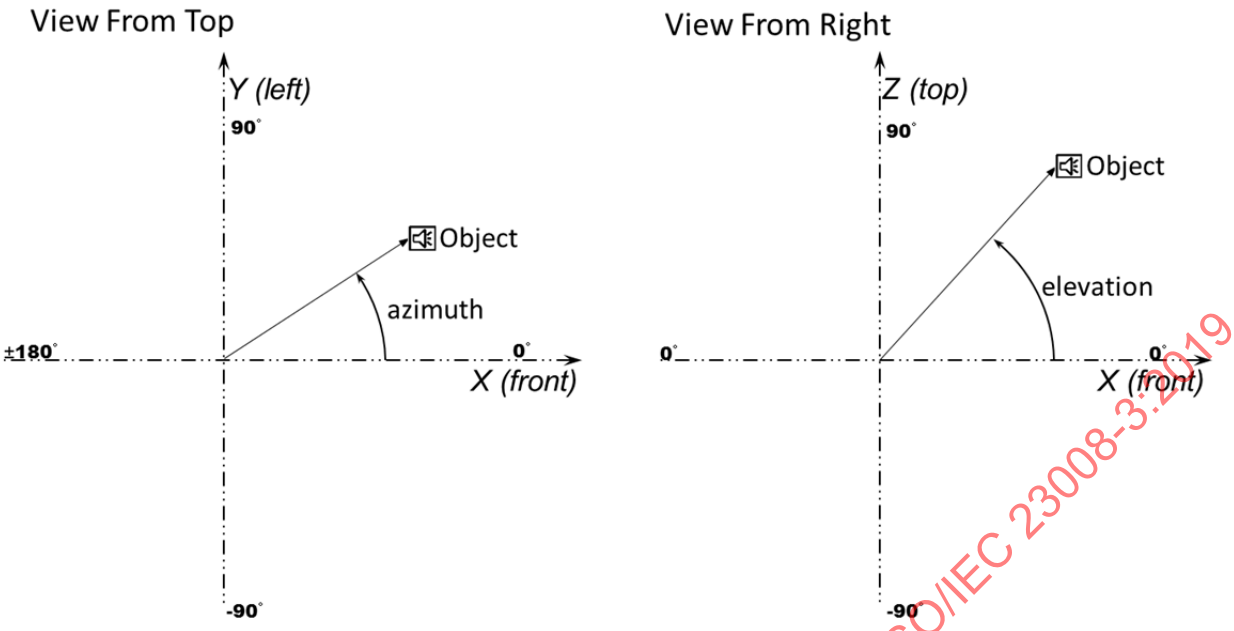


Figure 42 — Visualization of polar coordinate system for signalling spatial positions of audio objects

7.3 Syntax

7.3.1 Object metadata configuration

As mentioned in the extension payload (see Table 23), the ObjectMetadataConfig() specifies the metadata decoding method.

Table 133 — Syntax of ObjectMetadataConfig()

Syntax	No. of bits	Mnemonic
ObjectMetadataConfig()		
{		
lowDelayMetadataCoding;	**1**	**bslbf**
hasCoreLength;	**1**	**bslbf**
if (!hasCoreLength) {		
frameLength;	**6**	**uimsbf**
OAMFrameLength = (frameLength+1)<<6;		
} else {		
OAMFrameLength = outputFrameLength;		
}		
hasScreenRelativeObjects;	**1**	**bslbf**
if(hasScreenRelativeObjects) {		
for (o = 0; o < num_objects; o++) { /* NOTE 1 */		
isScreenRelativeObject[o]**;**	**1**	**bslbf**
}		
}		
hasDynamicObjectPriority	**1**	**bslbf**
hasUniformSpread;	**1**	**bslbf**
}		
NOTE num_objects is equal to the number of objects in the associated signal group.		

If the flag **lowDelayMetadataCoding** is true, low delay object metadata are present in the bitstream and therefore, the low delay object metadata syntax shall be used. Otherwise, efficient object metadata are present int the bitstream and the according syntax shall be used.

The coreCoderFrameLength (given by mpegh3daConfig()) shall be an integer multiple of the OAMFrameLength. The OAMFrameLength shall not be greater than the coreCoderFrameLength.

If the coreCoderFrameLength is an integer multiple of the OAMFrameLength (coreCoderFrameLength = $n \cdot$OAMFrameLength), the structure object_metadata() is sent n times in the objctMetadataFrame() as shown in Table 134. In this case, the i^{th} object_metadata() (i = 0,1,2,..,n-1) is available at the sample index $s[i]$, with $s[i]$ given by the following equation:

$$s[i] = (i + 1)\ \text{OAMFramelength-1}$$

where s is the sample index of an audio frame (0,1,2,.., $n \cdot$ OAMFramelength-1).

7.3.2 Top level object metadata syntax

Table 134 — Syntax of objectMetadataFrame()

Syntax	No. of bits	Mnemonic

objectMetadataFrame() {
  if (OAMFrameLength* < coreCoderFrameLength**) {
  for (i = 0; i < coreCoderFrameLength / OAMFrameLength; ++i) {
  object_metadata_present;
	1	**uimsbf**

if (object_metadata_present)
  object_metadata();
  }
  }
  else {
  object_metadata();
  }
}		

* given by ObjectMetadataConfig() ** given by mpegh3daConfig()		

Table 135 — Syntax of object_metadata()

Syntax	No. of bits	Mnemonic

object_metadata()
{
  if (lowDelayMetadataCoding==0) {
  object_metadata_efficient()
  } else {
  object_metadata_low_delay()
  }
}		

7.3.3 Subsidiary payloads for efficient object metadata decoding

Table 136 — Syntax of object_metadata_efficient()

Syntax	No. of bits	Mnemonic
object_metadata_efficient()		
{		
intracoded_object_metadata_efficient();		
has_differential_metadata;	**1**	**bslbf**
if (has_differential_metadata) {		
differential_object_metadata();		
}		
}		

Table 137 — Syntax of intracoded_object_metadata_efficient()

Syntax	No. of bits	Mnemonic
intracoded_object_metadata_efficient()		
{		
ifperiod;	**6**	**uimsbf**
if (num_objects>1) {		
common_azimuth;	**1**	**bslbf**
if (common_azimuth) {		
default_azimuth;	**8**	**tcimsbf**
}		
else {		
for (o = 0; o < num_objects; o++) {		
position_azimuth[o]**;**	**8**	**tcimsbf**
}		
}		
common_elevation;	**1**	**bslbf**
if (common_elevation) {		
default_elevation;	**6**	**tcimsbf**
}		
else {		
for (o = 0; o < num_objects; o++) {		
position_elevation[o]**;**	**6**	**tcimsbf**
}		
}		
common_radius;	**1**	**bslbf**
if (common_radius) {		
default_radius;	**4**	**uimsbf**
}		
else {		
for (o = 0; o < num_objects; o++) {		
position_radius[o]**;**	**4**	**uimsbf**
}		
}		
common_gain;	**1**	**bslbf**
if (common_gain) {		
default_gain;	**7**	**tcimsbf**
}		
else {		

Syntax	No. of bits	Mnemonic
for (o = 0; o < num_objects; o++) {		
gain_factor[o];	7	tcimsbf
}		
}		
common_spread;	1	bslbf
if (common_spread) {		
if (hasUniformSpread) {		
default_spread;	7	uimsbf
}		
else {		
default_spread_width;	7	uimsbf
default_spread_height;	5	uimsbf
default_spread_depth;	4	uimsbf
}		
}		
else {		
for (o = 0; o < num_objects; o++) {		
if (hasUniformSpread) {		
spread[o];	7	uimsbf
}		
else {		
spread_width[o];	7	uimsbf
spread_height[o];	5	uimsbf
spread_depth[o];	4	uimsbf
}		
}		
}		
if (hasDynamicObjectPriority) {		
common_dynamic_object_priority;	1	bslbf
if (common_dynamic_object_priority) {		
default_dynamic_object_priority;	3	uimsbf
}		
else {		
for (o = 0; o < num_objects; o++) {		
dynamic_object_priority[o];	3	uimsbf
}		
}		
}		
}		
else {		
position_azimuth;	8	tcimsbf
position_elevation;	6	tcimsbf
position_radius;	4	uimsbf
gain_factor;	7	tcimsbf
if (hasUniformSpread) {		
spread;	7	uimsbf
}		
else {		
spread_width;	7	uimsbf
spread_height;	5	uimsbf
spread_depth;	4	uimsbf
}		
if (hasDynamicObjectPriority) {		

Syntax	No. of bits	Mnemonic
dynamic_object_priority;	**3**	**uimsbf**
}		
}		
}		

Table 138 — Syntax of differential_object_metadata()

Syntax	No. of bits	Mnemonic
differential_object_metadata() {		
bits_per_point;	**4**	**uimsbf**
fixed_azimuth;	**1**	**bslbf**
if (!fixed_azimuth) {		
for (o = 0; o < num_objects; o++) {		
flag_azimuth;	**1**	**bslbf**
if (flag_azimuth) {		
num_points_azimuth = offset_data(bits_per_point);		
nbits_azimuth;	**3**	**uimsbf**
for (p = 0; p < num_points_azimuth; p++) {		
differential_azimuth[o][p]**;**	**nbits_azimuth + 2**	**tcimsbf**
}		
}		
}		
}		
fixed_elevation;	**1**	**bslbf**
if (!fixed_elevation) {		
for (o = 0; o < num_objects; o++) {		
flag_elevation;	**1**	**bslbf**
if (flag_elevation) {		
num_points_elevation = offset_data(bits_per_point);		
nbits_elevation;	**3**	**uimsbf**
for (p = 0; p < num_points_elevation; p++) {		
differential_elevation[o][p]**;**	**nbits_elevation + 2**	**tcimsbf**
}		
}		
}		
}		
fixed_radius;	**1**	**bslbf**
if (!fixed_radius) {		
for (o = 0; o < num_objects; o++) {		
flag_radius;	**1**	**bslbf**
if (flag_radius) {		
num_points_radius = offset_data(bits_per_point);		
nbits_radius;	**3**	**uimsbf**
for (p = 0; p < num_points_radius; p++) {		
differential_radius[o][p]**;**	**nbits_radius + 2**	**tcimsbf**
}		
}		
}		
}		
fixed_gain;	**1**	**bslbf**
if (!fixed_gain) {		
for (o = 0; o < num_objects; o++) {		

Syntax	No. of bits	Mnemonic
flag_gain;	**1**	**bslbf**
if (flag_gain) {		
num_points_gain = offset_data(bits_per_point);		
nbits_gain;	**3**	**uimsbf**
for (p = 0; p < num_points_gain; p++) {		
differential_gain[o][p]**;**	**nbits_gain + 2**	**tcimsbf**
}		
}		
}		
}		
fixed_spread;	**1**	**bslbf**
if (!fixed_spread) {		
for (o = 0; o < num_objects; o++) {		
if (hasUniformSpread) {		
flag_spread;	**1**	
if (flag_spread) {		
num_points_spread = offset_data(bits_per_point);		
nbits_spread;	**3**	
for (p = 0; p < num_points_spread; p++) {		
differential_spread[o][p]**;**	**nbits_spread + 2**	
}		
}		
}		
else {		
flag_spread_width;	**1**	**bslbf**
if (flag_spread_width) {		
num_points_spread_width = offset_data(bits_per_point);		
nbits_spread_width;	**3**	**uimsbf**
for (p = 0; p < num_points_spread_width; p++) {		
differential_spread_width[o][p]**;**	**nbits_spread_width + 2**	**tcimsbf**
}		
}		
flag_spread_height;	**1**	**bslbf**
if (flag_spread_height) {		
num_points_spread_height = offset_data(bits_per_point);		
nbits_spread_height;	**3**	**uimsbf**
for (p = 0; p < num_points_spread_height; p++) {		
differential_spread_height[o][p]**;**	**nbits_spread_height + 2**	**tcimsbf**
}		
}		
flag_spread_depth;	**1**	**bslbf**
if (flag_spread_depth) {		
num_points_spread_depth = offset_data(bits_per_point);		
nbits_spread_depth;	**3**	**uimsbf**
for (p = 0; p < num_points_spread_depth; p++) {		
differential_spread_depth[o][p]**;**	**nbits_spread_depth + 2**	**tcimsbf**
}		
}		
}		
}		
}		
if (hasDynamicObjectPriority) {		
fixed_dynamic_object_priority;	**1**	**bslbf**
if (!fixed_dynamic_object_priority) {		

Syntax	No. of bits	Mnemonic
for (o = 0; o < num_objects; o++) {		
flag_dynamic_object_priority;	**1**	**bslbf**
if (flag_dynamic_object_priority) {		
num_points_dynamic_object_priority = offset_data(bits_per_point);		
nbits_dynamic_object_priority;	**2**	**uimsbf**
for (p = 0; p < num_points_dynamic_object_priority; p++) {		
differential_dynamic_object_priority[o][p];	**nbits_dynamic_object_priority + 2**	**tcimsbf**
}		
}		
}		
}		
}		

Table 139 — Syntax of offset_data()

Syntax	No. of bits	Mnemonic
int offset_data(bits_per_point) {		
bitfield_syntax;	**1**	**bslbf**
if (bitfield_syntax) {		
offset_bitfield;	**ifperiod** [a]	**bslbf**
num_points = sum(offset_bitfield);		
}		
else {		
npoints;	**bits_per_point**	**uimsbf**
num_points = npoints + 1;		
for (p = 0; p < num_points; p++) {		
foffset[p];	**ceil(log2(ifperiod*))**	**uimsbf**
}		
}		
return num_points;		
}		
[a] Known from the intracoded object metadata for this current iframe_period.		

7.3.4 Subsidiary payloads for object metadata decoding with low delay

Table 140 — Syntax of object_metadata_low_delay ()

Syntax	No. of bits	Mnemonic
object_metadata_low_delay ()		
{		
has_intracoded_object_metadata;	**1**	**bslbf**
if (has_intracoded_object_metadata) {		
intracoded_object_metadata_low_delay();		
}		
else {		
dynamic_object_metadata();		
}		
}		

Table 141 — Syntax of intracoded_object_metadata_low_delay()

Syntax	No. of bits	Mnemonic
intracoded_object_metadata_low_delay()		
{		
if (num_objects>1) {		
fixed_azimuth;	**1**	**bslbf**
if (fixed_azimuth) {		
default_azimuth;	**8**	**tcimsbf**
}		
else {		
common_azimuth;	**1**	**bslbf**
if (common_azimuth) {		
default_azimuth;	**8**	**tcimsbf**
}		
else {		
for (o = 0; o < num_objects; o++) {		
position_azimuth[o]**;**	**8**	**tcimsbf**
}		
}		
}		
fixed_elevation;	**1**	**bslbf**
if (fixed_elevation) {		
default_elevation;	**6**	**tcimsbf**
}		
else {		
common_ elevation;	**1**	**bslbf**
if (common_elevation) {		
default_elevation;	**6**	**tcimsbf**
}		
else {		
for (o = 0; o < num_objects; o++) {		
position_elevation[o]**;**	**6**	**tcimsbf**
}		
}		
}		
fixed_radius;	**1**	**bslbf**
if (fixed_radius) {		
default_radius;	**4**	**uimsbf**
}		
else {		
common_radius;	**1**	**bslbf**
if (common_radius) {		
default_radius;	**4**	**uimsbf**
}		
else {		
for (o = 0; o < num_objects; o++) {		
position_ radius[o]**;**	**4**	**uimsbf**
}		
}		
}		
fixed_gain;	**1**	**bslbf**

Syntax	No. of bits	Mnemonic
if (fixed_gain) {		
default_gain;	**7**	**tcimsbf**
}		
else {		
common_gain;	**1**	**bslbf**
if (common_gain) {		
default_gain;	**7**	**tcimsbf**
}		
else {		
for (o = 0; o < num_objects; o++) {		
gain_factor[o];	**7**	**tcimsbf**
}		
}		
}		
fixed_spread;	**1**	**bslbf**
if (fixed_spread) {		
if (hasUniformSpread) {		
default_spread;	**7**	**uimsbf**
}		
else {		
default_spread_width;	**7**	**uimsbf**
default_spread_height;	**5**	**uimsbf**
default_spread_depth;	**4**	**uimsbf**
}		
}		
else {		
common_spread;	**1**	**bslbf**
if (common_spread) {		
if (hasUniformSpread) {		
default_spread;	**7**	**uimsbf**
}		
else {		
default_spread_width;	**7**	**uimsbf**
default_spread_height;	**5**	**uimsbf**
default_spread_depth;	**4**	**uimsbf**
}		
}		
else {		
for (o = 0; o < num_objects; o++) {		
if (hasUniformSpread) {		
spread[o];	**7**	**uimsbf**
}		
else {		
spread_width[o];	**7**	**uimsbf**
spread_height[o];	**5**	**uimsbf**
spread_depth[o];	**4**	**uimsbf**
}		
}		
}		
}		
if (hasDynamicObjectPriority) {		
fixed_dynamic_object_priority;	**1**	**bslbf**
if (fixed_dynamic_object_priority) {		
default_dynamic_object_priority;	**3**	**uimsbf**

Syntax	No. of bits	Mnemonic
}		
else {		
common_dynamic_object_priority;	**1**	**bslbf**
if (common_dynamic_object_priority) {		
default_dynamic_object_priority;	**3**	**uimsbf**
}		
else {		
for (o = 0; o < num_objects; o++) {		
dynamic_object_priority[o];	**3**	**uimsbf**
}		
}		
}		
}		
else {		
position_azimuth;	**8**	**tcimsbf**
position_elevation;	**6**	**tcimsbf**
position_radius;	**4**	**uimsbf**
gain_factor;	**7**	**tcimsbf**
if (hasUniformSpread) {		
spread;	**7**	**uimsbf**
}		
else {		
spread_width;	**7**	**uimsbf**
spread_height;	**5**	**uimsbf**
spread_depth;	**4**	**uimsbf**
}		
if (hasDynamicObjectPriority) {		
dynamic_object_priority;	**3**	**uimsbf**
}		
}		
}		

Table 142 — Syntax of dynamic_object_metadata()

Syntax	No. of bits	Mnemonic
dynamic_object_metadata() {		
flag_absolute;	**1**	**bslbf**
for (o = 0; o < num_objects; o++) {		
has_object_metadata;	**1**	**bslbf**
if (has_object_metadata) {		
single_dynamic_object_metadata(flag_absolute);		
}		
}		
}		

Table 143 — Syntax of single_dynamic_object_metadata()

Syntax	No. of bits	Mnemonic
single_dynamic_object_metadata (flag_absolute) {		
if (flag_absolute) {		
if (!fixed_azimuth[a]) {		
position_azimuth;	**8**	**tcimsbf**

Syntax	No. of bits	Mnemonic
}		
if (!fixed_elevation[a]) {		
position_elevation;	**6**	**tcimsbf**
}		
if (!fixed_radius[a]) {		
position_radius;	**4**	**uimsbf**
}		
if (!fixed_gain[a]) {		
gain_ factor;	**7**	**tcimsbf**
}		
if (!fixed_spread[a]) {		
if (hasUniformSpread) {		
spread;	**7**	**uimsbf**
}		
else {		
spread_width;	**7**	**uimsbf**
spread_height;	**5**	**uimsbf**
spread_depth;	**4**	**uimsbf**
}		
}		
if (hasDynamicObjectPriority) {		
if (!fixed_dynamic_object_priority[a]) {		
dynamic_object_priority;	**3**	**uimsbf**
}		
}		
}		
else {		
nbits;	**3**	**uimsbf**
num_bits = nbits +2;		
if (!fixed_azimuth[a]) {		
flag_azimuth;	**1**	**bslbf**
if (flag_azimuth) {		
position_azimuth_difference;	**num_bits**	**tcimsbf**
}		
}		
if (!fixed_elevation[a]) {		
flag_elevation;	**1**	**bslbf**
if (flag_elevation) {		
position_elevation_difference;	**min(num_bits,7)**	**tcimsbf**
}		
}		
if (!fixed_radius[a]) {		
flag_radius;	**1**	**bslbf**
if (flag_radius) {		
position_radius_difference;	**min(num_bits,5)**	**tcimsbf**
}		
}		
if (!fixed_gain[a]) {		
flag_gain;	**1**	**bslbf**
if (flag_gain) {		
gain_factor_difference ;	**min(num_bits,8)**	**tcimsbf**
}		

Syntax	No. of bits	Mnemonic
}		
if (!fixed_spread[a]) {		
if (hasUniformSpread) {		
flag_spread;	**1**	**bslbf**
if (flag_spread) {		
spread_difference ;	**min(num_bits,8)**	**tcimsbf**
}		
}		
else {		
flag_spread_width;	**1**	**bslbf**
if (flag_spread_width) {		
spread_width_difference ;	**min(num_bits,8)**	**tcimsbf**
}		
flag_spread_height;	**1**	**bslbf**
if (flag_spread_height) {		
spread_height_difference ;	**min(num_bits,6)**	**tcimsbf**
}		
flag_spread_depth;	**1**	**bslbf**
if (flag_spread_depth) {		
spread_depth_difference ;	**min(num_bits,5)**	**tcimsbf**
}		
}		
if (hasDynamicObjectPriority) {		
if (!fixed_dynamic_object_priority[a]) {		
flag_dynamic_object_priority;	**1**	**bslbf**
if (flag_dynamic_object_priority) {		
dynamic_object_priority_difference;	**min(num_bits,4)**	**tcimsbf**
}		
}		
}		
}		
}		

[a] Given by the preceding intracoded_object_metadata_low_delay() frame

7.3.5 Enhanced object metadata configuration

Table 144 — Syntax of EnhancedObjectMetadataConfig()

Syntax	No. of bits	Mnemonic
EnhancedObjectMetadataConfig()		
{		
/* static per group */		
hasDiffuseness;	**1**	**bslbf**
if (hasDiffuseness) {		
hasCommonGroupDiffuseness;	**1**	**bslbf**
}		
}		
hasExcludedSectors;	**1**	**bslbf**
if (hasExcludedSectors) {		
hasCommonGroupExcludedSectors;	**1**	**bslbf**
if (hasCommonGroupExcludedSectors) {		
useOnlyPredefinedSectors;	**1**	**bslbf**
}		
} else {		
hasCommonGroupExcludedSectors = 0;		

Syntax	No. of bits	Mnemonic
}		
hasClosestSpeakerCondition;	**1**	**bslbf**
if (hasClosestSpeakerCondition) {		
closestSpeakerThresholdAngle;	**7**	**uimsbf**
}		
/* static per object */		
for (o = 0; o < num_objects; o++) {		
hasDivergence[o]**;**	**1**	**bslbf**
if (hasDivergence[o]) {		
divergenceAzimuthRange[o]**;**	**6**	**uimsbf**
}		
if (hasCommonGroupExcludedSectors == 0) {		
useOnlyPredefinedSectors[o];	**1**	**bslbf**
}		
}		
}		

Table 145 — Syntax of EnhancedObjectMetadataFrame()

Syntax	No. of bits	Mnemonic
EnhancedObjectMetadataFrame()		
{		
/* dynamic per group */		
if (hasDiffuseness && hasCommonGroupDiffuseness) {		
if (independencyFlag == 0) {		
keepDiffuseness;	**1**	**bslbf**
} else {		
keepDiffuseness = 0;		
}		
if (keepDiffuseness == 0) {		
diffuseness;	**7**	**uimsbf**
}		
}		
if (hasCommonGroupExcludedSectors) {		
if (independencyFlag == 0) {		
keepExclusion;	**1**	**bslbf**
} else {		
keepExclusion = 0;		
}		
if (keepExclusion == 0) {		
numExclusionSectors;	**4**	**uimsbf**
if (useOnlyPredefinedSectors) {		
for (sc = 0; sc < numExclusionSectors; sc++) {		
excludeSectorIndex[sc];	**4**	**uimsbf**
}		
} else {		
for (sc = 0; sc < numExclusionSectors; sc++) {		
usePredefinedSector[sc];	**1**	**uimsbf**
if (usePredefinedSector[sc]) {		
excludeSectorIndex[sc];	**4**	**uimsbf**
} else {		
excludeSectorMinAzimuth[sc];	**7**	**uimsbf**
excludeSectorMaxAzimuth[sc];	**7**	**uimsbf**
excludeSectorMinElevation[sc];	**5**	**uimsbf**

Syntax	No. of bits	Mnemonic
excludeSectorMaxElevation[sc];	**5**	**uimsbf**
}		
}		
}		
}		
}		
/* dynamic per object */		
for (o = 0; o < num_objects; o++) {		
closestSpeakerPlayout[o]**;**	**1**	**bslbf**
if (hasDiffuseness && (hasCommonGroupDiffuseness == 0)) {		
if (independencyFlag == 0) {		
keepDiffuseness[o];	**1**	**bslbf**
} else {		
keepDiffuseness[o] = 0;		
}		
if (keepDiffuseness[o] == 0) {		
diffuseness[o];	**7**	**uimsbf**
}		
}		
if (hasDivergence[o]) {		
if (independencyFlag == 0) {		
keepDivergence[o];	**1**	**bslbf**
} else {		
keepDivergence[o] = 0;		
}		
if (keepDivergence[o] == 0) {		
divergence[o];	**7**	**uimsbf**
}		
}		
if (hasCommonGroupExcludedSectors == 0) {		
if (independencyFlag == 0) {		
keepExclusion[o]**;**	**1**	**bslbf**
} else {		
keepExclusion[o] = 0**;**		
}		
if (keepExclusion[o] == 0)		
{		
numExclusionSectors[o]**;**	**4**	**uimsbf**
if (useOnlyPredefinedSectors[o]) {		
for (sc = 0; sc < numExclusionSectors[o]; sc++) {		
excludeSectorIndex[o][sc];	**4**	**uimsbf**
}		
} else {		
for (sc = 0; sc < numExclusionSectors[o]; sc++) {		
usePredefinedSector[o][sc];	**1**	**uimsbf**
if (usePredefinedSector[o][sc]) {		
excludeSectorIndex[o][sc];	**4**	**uimsbf**
} else {		
excludeSectorMinAzimuth[o][sc];	**7**	**uimsbf**
excludeSectorMaxAzimuth[o][sc];	**7**	**uimsbf**
excludeSectorMinElevation[o][sc];	**5**	**uimsbf**

Syntax	No. of bits	Mnemonic
excludeSectorMaxElevation[o][sc];	**5**	**uimsbf**
}		
}		
}		
}		
}		
}		
}		

7.4 Data structure

7.4.1 Definition of ObjectMetadataConfig() payloads

lowDelayMetadataCoding indicates whether object metadata decoding with low delay is used. Otherwise, efficient object metadata are present in the bitstream.

hasCoreLength indicates whether the core coder works with the same number of audio samples per channel as the object metadata coder.

OAMFrameLength if the frame length of the core coder does not match the time resolution of the object metadata coder, the **OAMFrameLength** defines the size (in multiples of 64 audio samples) of a time slice for which a set of object metadata is available.

hasScreenRelativeObjects This flag specifies whether screen-relative objects are present.

isScreenRelativeObject This flag defines whether an object position is screen-relative. The object shall be rendered, such that its position is remapped according to subclause 18.2.4. It can still contain all valid angular values.

hasDynamicObjectPriority This flag indicates whether there is dynamic object priority data present in the bitstream or not. If there is no dynamic object priority data in the bitstream, mae_groupPriority indicates the dynamic object priority.

hasUniformSpread This flag indicates whether the spread of an object is given as uniform spread (flag is equal to 1) or as three independent values for width, height and depth (flag is equal to 0).

num_objects Is equal to the number of objects in the associated signal group.

7.4.2 Efficient object metadata decoding

7.4.2.1 Definition of object metadata payloads

7.4.2.1.1 Definition of object_metadata_efficient() payloads

has_differential_metadata indicates whether differential object metadata are present in the bitstream.

7.4.2.1.2 Definition of intracoded_object_metadata_efficient() payloads

ifperiod defines the number of audio frames for which OAM data are processed in intracoded_object_metadata_efficient() minus 1 (iframe_period = **ifperiod** + 1).

common_azimuth	indicates whether a common azimuth angle is used for all objects.
default_azimuth	defines the value of the common azimuth angle.
position_azimuth	if there is no common azimuth value, a value for each object is transmitted. If there is only one object, this is its azimuth angle.
common_elevation	indicates whether a common elevation angle is used for all objects.
default_elevation	defines the value of the common elevation angle.
position_elevation	if there is no common elevation value, a value for each object is transmitted. If there is only one object, this is its elevation angle.
common_radius	indicates whether a common radius value is used for all objects.
default_radius	defines the value of the common radius.
position_radius	if there is no common radius value, a value for each object is transmitted. If there is only one object, this is its radius.
common_gain	indicates whether a common gain value is used for all objects.
default_gain	defines the value of the common gain factor.
gain_factor	if there is no common gain value, a value for each object is transmitted. If there is only one object, this is its gain factor.
common_spread	indicates whether a common spread parameter is used for all objects.
default_spread	Defines the value of the common spread parameter in the case that there is just one uniform spread value.
default_spread_width	Defines the value of the common spread parameter for the spread in the dimension of width in the case of three independent spread values.
default_spread_height	Defines the value of the common spread parameter for the spread in the dimension of height in the case of three independent spread values.
default_spread_depth	Defines the value of the common spread parameter for the spread in the dimension of depth in the case of three independent spread values.

spread	If there is no common spread parameter, a value for each object is transmitted. If there is only one object, this is its spread parameter.
spread_width	If there is no common spread parameter, one value for the spread in the dimension of width is transmitted for each object. If there is only one object, this is its spread parameter in the dimension of width.
spread_height	If there is no common spread parameter, one value for the spread in the dimension of height is transmitted for each object. If there is only one object, this is its spread parameter in the dimension of height.
spread_depth	If there is no common spread parameter, one value for the spread in the dimension of depth is transmitted for each object. If there is only one object, this is its spread parameter in the dimension of depth.
common_dynamic_object_priority	indicates whether a common dynamic_object_priority is used for all objects.
default_dynamic_object_priority	defines the value of the common dynamic_object_priority.
dynamic_object_priority	This field defines the priority of the object. This field can take integer values between 0 and 7. The object may be discarded from rendering and decoding if the priority is lower than 7. If objects are discarded, the objects with lowest priority should be discarded first. If there is only one object, this is its dynamic_object_priority.

7.4.2.1.3 Definition of differential_object_metadata() payloads

bits_per_point	number of bits required to represent each of the polygon points (used in offset_data()).
fixed_azimuth	flag indicating whether the azimuth value is fixed for all objects.
flag_azimuth	flag per object indicating whether the azimuth value changes for this iframe_period.
nbits_azimuth	how many bits are required to represent the differential value minus 2.
differential_azimuth	value of the difference between the linearly interpolated and the actual value.
fixed_elevation	flag indicating whether the elevation value is fixed for all objects.

flag_elevation	flag per object indicating whether the elevation value changes for this iframe_period.
nbits_elevation	how many bits are required to represent the differential value minus 2.
differential_elevation	value of the difference between the linearly interpolated and the actual value.
fixed_radius	flag indicating whether the radius is fixed for all objects.
flag_radius	flag per object indicating whether the radius changes for this iframe_period.
nbits_radius	how many bits are required to represent the differential value minus 2.
differential_radius	value of the difference between the linearly interpolated and the actual value.
fixed_gain	flag indicating whether the gain factor is fixed for all objects.
flag_gain	flag per object indicating whether the gain radius changes for this iframe_period.
nbits_gain	how many bits are required to represent the differential value minus 2.
differential_gain	value of the difference between the linearly interpolated and the actual value.
fixed_spread	flag indicating whether the spread parameter is fixed for all objects.
flag_spread	flag per object indicating whether the spread parameter changes for this iframe_period.
nbits_spread	how many bits are required to represent the differential value minus 2.
differential_spread	value of the difference between the linearly interpolated and the actual value.
flag_spread_width	flag per object indicating whether the spread parameter in width dimension changes for this iframe_period.
nbits_spread_width	how many bits are required to represent the differential value minus 2.
differential_spread_width	value of the difference between the linearly interpolated and the actual value.
flag_spread_height	flag per object indicating whether the spread parameter in height dimension changes for this iframe_period.

nbits_spread_height	how many bits are required to represent the differential value minus 2.
differential_spread_height	value of the difference between the linearly interpolated and the actual value.
flag_spread_depth	flag per object indicating whether the spread parameter in depth dimension changes for this iframe_period.
nbits_spread_depth	how many bits are required to represent the differential value minus 2.
differential_spread_depth	value of the difference between the linearly interpolated and the actual value.
fixed_dynamic_object_priority	flag indicating whether the dynamic_object_priority is fixed for all objects.
flag_dynamic_object_priority	flag per object indicating whether the dynamic_object_priority changes for this iframe_period.
nbits_dynamic_object_priority	how many bits are required to represent the differential value minus 2.
differential_dynamic_object_priority	value of the difference between the linearly interpolated and the actual value.

7.4.2.1.4 Definition of offset_data() payloads

bitfield_syntax	flag indicating whether a vector with polygon time slice indices is present in the bitstream.
offset_bitfield	bool array containing ifperiod flags for each time slice whether they are polygon points or not.
npoints	number of polygon points minus 1 (num_points = **npoints** + 1).
foffset	time slice indices of the polygon points within iframe_period (frame_offset = **foffset**+1).

7.4.2.2 Decoding process

Each audio object has associated object metadata that describe the following properties:

- the temporal change of its position, given in spherical coordinates:
  - azimuth;
  - elevation;
  - radius;
- a linear gain that is to be applied by the object renderer;
- one or more values defining the object spread (either uniform or non-uniform);
- a value describing the dynamic object priority.

Each of these components is generally described by a discrete time signal y(n). In the following, the discrete time index n is used to denote the time slice to avoid confusion with the sample index of the audio signal that is sampled at a much higher sampling rate.

The general idea of encoding the object metadata is to segment y(n) into periods of a fixed length and to approximate these periods by means of polygons. Each transmitted frame contains at least one new polygon point, i.e. the final value. The OAM decoder always stores the last polygon point of a frame and uses it as a starting point for the next frame. This is important for the first frame since there is no previous value in this case.

Each OAM frame consists of iframe_period time slices which correspond to the audio frame length (per channel) of the core coder (in case, **hasCoreLength** is true).

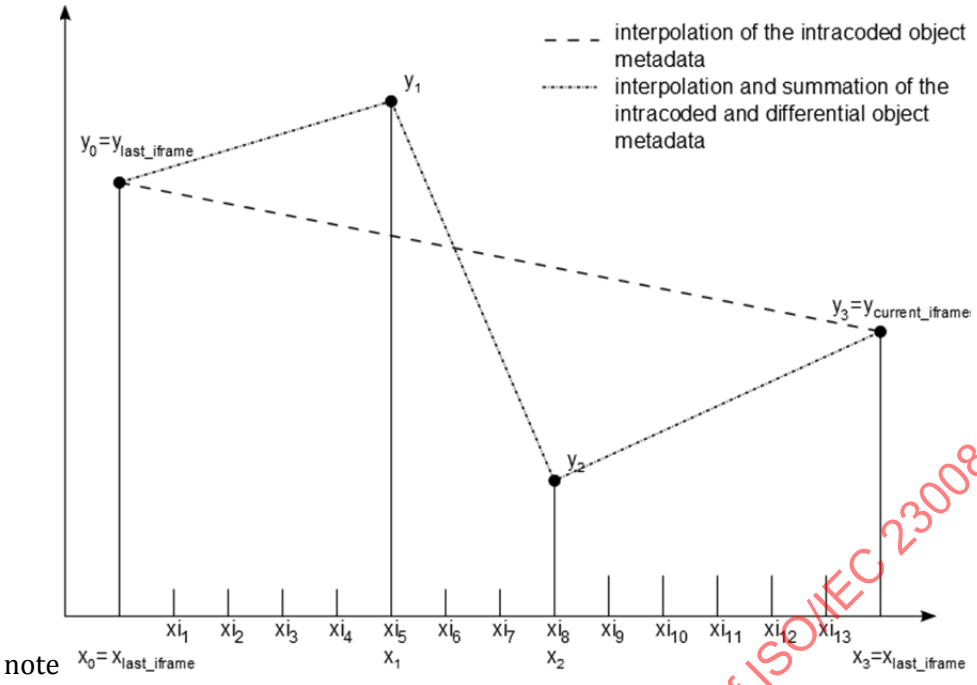
7.4.2.3 Decoding the transmitted data

First, as shown in Figure 43 by the dashed line, the values in between the last intracoded object metadata value y_{last_iframe} and the value of the current intracoded object metadata $y_{current\_iframe}$ are calculated for each component (azimuth elevation, radius, gain, spread and dynamic object priority) using linear interpolation.

When differential object metadata values are present in the bitstream, these are also used to calculate interpolated values. Those describe the difference between the interpolated intracoded object metadata values and the desired object metadata values. Hence, the interpolated output values *yi* are retrieved by summing up the components of the interpolated intracoded object metadata and – if present in the bitstream – the interpolated differential object metadata for each time slice *xi* (depicted in Figure 43 as the dotted line).

In Figure 43, there are num_points = 2 new polygon points given by differential_object_metadata() and the iframe_period is 14. The given polygon points are denoted as $y_{last\_iframe} = y_0$, y_1, y_2, and $y_{current\_iframe} = y_3$ which can be values for azimuth, elevation, radius, and gain. Each of these has a distinct time slice index which is denoted as $x_{last\_iframe} = x_0$, x_1, x_2 and $x_{current\_iframe} = x_3$. The indices of these given values are referred to as p = {0,1,2,3}.

The corresponding payload element name for time indices in case of differential object metadata is frame_offset[p] = **foffset**[p] +1 for p = {1, 2}. The indices xi_1 to xi_{14} represent the scale of the output time axis. For each *xi* an interpolated *yi* has to be calculated, as shown in Figure 44.



note

NOTE The dashed line depicts the first step (interpolation of intracoded object metadata). The dotted line shows the result of both interpolation steps.

Figure 43 — Interpolation in between the given polygon points

The positions (x-values) of the polygon points may be either given in the form of a sequence of integer values of the size num_points, indicated by **bitfield_syntax** = 0. They may also be given by the boolean array **offset_bitfield**, indicated by **bitfield_syntax** = 1. In the latter case, **offset_bitfield**[n] = 1 indicates the presence of a polygon point at time slice n. An **offset_bitfield** has the length of **ifperiod** time slices since the first polygon point is always known from the previous frame and polygon point for the last time slice is given by the intracoded object metadata.

The formula for the linear interpolation process is:

$$\alpha = \frac{xi_n - x_p}{x_{p+1} - x_p}$$

$$yi_n = (1 - \alpha) \cdot y_p + \alpha \cdot y_{p+1}$$

where xi_n is the current time index within the period $[x_p, x_{p+1}]$ that is specified by the frame_offset values. This yields the output depicted in Figure 44.

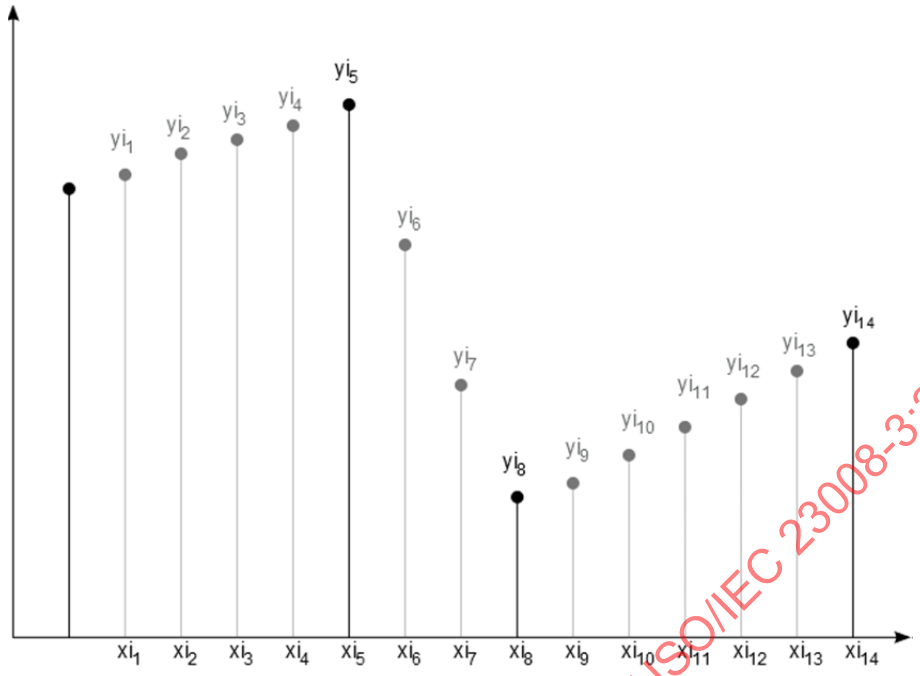


Figure 44 — Interpolated output values yi for each time slice xi

7.4.2.4 Post processing of object metadata

7.4.2.4.1 Scaling of object metadata

Apply scaling factors to revert encoder scaling of the input data for each component of every object.

```
descale_multidata()
{
    for (o = 0; o < num_objects; o++)
        azimuth[o] = azimuth[o] * 1.5;

    for (o = 0; o < num_objects; o++)
        elevation[o] = elevation[o] * 3.0;

    for (o = 0; o < num_objects; o++)
        radius[o] = pow(2.0, (radius[o] / 3.0)) / 2.0;

    for (o = 0; o < num_objects; o++)
        gain[o] = pow(10.0, (gain[o] - 32.0) / 40.0);

    if (uniform_spread == 1)
    {
        for (o = 0; o < num_objects; o++)
            spread[o] = spread[o] * 1.5;
    }
    else
    {
        for (o = 0; o < num_objects; o++)
            spread_width[o] = spread_width[o] * 1.5;

        for (o = 0; o < num_objects; o++)
            spread_height[o] = spread_height[o] * 3.0;

        for (o = 0; o < num_objects; o++)
```

```
            spread_depth[o] = (pow(2.0, (spread_depth[o] / 3.0)) / 2.0) - 0.5;
    }
    for (o = 0; o < num_objects; o++)
        dynamic_object_priority[o] = dynamic_object_priority[o];
}
```

7.4.2.4.2 Limiting the object metadata

Apply limiting to the decoded values to keep the values within a valid range.

```
limit_range()
{
    minval = -180;
    maxval =  180;
    for (o = 0; o < num_objects; o++)
        azimuth[o] = MIN(MAX(azimuth[o], minval), maxval);

    minval = -90;
    maxval =  90;
    for (o = 0; o < num_objects; o++)
        elevation[o] = MIN(MAX(elevation[o], minval), maxval);

    minval =    0.5;
    maxval = 16;
    for (o = 0; o < num_objects; o++)
        radius[o] = MIN(MAX(radius[o], minval), maxval);

    minval =    0.004;
    maxval = 5.957;
    for (o = 0; o < num_objects; o++)
        gain[o] = MIN(MAX(gain[o], minval), maxval);

    if (uniform_spread == 1)
    {
        minval =    0;
        maxval =  180;
        for (o = 0; o < num_objects; o++)
            spread[o] = MIN(MAX(spread[o], minval), maxval);
    }
    else
    {
        minval =    0;
        maxval =  180;
        for (o = 0; o < num_objects; o++)
            spread_width[o] = MIN(MAX(spread_width[o], minval), maxval);

        minval =   0;
        maxval =  90;
        for (o = 0; o < num_objects; o++)
            spread_height[o] = MIN(MAX(spread_height[o], minval), maxval);

        minval =   0;
        maxval =  15.5;
        for (o = 0; o < num_objects; o++)
            spread_depth[o] = MIN(MAX(spread_depth[o], minval), maxval);
    }
    minval = 0;
    maxval = 7;
    for (o = 0; o < num_objects; o++)
        dynamic_object_priority[o] = MIN(MAX(dynamic_object_priority[o], minval),
         maxval);
}
```

7.4.3 Object metadata decoding with low delay

7.4.3.1 Definition of object metadata payloads

7.4.3.1.1 Definition of object_metadata_low_delay() payloads

has_intracoded_object_metadata indicates whether the current frame is intracoded or differentially coded.

7.4.3.1.2 Definition of intracoded_object_metadata_low_delay() payloads

fixed_azimuth flag indicating whether the azimuth value is fixed for all object and not transmitted in case of dynamic_object_metadata().

default_azimuth defines the value of the fixed or common azimuth angle.

common_azimuth indicates whether a common azimuth angle shall be used for all objects.

position_azimuth if there is no common azimuth value, a value for each object is transmitted. If there is only one object, this is its azimuth angle.

fixed_elevation flag indicating whether the elevation value is fixed for all objects and not transmitted in case of dynamic_object_metadata().

default_elevation defines the value of the fixed or common elevation angle.

common_elevation indicates whether a common elevation angle shall be used for all objects.

position_elevation if there is no common elevation value, a value for each object is transmitted. If there is only one object, this is its elevation angle.

fixed_radius flag indicating whether the radius is fixed for all objects and not transmitted in case of dynamic_object_metadata().

default_radius defines the value of the fixed or common radius.

common_radius indicates whether a common radius value shall be used for all objects.

position_radius if there is no common radius value, a value for each object is transmitted. If there is only one object, this is its radius.

fixed_gain flag indicating whether the gain factor is fixed for all objects and not transmitted in case of dynamic_object_metadata().

default_gain defines the value of the fixed or common gain factor.

common_gain	indicates whether a common gain value shall be used for all objects.
gain_factor	if there is no common gain value, a value for each object is transmitted. If there is only one object, this is its gain factor.
fixed_spread	flag indicating whether the spread parameter is fixed for all objects and not transmitted in case of dynamic_object_metadata().
default_spread	defines the value of the fixed or common spread parameter.
default_spread_width	defines the value of the fixed or common spread parameter in the width dimension.
default_spread_height	defines the value of the fixed or common spread parameter in the height dimension.
default_spread_depth	defines the value of the fixed or common spread parameter in the depth dimension.
common_spread	indicates whether a common spread parameter shall be used for all objects.
spread	if there is no common spread parameter, a value for each object is transmitted. If there is only one object, this is its spread parameter.
spread_width	if there is no common spread parameter, a value for the spread in the width dimension is transmitted for each object. If there is only one object, this is its spread parameter in width dimension.
spread_height	if there is no common spread parameter, a value for the spread in the height dimension is transmitted for each object. If there is only one object, this is its spread parameter in height dimension.
spread_depth	if there is no common spread parameter, a value for the spread in the depth dimension is transmitted for each object. If there is only one object, this is its spread parameter in depth dimension.
fixed_dynamic_object_priority	flag indicating whether the dynamic_object_priority is fixed for all objects and not transmitted in case of dynamic_object_metadata().
default_dynamic_object_priority	defines the value of the fixed or common dynamic_object_priority.

common_dynamic_object_priority indicates whether a common dynamic_object_priority value shall be used for all objects.

dynamic_object_priority This field defines the priority of the object.

This field can take integer values between 0 and 7. The object may be discarded from rendering and decoding if the priority is lower than 7. If objects are discarded, the objects with lowest priority should be discarded first.

If there is only one object, this is its dynamic_object_priority.

7.4.3.1.3 Definition of dynamic_object_metadata() payloads

flag_absolute indicates whether the values of the components are transmitted differentially or in absolute values.

has_object_metadata indicates whether there are object metadata present in the bitstream or not.

7.4.3.1.4 Definition of single_dynamic_object_metadata() payloads

position_azimuth the absolute value of the azimuth angle if the value is not fixed.

position_elevation the absolute value of the elevation angle if the value is not fixed.

position_radius the absolute value of the radius if the value is not fixed.

gain_factor the absolute value of the gain factor if the value is not fixed.

spread the absolute value of the spread parameter if the value is not fixed.

spread_width the absolute value of the spread parameter in the width dimension if the value is not fixed.

spread_height the absolute value of the spread parameter in theheight dimension if the value is not fixed.

spread_depth the absolute value of the spread parameter in the depth dimension if the value is not fixed.

dynamic_object_priority the absolute value of the dynamic_object_priority if the value is not fixed.

nbits how many bits are required to represent the differential values.

flag_azimuth flag per object indicating whether the azimuth value changes.

position_azimuth_difference	difference between the previous and the active value.
flag_elevation	flag per object indicating whether the elevation value changes.
position_elevation_difference	value of the difference between the previous and the active value.
flag_radius	flag per object indicating whether the radius changes.
position_radius_difference	difference between the previous and the active value.
flag_gain	flag per object indicating whether the gain radius changes.
gain_factor_difference	difference between the previous and the active value.
flag_spread	flag per object indicating whether the spread parameter changes.
spread_difference	difference between the previous and the active value.
flag_spread_width	flag per object indicating whether the spread parameter in the width dimension changes.
spread_width_difference	difference between the previous and the active value.
flag_spread_height	flag per object indicating whether the spread parameter in the height dimension changes.
spread_height_difference	difference between the previous and the active value.
flag_spread_depth	flag per object indicating whether the spread parameter in the depth dimension changes.
spread_depth_difference	difference between the previous and the active value.
flag_dynamic_object_priority	flag per object indicating whether the dynamic_object_priority changes.
dynamic_object_priority_difference	difference between the previous and the active value.

7.4.3.2 Decoding process

7.4.3.2.1 General

Object metadata with low delay follows a modified DPCM procedure. It allows switching between differentially and absolutely coded values.

7.4.3.2.2 Decoding the transmitted data

In case intracoded object metadata are present in the bitstream, the object metadata values can be read from the bitstream directly. If dynamic object metadata are in the bitstream, the object metadata values can be read from the bitstream as well when **flag_absolute** is true. Otherwise, the following equation is to be used.

$$y[n] = y[n-1] + d[n]$$

where y denotes the output value, d is the difference value read from the bitstream and n is the active time slice for each component.

7.4.3.2.3 Post processing of object metadata

7.4.3.2.3.1 Scaling of object metadata

Apply a scaling factor to reverse the encoder scaling of the input data for each component of every object.

```
descale_multidata()
{
    for (o = 0; o < num_objects; o++)
        azimuth[o] = azimuth[o] * 1.5;

    for (o = 0; o < num_objects; o++)
        elevation[o] = elevation[o] * 3.0;

    for (o = 0; o < num_objects; o++)
        radius[o] = pow(2.0, (radius[o] / 3.0)) / 2.0;

    for (o = 0; o < num_objects; o++)
        gain[o] = pow(10.0, (gain[o] - 32.0) / 40.0);

    if (uniform_spread == 1)
    {
        for (o = 0; o < num_objects; o++)
            spread[o] = spread[o] * 1.5;
    }
    else
    {
        for (o = 0; o < num_objects; o++)
            spread_width[o] = spread_width[o] * 1.5;

        for (o = 0; o < num_objects; o++)
            spread_height[o] = spread_height[o] * 3.0;

        for (o = 0; o < num_objects; o++)
            spread_depth[o] = (pow(2.0, (spread_depth[o] / 3.0)) / 2.0) - 0.5;
    }
    for (o = 0; o < num_objects; o++)
        dynamic_object_priority[o] = dynamic_object_priority[o];
}
```

7.4.3.2.3.2 Limiting the object metadata

Apply limiting to the decoded values for each component of every object to keep the values within a valid range.

```
limit_range()
{
   minval = -180;
   maxval =  180;
   for (o = 0; o < num_objects; o++)
      azimuth[o] = MIN(MAX(azimuth[o], minval), maxval);

   minval = -90;
```

```
    maxval =  90;
    for (o = 0; o < num_objects; o++)
       elevation[o] = MIN(MAX(elevation[o], minval), maxval);

    minval =    0.5;
    maxval = 16;
    for (o = 0; o < num_objects; o++)
       radius[o] = MIN(MAX(radius[o], minval), maxval);

    minval =    0.004;
    maxval = 5.957;
    for (o = 0; o < num_objects; o++)
       gain[o] = MIN(MAX(gain[o], minval), maxval);

    if (uniform_spread == 1)
    {
       minval =    0;
       maxval =  180;
       for (o = 0; o < num_objects; o++)
          spread[o] = MIN(MAX(spread[o], minval), maxval);
    }
    else
    {
       minval =    0;
       maxval =  180;
       for (o = 0; o < num_objects; o++)
          spread_width[o] = MIN(MAX(spread_width[o], minval), maxval);

       minval =   0;
       maxval =  90;
       for (o = 0; o < num_objects; o++)
          spread_height[o] = MIN(MAX(spread_height[o], minval), maxval);

       minval =   0;
       maxval =  15.5;
       for (o = 0; o < num_objects; o++)
          spread_depth[o] = MIN(MAX(spread_depth[o], minval), maxval);
    }
    minval = 0;
    maxval = 7;
    for (o = 0; o < num_objects; o++)
       dynamic_object_priority[o] = MIN(MAX(dynamic_object_priority[o], minval),
        maxval);
}
```

7.4.4 Enhanced object metadata

7.4.4.1 Enhanced object metadata configuration semantics

hasDiffuseness — Flag which indicates whether diffuseness information is present in the payload frame.

hasCommonGroupDiffuseness — Flag which indicates whether the transmitted diffuseness information applies to a whole group or to individual objects.

hasExcludedSectors — Flag which indicates whether information about exclusion sectors is present in the payload frame.

hasCommonGroupExcludedSectors Flag which indicates whether the transmitted information about exclusion sectors applies to a whole group or to individual objects.

useOnlyPredefinedSectors This flag determines whether all exclusion sectors are chosen from the set of predefined sectors (value of 1) as identified by a table entry or whether sectors may also be signalled by means of explicit azimuth and elevation ranges in the bitstream (value of 0).

hasClosestSpeakerCondition If the 'closest loudspeaker playout flag' of the current group is set to 1, it is possible to restrict the processing to loudspeakers that are located in a specified area around the members of the group. This flag defines if the 'closest loudspeaker processing' shall happen unconditioned (value of 0) or conditioned (value of 1).

closestSpeakerThresholdAngle If the 'closest loudspeaker processing' shall only happen if one or more loudspeakers are located in a defined area around the members of the group, the threshold angle for this area is given by this field.

φ_{thresh} = 1.5 · closestSpeakerThresholdAngle;

φ_{thresh} = min (max (φ_{thresh} , 0), 180);

θ_{thresh} = 1.5 · closestSpeakerThresholdAngle;

θ_{thresh} = min (max (θ_{thresh} , 0), 90);

hasDivergence Flag which indicates whether divergence information is present in the payload frame.

divergenceAzimuthRange If the divergence of the object or group is larger than 0.0 (divergence > 0), the divergenceAzimuthRange defines the positioning of the virtual sources. The field can take values between 0 and 63, resulting in azimuth offset angles between 0° and 180°:

φ_{offset} = min(max(3.0 · divergenceAzimuthRange, 0),180);

7.4.4.2 Enhanced object metadata frame semantics

keepDiffuseness Flag which indicates whether the diffuseness from the previous frame shall be re-used for the current frame (value of 0) or whether a new diffuseness shall be transmitted in the subsequent bitstream field.

diffuseness Definition of the diffuseness of the objects (of a group); this field can take values between 0 and 127, corresponding to diffuseness values between 0.0 and 1.0:

diffuseness = (diffuseness / 127);

keepExclusion Flag which indicates whether the exclusion sectors from the previous frame shall be re-used for the current frame (value of 0) or whether new exclusion sectors shall be transmitted in the subsequent bitstream fields.

numExclusionSectors This field defines the number of sectors/areas that shall be excluded from rendering the of the affected object(s). It allows for values between 0 and 15. A value of 0 indicates that no loudspeakers shall be excluded.

usePredefinedSector This flag defines if the following sector is a predefined one (value of 1) identified by a table entry or if a detailed sector definition by azimuth and elevation ranges follows in the bitstream (value of 0).

excludeSectorIndex Identifier of the predefined exclusion sector as defined in Table 146.

Table 146 — Value of excludeSectorIndex

Value of excludeSectorIndex	Short description	Explanation
0	No positive elevation	Exclude all loudspeaker with positive elevation angles
1	No negative elevation	Exclude all loudspeakers with negative elevation angles
2	No front	Exclude all front loudspeakers
3	No right side	Exclude all right side loudspeakers
4	No left side	Exclude all left side loudspeakers
5	No surround	Exclude all surround loudspeakers
6	Screen only	Exclude all loudspeakers that are not located in the reproduction screen area
7-15	Reserved	n/a

excludeSectorMinAzimuth This field defines the minimum azimuth of the excluded area.

$\varphi_{sector,min}$ = 3.0 · (excludeSectorMinAzimuth - 63);

$\varphi_{sector,min}$ = min (max ($\varphi_{sector,min}$, -180), 180);

excludeSectorMaxAzimuth This field defines the maximum azimuth of the excluded area.

$\varphi_{sector,max}$ = 3.0 · (excludeSectorMaxAzimuth - 63);

$\varphi_{sector,max}$ = min (max ($\varphi_{sector,max}$, -180), 180);

excludeSectorMinElevation This field defines the minimum elevation of the excluded area.

$\theta_{sector,min}$ = 6.0 · (excludeSectorMinElevation - 15);

$\theta_{sector,min}$ = min (max ($\theta_{sector,min}$, -90), 90);

excludeSectorMaxElevation This field defines the maximum elevation of the excluded area.

$\theta_{sector,max}$ = 6.0 · (excludeSectorMaxElevation - 15);

$\theta_{sector,max}$ = min (max ($\theta_{sector,max}$, -90), 90);

closestSpeakerPlayout This flag defines that the object shall not be rendered with the object renderer but instead directly be played back by the loudspeaker which is nearest to the geometric position of the object as defined in subclause 18.6.

keepDivergence Flag which indicates whether the divergence from the previous frame shall be re-used for the current frame (value of 0) or whether a new divergence shall be transmitted in the subsequent bitstream field.

divergence This field defines the divergence of the objects (of a group). The field can take values between 0 and 127, corresponding to divergence values between 0.0 and 1.0:

divergence = (divergence / 127);

8 Object rendering

8.1 Description

The audio object rendering is carried out as a process to convert/render the object based audio signals into a channel based representation.

8.2 Terms and definitions

$\hat{\mathbf{p}}$	Unit length vector representing the direction of the audio object.
$\hat{\mathbf{l}}_n$	Unit length vector representing the direction of loudspeaker n.
$\mathbf{L}_{abc}$	Triplet of unit length loudspeaker vectors $\hat{\mathbf{l}}_a, \hat{\mathbf{l}}_b, \hat{\mathbf{l}}_c$ in matrix form.
r	Object position: radius in [m].
θ	Object position: elevation angle in [°].
φ	Object position: azimuth angle in [°].
$\alpha, \alpha_{width}, \alpha_{height}$	Spread parameter in [°].
a	Audio object linear gain factor.
$\mathbf{X}$	Matrix with audio samples of all objects (rows: objects, columns: samples).
$\mathbf{Y}$	Matrix with loudspeaker samples (rows: loudspeakers, columns: samples).
Audio object	Audio + metadata.
VBAP	Vector base amplitude panning [1].

8.3 Input data

Audio objects consist of audio data $_{x}$ and metadata. Metadata is conveyed for every audio object at defined timestamps. The metadata consists of the following data organized per each audio object:

— Spherical coordinates with radius r [m], elevation angle θ [°] and azimuth angle φ[°]:

$\mathbf{s} = \begin{bmatrix} r \\ \theta \\ \varphi \end{bmatrix}$ (see Figure 45).

— Linear gain factor a.

— Uniform spread parameter $\alpha \in [0°, 180°]$ or non-uniform spread parameters $\alpha_{\text{width}} \in [0°,180°]$ and $\alpha_{\text{height}} \in [0°, 90°]$.

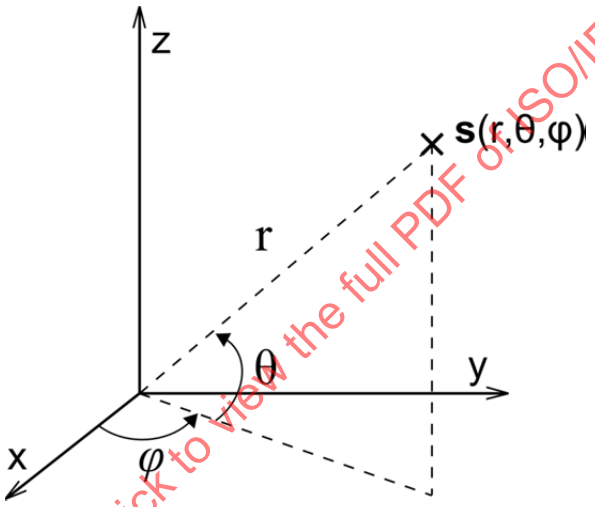


Figure 45 — Polar coordinate system used to specify the object position

Additionally, the following definitions apply.

— Forward looking (frontal direction) is along the x-axis:$\theta = 0°$, $\varphi = 0°$.

— Azimuth (φ) angle increases with counter clockwise rotation;

— $0° \le \varphi \le 180°$ Left hemisphere.

— $-180° \le \varphi \le 0°$ Right hemisphere.

— $0° \le \theta \le 90°$ Upper hemisphere.

— $-90° \le \theta \le 0°$ Lower hemisphere.

The calculations described here require Cartesian coordinates. For this purpose, the coordinates are converted according to:

$$x = r\cos(\theta)\cos(\varphi)$$
$$y = r\cos(\theta)\sin(\varphi)$$
$$z = r\sin(\theta)$$

8.4 Processing

8.4.1 General remark

Object rendering shall be restricted to non-LFE output channels, i.e. the object renderer shall ignore LFE channels in the target loudspeaker configuration, if present. LFE channels in the target loudspeaker configuration shall thus neither be taken into account in the triangulation, as defined in subclause 8.4.3, nor in the rendering, as defined in subclause 8.4.4.

8.4.2 Imaginary loudspeakers

Not all considered loudspeaker configurations are complete 3D setups that cover all solid angles. Hence, with regard to the triangulation that is required for the VBAP rendering algorithm, there are voids in the triangulation surface which may be considered as invalid solid angles. These voids are filled by adding imaginary loudspeakers. The object renderer first computes the panning gains for the extended set of loudspeakers which also includes these imaginary loudspeakers (see subclause 8.4.4). In a second step, a downmix matrix is applied to the gain vector which equally distributes the sound energy of each imaginary loudspeaker among his neighbours by applying a weighting factor of 1/sqrt(N) where N is the number of neighbours. Finally, the down-mixed gain vector is power normalized.

Imaginary loudspeakers are added according to the following rules. In the definition of the loudspeaker sub-sets, “A”, “B”, “C”, “D”, the tolerances defined in Table 162 shall be taken into account when matching the actual loudspeaker positions (azimuth and elevation angles) to the channel labels.

1) If no loudspeaker exists at or above 45° elevation, add an imaginary loudspeaker at [0°,90°].

2) If no loudspeaker exists at or below -45° elevation, add an imaginary loudspeaker at [0°,-90°].

3) If exactly one of the loudspeaker sub-sets

   — sub-set A: {CH_M_L030, CH_M_R030, CH_U_L030, CH_U_R030}

   or

   — sub-set B: {CH_M_L045, CH_M_R045, CH_U_L045, CH_U_R045}

   exists, but no other loudspeaker exists within or on the edges of the quadrilateral defined by the actual positions of the 4 sub-set loudspeakers, then add an imaginary loudspeaker at the mean azimuth angle of those 4 sub-set loudspeakers and the mean elevation angle of those 4 sub-set loudspeakers, where the mean angles shall be derived from the actual azimuth and elevation angles of those 4 sub-set loudspeakers.

4) If exactly one of the sub-sets

   — sub-set C: {CH_M_L110, CH_M_R110, CH_U_L110, CH_U_R110}

   or

   — sub-set D: {CH_M_L135, CH_M_R135, CH_U_L135, CH_U_R135}

   exists, but no other loudspeaker exists within or on the edges of the quadrilateral defined by the actual positions of the 4 sub-set loudspeakers, then add an imaginary loudspeaker at the mean azimuth angle of those 4 sub-set loudspeakers plus 180° and the mean elevation angle of those 4

sub-set loudspeakers, where the mean angles shall be derived from the actual azimuth and elevation angles of those 4 sub-set loudspeakers.

5) Sort all loudspeakers with an absolute elevation angle smaller than 45° according to their azimuth angle and fill gaps greater than 160° by the minimum number of equally spaced imaginary loudspeakers with 0° elevation.

The neighbourhood of an imaginary loudspeaker is defined by the edges of the triangulation mesh specified in subclause 8.4.3.2. If an edge exists between a loudspeaker and the considered imaginary loudspeaker, then this loudspeaker is a neighbour.

The purpose of the downmix matrix is to eliminate the added imaginary loudspeakers and to restrict the calculated gains to the existing loudspeakers. If an imaginary loudspeaker is the neighbour of another imaginary loudspeaker, then the gains cannot be reduced to the existing loudspeakers. This problem is solved by the following approach: First, an energy distribution matrix $\boldsymbol{D}$ is constructed where the elements $d_{ij} = 1/N_i$ specifies the amount of energy which is re-distributed from the imaginary loudspeaker i to loudspeaker j where N_i denotes the number of neighbours. Column vectors which belong to an existing loudspeaker have only one non-zero value $d_{ii} = 1$. In each iteration step we then multiply this matrix by $\boldsymbol{D}$. This corresponds to re-distributing energy portions to the neighbours according to the given weights. This is repeated until all elements which belong to the imaginary loudspeakers are less than or equal to 10^{-4} (≤-40dB). The element-wise square root finally yields the elements of the downmix matrix. Mathematically, the whole process is given as follows:

$$\boldsymbol{M} = \mathrm{sqrt}(\boldsymbol{D}^n)$$

where n denotes the number of iterations, sqrt(•) denotes the element-wise square root, and $\boldsymbol{M}$ denotes the resulting downmix matrix.

NOTE Faster convergence is achieved by computing $\boldsymbol{D}^{(2^n)}$ instead of $\boldsymbol{D}^n$. This means that the renderer does not multiply the iteration result by $\boldsymbol{D}$, but by itself.

8.4.3 Dividing the loudspeaker setup into a triangle mesh

8.4.3.1 General

To calculate 3D VBAP, a triangulation of the convex hull around the given loudspeaker setup is required. Within this hull loudspeaker triplets are defined that consist of three adjacent loudspeaker positions in 3D space. Each loudspeaker triplet shall fulfil the following requirements in order to generate useful results.

— The vectors from the listener position to the triangle corners shall be linearly independent.

— Shall not intersect with any other loudspeaker triplet.

The triangulation in 3D space can be performed in several different ways and there are many ways to divide a loudspeaker setup into triangles. Please note that the additional triangles for the areas covered by the imaginary loudspeakers are defined by the imaginary loudspeaker and two successive neighbours.

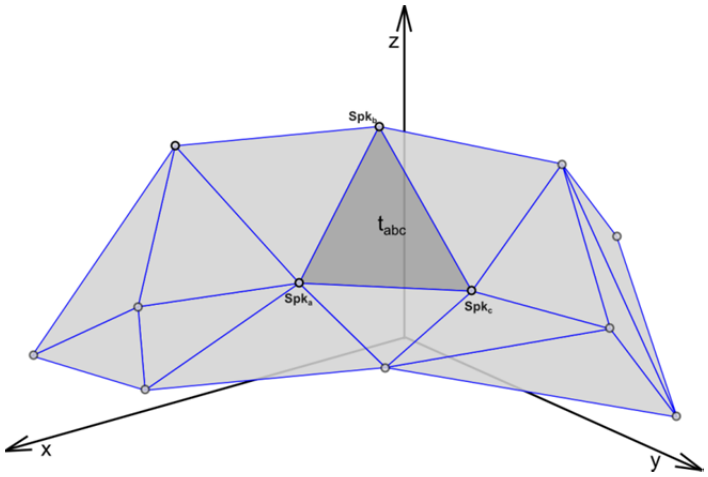


Figure 46 — Triangulation example – Triplet t_{abc} defined by Spk_a, Spk_b, Spk_c, vectors

8.4.3.2 Automatic triangulation

The triangulation mesh is determined by means of a Delaunay triangulation algorithm. As all vertices, i.e. the loudspeaker positions, are located on a sphere surface, the Delaunay solution can be found by calculating the convex hull of the given vertex set. The automatic triangulation algorithm uses the QuickHull algorithm which has been extended to yield left-right and front-back symmetric triangulation meshes.

1) Sort all vertices in ascending order according to the vertex index specified in the following pseudo code.

```
function index = vertex_index(azimuth, elevation)
{
   azimuth   = 180 - mod(180 - azimuth, 360);
   elevation = max(-90, min(90, elevation));
   idx_azi   = round(abs(90 - abs(azimuth)));
   idx_ele   = round(abs(elevation));
   index     = idx_azi + 181 * idx_ele;
}
```

where the modulo operator mod(x,y) returns values in the range [0,y[according to:

mod(x,y) = x - n*y, where n = floor(x/y)

with floor(i) rounding i to the nearest integer towards minus infinity.

2) Choose any sub-set of the vertices with a convex hull as initial polyhedron.

3) Extend the initial polyhedron sequentially by the sorted list of vertices. If the new vertex is located outside of the polyhedron or on its surface, replace those surface patches by the triangles that are defined by the new vertex and the border (the outer edges) of those surface patches.

   As a result, the most recently added vertex defines the sub-division of a planar polyeder into triangles.

8.4.4 Rendering algorithm

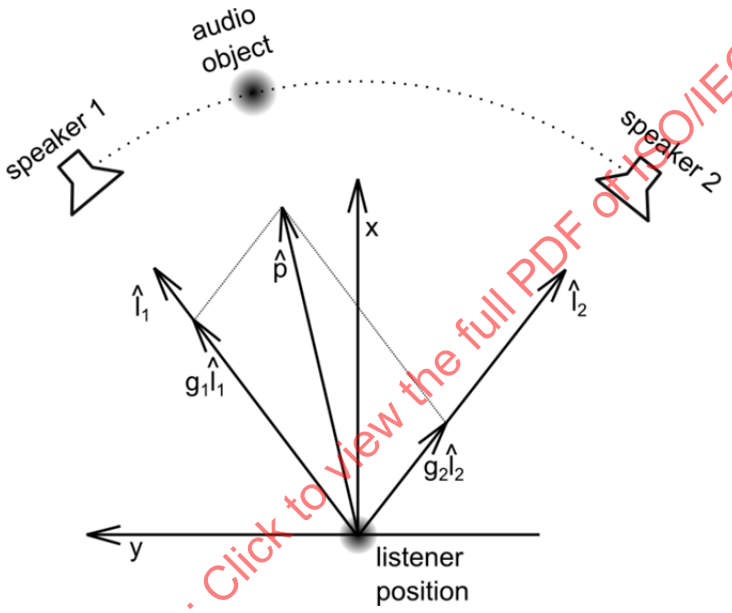
8.4.4.1 General

The rendering is based on VBAP. The algorithm and descriptions are based on [1]. As all setups are extended by imaginary loudspeakers to a complete 3D setup, 3D VBAP shall be used for rendering.

An example of a realization of distance and spread depth rendering can be found in Annex K.

8.4.4.2 3D VBAP

A triplet-wise panning shall be applied for 3D setups and for all other setups. This is because all setups are extended by imaginary loudspeakers, if necessary. For this, the audio object is applied to a maximum of three loudspeakers. All calculations are performed for each loudspeaker triplet. The triplets are defined by the loudspeaker triangulation (see subclause 8.4.3).



NOTE For the sake of simplicity, here the 2D VBAP case is shown with two instead of 3 base vectors.

Figure 47 — Audio object described as linear combination of loudspeaker vectors

The audio object vector $\hat{\mathbf{p}}$ is expressed as a linear combination of the loudspeaker triplet $\hat{\mathbf{l}}_1, \hat{\mathbf{l}}_2, \hat{\mathbf{l}}_3$:

$$\hat{\mathbf{p}} = g_1\hat{\mathbf{l}}_1 + g_2\hat{\mathbf{l}}_2 + g_3\hat{\mathbf{l}}_3$$

In matrix form and transposed, gain factors can be calculated using the formula:

$$g = \hat{\mathbf{p}}^{\mathrm{T}}\mathbf{L}_{123}^{-1} = \begin{bmatrix} p_1 & p_2 & p_3 \end{bmatrix} \begin{bmatrix} l_{11} & l_{12} & l_{13} \\ l_{21} & l_{22} & l_{23} \\ l_{31} & l_{32} & l_{33} \end{bmatrix}^{-1}$$

The loudspeaker triangle which contains the panning direction is found by sequentially calculating g_1, g_2, g_3 for the list of triplets until all gain factors are greater than or equal to zero. This set yields the vector:

$$\boldsymbol{g} = [g_1, g_2, g_3].$$

Finally, the audio object's gain factor $_a$ is applied and power normalization is performed:

$$\boldsymbol{g}_{\text{scaled}} = \frac{a\boldsymbol{g}}{\|\boldsymbol{g}\|}$$

This is the vector which is then used to construct the rendering matrix $\boldsymbol{G}$.

8.4.4.3 Audio processing

The audio processing function transforms audio objects into channel based loudspeaker signals.

Let **X** be the audio samples of all objects in matrix representation with dimension *O*- number of objects, *N*- number of audio samples per audio processing block. Each row contains the audio samples of one object.

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,N} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{O,1} & x_{O,2} & \cdots & x_{O,N} \end{bmatrix}$$

Let **G** be the gain factors of all audio objects and loudspeakers in matrix representation with dimension *O*- number of objects, *S*- number of loudspeakers. Each column contains all gain factors for one audio object.

$$\mathbf{G} = \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,O} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,O} \\ \vdots & \vdots & \ddots & \vdots \\ g_{S,1} & g_{S,2} & \cdots & g_{S,O} \end{bmatrix}$$

NOTE There will be a maximum of three non-zero values per column.

Matrix **Y** contains the output samples that can be calculated with the formula:

Y = **GX**

Each row in **Y** contains the output audio samples for one loudspeaker.

8.4.4.4 Gain factor crossfading

Metadata is conveyed for every audio object at defined timestamps. To achieve a smooth transition when metadata changes, the **G** matrices are interpolated linearly between adjacent timestamps and applied on a per sample basis.

8.4.4.5 Audio processing in QMF-domain

Alternatively, object rendering may be performed in the QMF-domain using QMF-transformed audio signals. In the QMF-domain, the input data is given by a three-dimensional matrix with complex elements $x^k_{o,n}$ where *o* denotes the object index, *n* denotes the slot index, and *k* denotes the band index. As the VBAP panning gains are frequency independent, object rendering in the QMF-domain is realized analogously to the time domain processing,

$\boldsymbol{Y}_k = \boldsymbol{G}\boldsymbol{X}_k$,

where

$$X_k = \begin{bmatrix} x^k_{1,1} & x^k_{1,2} & \cdots & x^k_{1,N} \\ x^k_{2,1} & x^k_{2,2} & \cdots & x^k_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ x^k_{O,1} & x^k_{O,2} & \cdots & x^k_{O,N} \end{bmatrix}$$

denotes the two-dimensional sub-set of the input data matrix that belongs to QMF band *k*. The matrix

$$Y_k = \begin{bmatrix} y^k_{1,1} & y^k_{1,2} & \cdots & y^k_{1,N} \\ y^k_{2,1} & y^k_{2,2} & \cdots & y^k_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ y^k_{S,1} & y^k_{S,2} & \cdots & y^k_{S,N} \end{bmatrix}$$

denotes the two-dimensional sub-set of the output data matrix with complex elements $y^k_{s,n}$ where *s* denotes the loudspeaker index, *n* denotes the slot index, and *k* denotes the band index. The gain factor cross-fading is realized analogously to the time domain processing with the only difference that the cross-fading relates to slot indices rather than sample indices.

8.4.4.6 Time-alignment

Object rendering requires object metadata side information embedded into a MPEG-H 3D audio bitstream (see Clause 15) in combination with the audio signals which are associated with each audio object. Both data streams shall be time-aligned for proper rendering. This time-alignment is realized by the encoder such that the first frame of the current decoded object metadata, which specifies the metadata values over a period of *iframe_period* frames, is applied to the current audio data frame.

8.4.4.7 Spreading

The object spreading specified in the following subclause is applied if all of the following conditions are satisfied.

Audio objects with a spatial extent (spread $\alpha > 0.0°$ (uniform spread) or spread_width $\alpha_{\text{width}} > 0.0°$ (non-uniform spread)) are processed by means of the multiple direction amplitude panning (MDAP) method [7]. This method involves the computation of a set of panning gains $g_{\text{scaled,m}}$ for $M = 18$ MDAP directions p_m around the panning direction $p_0 = \hat{p}$.

The determination of the MDAP directions requires the computation of two base vectors, u and v. If the loudspeaker setup contains height loudspeakers, these vectors are computed as follows:

$$v = \begin{cases} \text{cart}(\phi, \theta + 90^\circ, 1), & \theta < 0^\circ \\ \text{cart}(\phi, \theta - 90^\circ, 1), & \theta \geq 0^\circ \end{cases}$$

$$u = v \times p_0$$

where cart(·) denotes the transform from spherical coordinates to Cartesian coordinates and × denotes the cross product. Otherwise, u and v are computed as follows:

$$w = \begin{cases} \text{cart}(\phi, \theta + 90^\circ, 1), & \theta < 0^\circ \\ \text{cart}(\phi, \theta - 90^\circ, 1), & \theta \geq 0^\circ \end{cases}$$

$$u = w \times p_0$$

$$v = [0, \ldots, 0]^T$$

If non-uniform spread values are transmitted, the ratio of the spread parameter in the width direction α_{width} and the spread parameter in the height direction α_{height} are used to determine the ratio.

$$a_{r=} \frac{\alpha_{\text{height}}}{\alpha_{\text{width}}}$$

This ratio shall be used to scale the base vector v:

$$v = v \cdot a_r .$$

The MDAP directions are then computed from 18 pattern vectors p_m':

$$\begin{aligned}
p_1' &= u \\
p_2' &= 0.75\,u + 0.25\,p_0 \\
p_3' &= 0.375\,u + 0.625\,p_0 \\
p_4' &= -u \\
p_5' &= -0.75\,u + 0.25\,p_0 \\
p_6' &= -0.375\,u + 0.625\,p_0 \\
p_7' &= 0.5\,u + 0.866\,v + p_0/3 \\
p_8' &= 0.5\,p_7' + 0.5\,p_0 \\
p_9' &= 0.25\,p_7' + 0.75\,p_0 \\
p_{10}' &= -0.5\,u + 0.866\,v + p_0/3 \\
p_{11}' &= 0.5\,p_{10}' + 0.5\,p_0 \\
p_{12}' &= 0.25\,p_{10}' + 0.75\,p_0 \\
p_{13}' &= -0.5\,u - 0.866\,v + p_0/3 \\
p_{14}' &= 0.5\,p_{13}' + 0.5\,p_0
\end{aligned}$$

$$p_{15}' = 0.25\, p_{13}' + 0.75\, p_0$$
$$p_{16}' = 0.5\, u - 0.866\, v + p_0/3$$
$$p_{17}' = 0.5\, p_{16}' + 0.5\, p_0$$
$$p_{18}' = 0.25\, p_{16}' + 0.75\, p_0$$

Together with the spread parameter α (uniform spread) or the spread parameter α_{width} (non-uniform spread), this yields the MDAP directions p_m:

$$p_m = p_m' + \frac{p_0}{\tan(\alpha')}$$

where α' is equal to α limited to$[0.001^\circ, 90^\circ]$, or respectively to α_{width} limited to $[0.001^\circ, 90^\circ]$ for the transmission of non-uniform spread. The normalization to unit norm is not necessary since the normalization to unit norm is performed when $g_{scaled,m}$ is computed for the MDAP directions p_m.

If the spread parameter α (or respectively α_{width} for non-uniform spread) exceeds 90°, then a cross-fading towards power-normalized unit gain $1/\sqrt{N}$ for all of the N loudspeakers is applied to yield the final panning gains:

$\lambda = \frac{\alpha - 90^\circ}{90^\circ}$, or $\lambda = \frac{\alpha_{width} - 90^\circ}{90^\circ}$ (non-uniform spread) respectively.

$$g_{MDAP}' = (1 - \lambda)\frac{\sum_{m=0}^{M} g_{scaled,m}}{\left\|\sum_{m=0}^{M} g_{scaled,m}\right\|} + \lambda\, [1, \ldots, 1]^T/\sqrt{N}$$

$$g_{MDAP} = a\frac{g_{MDAP}'}{\|g_{MDAP}'\|}$$

Otherwise, the final panning gains are computed as follows:

$$g_{MDAP} = a\frac{\sum_{m=0}^{M} g_{scaled,m}}{\left\|\sum_{m=0}^{M} g_{scaled,m}\right\|}$$

9 SAOC 3D

9.1 Description

Spatial audio object coding 3D audio reproduction (SAOC 3D) shall be based on MPEG SAOC technology specified in ISO/IEC 23003-2. The SAOC 3D technology is used to enable interactive rendering functionality for audio object-based content.

9.2 Definitions

K	is the number of hybrid subbands
L	is the number of parameter sets
N_{dec}	is the number of used decorrelators
M_{proc}	is the number of processing bands

M_{QMF}	is the number of QMF subbands depending on sampling frequency
N	is the number of SAOC 3D input signals (channels and objects)
N_{ch}	is the number of SAOC 3D input channels
N_{obj}	is the number of SAOC 3D input audio objects
N_{dmx}	is the number of SAOC 3D downmix signals (channel and object signals)
$N_{\text{ch}}^{\text{dmx}}$	is the number of SAOC 3D downmix signals for channel inputs
$N_{\text{obj}}^{\text{dmx}}$	is the number of SAOC 3D downmix signals object inputs
N_{premix}	is the number of premix channels
N_{out}	is the number of SAOC 3D output channels
N_{g}^{q}	is the number of downmix signals assigned to group $\mathbf{g}_q$, defined for all group indices q
$\mathbf{g}_q$	is a vector with the indices of the downmix signals assigned to the same group, defined for all group indices q
$\mathbf{R}^{l,m}$	is the time and frequency variant rendering matrix, defined for all parameter time slots l and all processing bands m
$\mathbf{s}^{n,k}$	is a vector with the hybrid subband (encoder) input channels, defined for all time slots n and all hybrid subbands k
$\mathbf{x}^{n,k}$	is a vector with the hybrid subband (decoder) input signals (downmix), defined for all time slots n and all hybrid subbands k
$\hat{\mathbf{y}}^{n,k}$	is a vector with the (decoder) output hybrid subband signals, which are fed into the hybrid synthesis filter banks, defined for all time slots n and all hybrid subbands k
$\mathbf{y}_{\text{dry}}^{n,k}$	is a vector with the parametrically estimated hybrid subband signals, defined for all time slots n and all hybrid subbands k
$\mathbf{y}_{\text{wet}}^{n,k}$	is a vector with the decorrelated hybrid subband signals, defined for all time slots n and all hybrid subbands k
$\mathbf{D}$	is the downmixing matrix
$\mathbf{D}_{\text{DMG}}$	is the three dimensional matrix holding the dequantized, and mapped DMG data for every input signal, downmix channel, and parameter set
$\mathbf{D}_{\text{IOC}}$	is the four dimensional matrix holding the dequantized, and mapped IOC data for every input channel pair, every parameter set, and M_{proc} bands

$\mathbf{D}_{OLD}$	is the three dimensional matrix holding the dequantized, and mapped OLD data for every input channel, every parameter set, and M_{proc} bands
$\mathbf{m} = diag(\mathbf{M})$	is the main diagonal of matrix **M**
$\mathbf{H} = matdiag(\mathbf{M})$	is a matrix containing the elements from the main diagonal of matrix **M** on the main diagonal and zero values on the off-diagonal positions
ε	is a constant used to avoid division by zero $\varepsilon = 10^{-9}$
SAOC 3D	Spatial audio object coding for 3D audio reproduction

9.3 Delay and synchronization

The SAOC 3D decoder introduces a delay when processing the time domain signal coming from a downmix decoder. The transmission of the SAOC 3D side information with respect to the transmission of the coded downmix signal is performed in such a manner that there is no need to delay the downmix signal further before the SAOC processing.

9.4 Syntax

9.4.1 Payloads for SAOC 3D

Table 147 — Syntax of SAOC3DSpecificConfig()

Syntax	No. of bits	Mnemonic
SAOC3DSpecificConfig()		
{		
bsSamplingFrequencyIndex;	**4**	**uimsbf**
if (bsSamplingFrequencyIndex == 15) {		
bsSamplingFrequency;	**24**	**uimsbf**
}		
bsFreqRes;	**3**	**uimsbf**
bsDoubleFrameLengthFlag;	**1**	**uimsbf**
bsNumSaocDmxChannels;	**5**	**uimsbf**
bsNumSaocDmxObjects;	**5**	**uimsbf**
bsDecorrelationMethod;	**1**	**uimsbf**
NumInputSignals = 0;		
if (bsNumSaocDmxChannels > 0) {		
saocChannelLayout = SpeakerConfig3d();		
NumSaocChannels = SAOC3DgetNumChannels(saocChannelLayout);		[a]
NumInputSignals += NumSaocChannels;		
}		
bsNumSaocObjects;	**8**	**uimsbf**
NumInputSignals += bsNumSaocObjects;		
for (i=0; i<NumSaocChannels; i++) {		
bsRelatedTo[i][i] = 1;		
for(j=i+1; j< NumSaocChannels; j++) {		
bsRelatedTo[i][j];	**1**	**uimsbf**
bsRelatedTo[j][i] = bsRelatedTo[i][j];		
}		
}		
for (i=NumSaocChannels; i<NumInputSignals; i++) {		

Syntax	No. of bits	Mnemonic
for(j=0; j<NumSaocChannels; j++) {		
bsRelatedTo[i][j] = 0;		
bsRelatedTo[j][i] = 0;		
}		
}		
for (i=NumSaocChannels; i<NumInputSignals; i++) {		
bsRelatedTo[i][i] = 1;		
for(j=i+1; j<NumInputSignals; j++) {		
bsRelatedTo[i][j];	**1**	**uimsbf**
bsRelatedTo[j][i] = bsRelatedTo[i][j];		
}		
}		
bsOneIOC;	**1**	**uimsbf**
bsSaocDmxMethod;	**1**	**uimsbf**
if (bsSaocDmxMethod == 1) {		
NumPremixedChannels = SAOC3DgetNumChannels(referenceLayout);		[a, b]
}		
bsDualMode;	**1**	**uimsbf**
if (bsDualMode) {		
bsBandsLow;	**5**	**uimsbf**
bsBandsHigh = numBands;		[c]
} else {		
bsBandsLow = numBands;		
}		
bsDcuFlag;	**1**	**uimsbf**
if (bsDcuFlag == 1) {		
bsDcuMandatory;	**1**	**uimsbf**
bsDcuDynamic;	**1**	**uimsbf**
if (bsDcuDynamic == 0) {		
bsDcuMode;	**1**	**uimsbf**
bsDcuParam;	**4**	**uimsbf**
}		
} else {		
bsDcuMandatory = 0;		
bsDcuDynamic = 0;		
bsDcuMode = 0;		
bsDcuParam = 0;		
}		
ByteAlign();		
SAOC3DExtensionConfig();		
}		

[a] SAOC3DgetNumChannels() defines the number of SAOC 3D input channels from data obtained by the bitstream syntax element SpeakerConfig3d().
[b] referenceLayout is defined in subclause 5.3.2.
[c] numBands shall be as defined in ISO/IEC 23003-2.

Table 148 — Syntax of SAOC3DgetNumChannels()

Syntax	No. of bits	Mnemonic

SAOC3DgetNumChannels(Layout)
{
numChannels = numSpeakers;
for (i = 0; i < numSpeakers; i++) {
if (Layout.isLFE[i] == 1) {
numChannels = numChannels - 1;
}
}
return numChannels;
}
		a
		
		b

a The function SAOC3DgetNumChannels() returns the number of available non-LFE channels numChannels. b numSpeakers is defined in Syntax of SpeakerConfig3d(). If speakerLayoutType == 0, numSpeakers shall represent the number of loudspeakers corresponding to the ChannelConfiguration value, CICPspeakerLayoutIdx, as defined in ISO/IEC 23001-8.		

Table 149 — Syntax of Saoc3DFrame()

Syntax	No. of bits	Mnemonic

Saoc3DFrame()
{
SAOC3DFramingInfo();
bsIndependencyFlag**;**
for(i=0; i<NumInputSignals; i++) {
idxOLD[i] = EcDataSaoc(OLD, i, numBands);
}
k=0;
iocIdx1=0;
iocIdx2=0;
for(i=0; i<NumInputSignals; i++) {
idxIOC[i][i] = 0;
for(j=i+1; j<NumInputSignals; j++) {
if (bsRelatedTo[i][j] != 0) {
if (bsOneIOC == 0) {
idxIOC[i][j] = EcDataSaoc(IOC, k, numBands);
k++;
} else {
if (k == 0)
idxIOC[i][j] = EcDataSaoc(IOC, k, numBands);
k++;
iocIdx1=i;
iocIdx2=j;
} else {
idxIOC[i][j] = idxIOC[iocIdx1][iocIdx2];
}
}
} else {
idxIOC[i][j] = 5;
}
idxIOC[j][i] = idxIOC[i][j];
}
}
	1
	
	a
		uimsbf

Syntax	No. of bits	Mnemonic
if (bsNumSaocDmxObjects==0) {		
for(i=0; i< bsNumSaocDmxChannels; i++) {		
idxDMG[i] = EcDataSaoc(DMG, 0, NumInputSignals);		
}		
} else {		
dmgIdx = 0;		
for(i=0; i<bsNumSaocDmxChannels; i++) {		
idxDMG[i] = EcDataSaoc(DMG, 0, NumSaocChannels);		
}		
dmgIdx = bsNumSaocDmxChannels;		
if (bsSaocDmxMethod == 0) {		
for(i=dmgIdx; i<dmgIdx + bsNumSaocDmxObjects; i++) {		
idxDMG[i] = EcDataSaoc(DMG, 0, bsNumSaocObjects);		
}		
} else {		
for(i=dmgIdx; i<dmgIdx + bsNumSaocDmxObjects; i++) {		
idxDMG[i] = EcDataSaoc(DMG, 0, NumPremixedChannels);		
}		
}		
}		
if (bsDcuFlag == 1) && (bsDcuDynamic == 1) {		
if (bsIndependencyFlag == 1) {		
bsDcuDynamicUpdate = 1;		
} else {		
bsDcuDynamicUpdate;	**1**	**uimsbf**
}		
if (bsDcuDynamicUpdate == 1) {		
bsDcuMode;	**1**	**uimsbf**
bsDcuParam;	**4**	**uimsbf**
}		
}		
ByteAlign();		
SAOC3DExtensionFrame();		
}		

[a] numBands shall be as defined in ISO/IEC 23003-2.

Table 150 — Syntax of SAOC3DFramingInfo()

Syntax	No. of bits	Mnemonic
SAOC3DFramingInfo()		
{		
bsFramingType;	**1**	**uimsbf**
bsNumParamSets;	**3**	**uimsbf**
if (bsFramingType) {		
for (ps=0; ps<numParamSets; ps++) {		[a]
bsParamSlot[ps];	**nBitsParamSlot** [b]	**uimsbf**
}		
}		
}		

[a] numParamSets is defined by numParamSets = bsNumParamSets + 1.

[b] nBitsParamSlot is defined according to nBitsParamSlot = ceil(log2(numSlots)).

9.4.2 Definition of SAOC 3D payloads

The following tables contain definitions of used SAOC 3D bitstream syntactic elements and variables.

Table 151 — Definitions of SAOC 3D syntactic elements

SAOC 3D syntactic elements	Definition (in accordance with ISO/IEC 23003-2)	Reference
EcDataSaoc()	EcDataSaoc()	Table 22
SAOC3DExtensionConfig()	SAOCExtensionConfig()	Table 6
SAOC3DExtensionConfigData(bsSaoc3DExtType)	N/A	
SAOC3DExtensionFrame()	SAOCExtensionFrame()	Table 27
SAOC3DExtensionFrameData(bsSaoc3DExtType)	N/A	
ByteAlign()	ByteAlign()	

Table 152 — Definitions of SAOC 3D bitstream variables

SAOC 3D bitstream variables	Definition (in accordance with ISO/IEC 23003-2)
bsSamplingFrequencyIndex	bsSamplingFrequencyIndex
bsSamplingFrequency	bsSamplingFrequency
bsDcuFlag	bsDcuFlag
bsOneIOC	bsOneIOC
bsDcuMode	bsDcuMode
bsDcuMandatory	bsDcuMandatory
bsDcuDynamic	bsDcuDynamic
bsDcuDynamicUpdate	bsDcuDynamicUpdate
bsDcuParam	bsDcuParam
bsIndependencyFlag	bsIndependencyFlag
bsRelatedTo	bsRelatedTo

saocChannelLayout — Defines the input channel layout for which SAOC 3D parameters are transmitted.

NumSaocChannels — Defines the number of input channels for which SAOC 3D parameters are transmitted.

bsNumSaocObjects — Defines the number of input objects for which SAOC 3D parameters are transmitted.

bsDoubleFrameLengthFlag — Indicates whether the SAOC 3D frame length is equal to or double of the core coder output frame length. The number of time slots is given by: numSlots = ((bsDoubleFrameLengthFlag + 1) · outputFrameLength) / 64, where outputFrameLength is defined in ISO/IEC 23003-3:2012, 6.1, Table 70.

bsNumSaocDmxChannels — Defines the number of SAOC 3D downmix channels for channel based content.

bsNumSaocDmxObjects — Defines the number of SAOC 3D downmix channels for object based content according to Table 153.

Table 153 — Definition of decoding mode

bsNumSaocDmxObjects	Mode	Meaning
0	"Combined"	All input signals are combined into N_{ch} channels
1 ... 31	"Independent"	The channel-based and object-based signals are downmixed independently into N_{ch} and N_{obj} channels

bsSaocDmxMethod Defines the downmix matrix mode and number of premixing channels according to Table 154.

Table 154 — bsSaocDmxMethod

bsSaocDmxMethod	Mode	Meaning
0	"Direct"	Downmix matrix is defined directly by DMGs.
1	"Premixing"	Downmix matrix is defined as a product of the matrix obtained from the dequantized DMGs and a premixing matrix obtained from the spatial information of the input audio objects and the reference layout. The Premixing mode can be used only if the reference layout is defined in mpegh3daConfig().

NumPremixedChannels Defines the number of premixing channels (N_{premix}) for the input audio objects.

bsDecorrelationMethod Defines the decorrelation method according to Table 155.

Table 155 — bsDecorrelationMethod

bsDecorrelationMethod	Meaning
0	"Energy compensation method"
1	"Covariance adjustment method"

bsDualMode Indicates if decoding operates in different modes for a low and high band range according to Table 156.

Table 156 — bsDualMode

bsDualMode	Meaning
0	Same decoding mode for full band range (i.e., no separate high band range)
1	Different decoding modes for low and high band ranges

bsBandsLow Defines the number of parameter bands for which decoding should be processed according to prediction based scheme.

Prediction based scheme should be used for the parameter band range: 0 <= pb < bsBandsLow.

Energy based scheme should be used for the parameter band range: bsBandsLow <= pb < numBands.

bsSaocExtType Indicates type of the SAOC 3D extension data according to Table 157.

Table 157 — bsSaoc3DExtType

bsSaoc3DExtType	SAOC3DExtensionFrameData()
0 ... 7	present
8 ... 15	not present

9.5 SAOC 3D processing

9.5.1 Compressed data stream decoding and dequantization of SAOC 3D data

The process for dequantization of the DMG, OLD, IOC parameters shall be as specified in ISO/IEC 23003-2.

9.5.2 Time/frequency transforms

The hybrid filterbank as specified in ISO/IEC 23003-1 shall be applied.

9.5.3 Signals and parameters

9.5.3.1 Dimensionality of signals and parameters

The audio signals are defined for every time slot n and every hybrid subband k. The corresponding SAOC 3D parameters are defined for each parameter time slot l and processing band m. The subsequent mapping between the hybrid and parameter domains shall be as specified by ISO/IEC 23003-1:2007, Table A.31. Hence, all calculations are performed with respect to the appropriate time/band indices and the corresponding dimensionalities are implied for each introduced variable.

The data available at the SAOC 3D decoder consists of the multi-channel downmix signal **X**, the covariance matrix **E**, the rendering matrix **R** and downmix matrix **D**.

9.5.3.2 Object parameters

The covariance matrix **E** of size $N \times N$ with elements $e_{i,j}$ represents an approximation of the original signal covariance matrix $\mathbf{E} \approx \mathbf{S}\mathbf{S}^*$ and is obtained from the OLD and IOC parameters as:

$$e_{i,j} = \sqrt{OLD_i OLD_j}\, IOC_{i,j}$$

Here, the dequantized object parameters are obtained as:

$$OLD_i = \mathbf{D}_{\mathrm{OLD}}(i,l,m), \quad IOC_{i,j} = \mathbf{D}_{\mathrm{IOC}}(i,j,l,m)$$

9.5.3.3 Downmix matrix

The downmix matrix **D** applied to the input audio signals **S** determines the downmix signal as **X** = **DS**. The downmix matrix **D** of size $N_{\mathrm{dmx}} \times N$ is obtained as:

$\mathbf{D} = \mathbf{D}_{\mathrm{dmx}}\mathbf{D}_{\mathrm{premix}}$.

— "Direct mode" (bsSaocDmxMethod == 0):

The matrix $\mathbf{D}_{\mathrm{premix}}$ of size $N \times N$ is defined as:

$\mathbf{D}_{premix} = \mathbf{I}$.

— "Premixing mode" (bsSaocDmxMethod == 1):

The matrix $\mathbf{D}_{premix}$ of size $(N_{ch} + N_{premix}) \times N$ is defined as:

$$\mathbf{D}_{premix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{A} \end{bmatrix}.$$

The premixing matrix $\mathbf{A}$ of size $N_{premix} \times N_{obj}$ is received as an input to the SAOC 3D decoder from the object renderer as a function of decoded object metadata and channel configuration of the reference output format (number and geometric positions of premixed channels, "referenceLayout").

The matrix $\mathbf{D}_{dmx}$ of size $N_{dmx} \times \bar{N}$ is obtained from the DMG parameters as:

— " "Combined decoding mode" (bsNumSaocDmxObjects == 0):

$$d_{i,j} = 10^{0.05 DMG_{i,j}}$$

— "Independent decoding mode" (bsNumSaocDmxObjects >= 1):

$$d_{i,j} = \begin{cases} 10^{0.05 DMG_{i,j}} & ,0 \le i < N_{ch}^{dmx}, 0 \le j < N_{ch} \\ 10^{0.05 DMG_{i,j-N_{ch}}} & ,N_{ch}^{dmx} \le i < N_{dmx}, N_{ch} \le j < \bar{N} \\ 0 & ,\text{otherwise} \end{cases},$$

where $\bar{N} = N$ for the "direct mode" or $\bar{N} = N_{ch} + N_{premix}$ for the "premixing mode".

Here, the dequantized downmix parameters are obtained as:

$DMG_{i,j} = \mathbf{D}_{DMG}(i,j,l)$.

9.5.3.4 Rendering matrix

The rendering matrix $\mathbf{R}$ applied to the input audio signals $\mathbf{S}$ determines the target rendered output $\hat{\mathbf{Y}} = \mathbf{RS}$.

The rendering matrix $\mathbf{R}$ of size $N_{out} \times N$ is given by:

$\mathbf{R} = [\mathbf{R}_{ch} \quad \mathbf{R}_{obj}]$,

where the matrix $\mathbf{R}_{ch}$ of size $N_{out} \times N_{ch}$ is associated with input channels and matrix $\mathbf{R}_{obj}$ of size $N_{out} \times N_{obj}$ is associated with input objects.

The rendering matrix $\mathbf{R}_{ch}$ is received as an input to the SAOC 3D decoder from the format converter as a function of the: channel configuration of the channels for which SAOC 3D parameters are transmitted (number and geometric positions of SAOC 3D input channels, saocChannelLayout) and the reproduction layout which is received as input to the SAOC 3D decoder:

$$\mathbf{R}_{ch} = \mathbf{M}_{DMX},$$

where matrix $\mathbf{M}_{DMX}$ is defined in subclause 10.2.2. The downmix matrix $\mathbf{M}_{DMX}$ is different to the one which is used for regular audio channels, but is just computed in the same way by the format converter

The rendering matrix $\mathbf{R}_{obj}$ is received as an input to the SAOC 3D decoder from the object renderer as a function of decoded object metadata and the reproduction layout which is received as input to the SAOC 3D decoder:

$$\mathbf{R}_{obj} = \mathbf{G},$$

where matrix $\mathbf{G}$ is defined in subclause 8.4.4.3 for the specified timestamps. The rendering matrix $\mathbf{R}_{obj}$ is determined for each parameter time-slot l by linear interpolation of the gain factors $\mathbf{G}$, between the timestamps preceding and following the parameter time-slot l.

9.5.3.5 Target output covariance matrix

The covariance matrix $\mathbf{C}$ of size $N_{out} \times N_{out}$ representing an approximation of the target output signal covariance $\mathbf{C} \approx \mathbf{YY}^*$ is obtained as:

$$\mathbf{C} = \mathbf{RER}^*$$

9.5.4 SAOC 3D decoding

9.5.4.1 Overview

The method for obtaining an output signal using SAOC 3D parameters and rendering information is described in this subclause. The basic structure of the SAOC 3D decoder, consisting of the parameter processor and the downmix processor, is depicted in Figure 48.

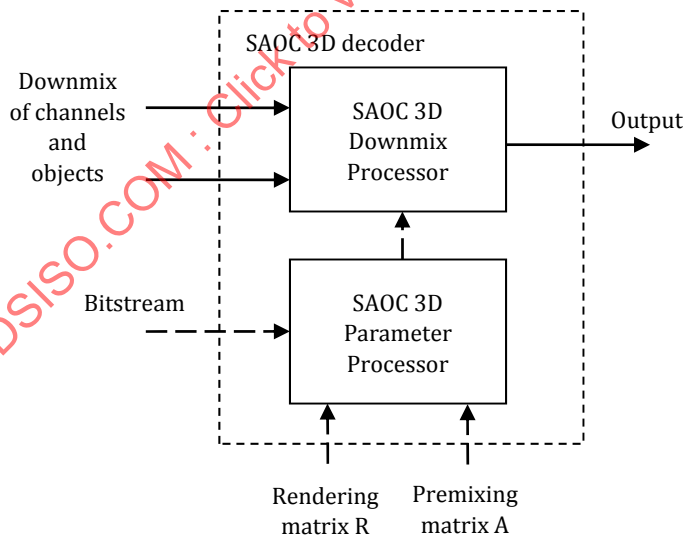


Figure 48 — Structure of the SAOC 3D decoder

9.5.4.2 SAOC 3D downmix processor

9.5.4.2.1 General

The detailed structure of the SAOC 3D downmix processor is depicted in Figure 49.

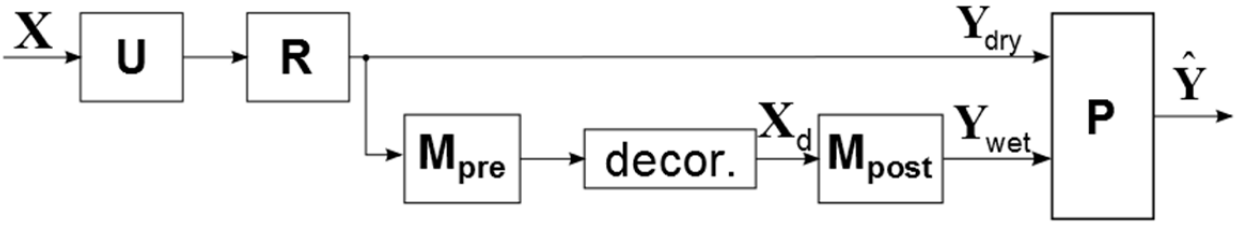


Figure 49 — Structure of the downmix processor

The output signal $\hat{\mathbf{Y}}$ is computed from the signals $\mathbf{Y}_{dry}$ and $\mathbf{Y}_{wet}$ as:

$$\hat{\mathbf{Y}} = \mathbf{P}\begin{bmatrix}\mathbf{Y}_{dry}\\ \mathbf{Y}_{wet}\end{bmatrix}$$

The signals $\mathbf{Y}_{dry}$ and $\mathbf{Y}_{wet}$ are calculated from the downmix signal $\mathbf{X}$ and the decorrelated signal $\mathbf{X}_d$ as:

$$\mathbf{Y}_{dry} = \mathbf{RUX}$$

$$\mathbf{Y}_{wet} = \mathbf{M}_{post}\mathbf{X}_d$$

The matrix product $(\mathbf{RU})^{l,k}$ and the matrix $(\mathbf{P})^{l,k}$, computed for every parameter time-slot l, are interpolated over all time-slots n in accordance with ISO/IEC 23003-1:2007, 6.5.2.1, considering:

$$\mathbf{W}_1^{l,k} = (\mathbf{RU})^{l,k}, \text{ for computing } (\mathbf{RU})^{n,k} = \mathbf{M}_1^{n,k}$$

and

$$\mathbf{W}_1^{l,k} = \mathbf{P}^{l,k}, \text{ for computing } (\mathbf{P})^{n,k} = \mathbf{M}_1^{n,k}$$

where $\mathbf{W}_1^{l,k}$ and $\mathbf{M}_1^{n,k}$ shall be variables as specified in ISO/IEC 23003-1.

9.5.4.2.2 Parametric unmixing matrix

The parametric unmixing matrix **U** is obtained according to the "decoding mode" as:

— "Combined decoding mode" (bsNumSaocDmxObjects == 0):

$$\mathbf{U} = \mathbf{ED}^*\mathbf{J}.$$

The matrix $\mathbf{J} \approx \mathbf{\Delta}^{-1}$ of size $N_{dmx} \times N_{dmx}$ for $\mathbf{\Delta} = \mathbf{DED}^*$ is derived according to subclause 9.5.4.2.5.

— "Independent decoding mode" (bsNumSaocDmxObjects >= 1):

$$\mathbf{U} = \begin{bmatrix}\mathbf{U}_{ch} & 0\\ 0 & \mathbf{U}_{obj}\end{bmatrix}, \ \mathbf{U}_{ch} = \mathbf{E}_{ch}\mathbf{D}_{ch}^*\mathbf{J}_{ch}, \ \mathbf{U}_{obj} = \mathbf{E}_{obj}\mathbf{D}_{obj}^*\mathbf{J}_{obj}.$$

The channel based covariance matrix $\mathbf{E}_{ch}$ of size $N_{ch} \times N_{ch}$ and the object based covariance matrix $\mathbf{E}_{obj}$ of size $N_{obj} \times N_{obj}$ are obtained from the covariance matrix **E** by selecting only the corresponding main diagonal blocks:

$$\mathbf{E} = \begin{bmatrix} \mathbf{E}_{\text{ch}} & \mathbf{E}_{\text{ch,obj}} \\ \mathbf{E}_{\text{obj,ch}} & \mathbf{E}_{\text{obj}} \end{bmatrix}.$$

The object-channel cross-covariance matrix $\mathbf{E}_{\text{ch,obj}} = \mathbf{E}^{*}_{\text{obj,ch}}$ is not used in the computations.

The channel based downmix matrix $\mathbf{D}_{\text{ch}}$ of size $N_{\text{ch}}^{\text{dmx}} \times N_{\text{ch}}$ and the object based downmix matrix $\mathbf{D}_{\text{obj}}$ of size $N_{\text{obj}}^{\text{dmx}} \times N_{\text{obj}}$ are obtained from the downmix matrix $\mathbf{D}$ by selecting only the corresponding main diagonal blocks:

$$\mathbf{D} = \begin{bmatrix} \mathbf{D}_{\text{ch}} & 0 \\ 0 & \mathbf{D}_{\text{obj}} \end{bmatrix}.$$

The matrices $\mathbf{J}_{\text{ch}} \approx \mathbf{\Delta}^{-1}$ of size $N_{\text{ch}}^{\text{dmx}} \times N_{\text{ch}}^{\text{dmx}}$ for $\mathbf{\Delta} = \mathbf{D}_{\text{ch}}\mathbf{E}_{\text{ch}}\mathbf{D}^{*}_{\text{ch}}$ and $\mathbf{J}_{\text{obj}} \approx \mathbf{\Delta}^{-1}$ of size $N_{\text{obj}}^{\text{dmx}} \times N_{\text{obj}}^{\text{dmx}}$ for $\mathbf{\Delta} = \mathbf{D}_{\text{obj}}\mathbf{E}_{\text{obj}}\mathbf{D}^{*}_{\text{obj}}$ are derived according to subclause 9.5.4.2.5.

9.5.4.2.3 Decorrelation

The pre-processing matrix $\mathbf{M}_{\text{pre}}$ is defined for different output configurations in Annex B.

The post-processing matrix $\mathbf{M}_{\text{post}}$ is obtained as:

$$\mathbf{M}_{\text{post}} = \mathbf{M}^{*}_{\text{pre}}\mathbf{J}_{\text{pre}}.$$

The matrix $\mathbf{J}_{\text{pre}} \approx \left(\mathbf{M}_{\text{pre}}\mathbf{M}^{*}_{\text{pre}}\right)^{-1}$ is derived according to subclause 9.5.4.2.5.

The decorrelated signals $\mathbf{X}_{\text{d}}$ shall be created from the decorrelator as specified in ISO/IEC 23003-1:2007, 6.6.2, with bsDecorrConfig == 0 and a decorrelator index *X* according to Tables B.1 and B.2. Hence, the function $decorrFunc(\cdot)$ denotes the decorrelation process:

$$\mathbf{X}_{\text{d}} = decorrFunc\left(\mathbf{M}_{\text{pre}}\mathbf{Y}_{\text{dry}}\right).$$

9.5.4.2.4 Mixing matrix P

The following covariance matrices notation is introduced:

— The matrix $\mathbf{E}_{\mathbf{Y}}^{\text{dry}}$ representing the covariance of the parametrically estimated signal $\mathbf{E}_{\mathbf{Y}}^{\text{dry}} \approx \mathbf{Y}_{\text{dry}}\mathbf{Y}^{*}_{\text{dry}}$ and defined as:

$$\mathbf{E}_{\mathbf{Y}}^{\text{dry}} = \mathbf{RU}\left(\mathbf{DED}^{*}\right)\mathbf{U}^{*}\mathbf{R}^{*}$$

— The matrix $\mathbf{E}_{\mathbf{Y}}^{\text{wet}}$ representing the covariance matrix of the decorrelated signal $\mathbf{E}_{\mathbf{Y}}^{\text{wet}} \approx \mathbf{Y}_{\text{wet}}\mathbf{Y}^{*}_{\text{wet}}$ and defined as:

$$\mathbf{E}_{\mathbf{Y}}^{\text{wet}} = \mathbf{M}_{\text{post}}\left[matdiag(\mathbf{M}_{\text{pre}}\mathbf{E}_{\mathbf{Y}}^{\text{dry}}\mathbf{M}^{*}_{\text{pre}})\right]\mathbf{M}^{*}_{\text{post}}$$

— The matrix $\mathbf{\Delta}_E$ describing difference between the target output covariance and the covariance of the parametrically estimated signals and computed as:

$$\mathbf{\Delta}_E = \mathbf{C} - \mathbf{E}_{\mathbf{Y}}^{\text{dry}}$$

The mixing matrix $\mathbf{P}$ of size $N_{\text{out}} \times 2N_{\text{out}}$ is given by:

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_{\text{dry}} & \mathbf{A}_{\text{wet}}\mathbf{P}_{\text{wet}} \end{bmatrix}$$

The limitation matrix $\mathbf{A}_{\text{wet}}$ of size $N_{\text{out}} \times N_{\text{out}}$ is given by:

$$\mathbf{A}_{\text{wet}} = \text{matdiag}\left(\min\left(1, \sqrt{\max\left(0, \lambda_{Dec} \frac{\mathbf{E}_{\mathbf{Y}}^{\text{dry}}(i,i)}{\max\left(\varepsilon, \hat{\mathbf{E}}_{\mathbf{Y}}^{\text{wet}}(i,i)\right)} \right)} \right) \right)$$

where $\lambda_{Dec} = 4$ is a constant used to limit the amount of decorrelated component added to the output signals. The matrix $\hat{\mathbf{E}}_{\mathbf{Y}}^{\text{wet}}$ representing the estimated covariance matrix of the decorrelated signals after the mixing matrix $\mathbf{P}_{\text{wet}}$ has been applied, and defined as:

$$\hat{\mathbf{E}}_{\mathbf{Y}}^{\text{wet}} = \mathbf{P}_{\text{wet}}\mathbf{E}_{\mathbf{Y}}^{\text{wet}}\mathbf{P}_{\text{wet}}^{*}$$

The mixing matrices $\mathbf{P}_{\text{dry}}$, $\mathbf{P}_{\text{wet}}$ of size $N_{\text{out}} \times N_{\text{out}}$ are obtained according to the "decorrelation method" as:

— "Energy compensation method" (bsDecorrelationMethod == 0):

$\mathbf{P}_{\text{dry}} = \mathbf{I}$

$$\mathbf{P}_{\text{wet}}(i,j) = \begin{cases} \sqrt{\max\left(0, \dfrac{\mathbf{C}(i,i) - \mathbf{E}_{\mathbf{Y}}^{\text{dry}}(i,i)}{\max\left(\varepsilon, \mathbf{E}_{\mathbf{Y}}^{\text{wet}}(i,i)\right)} \right)} & i = j, \\ 0 & i \neq j. \end{cases}$$

— "Covariance adjustment method" (bsDecorrelationMethod == 1):

$\mathbf{P}_{\text{dry}} = \mathbf{I}$

$$\mathbf{P}_{\text{wet}} = \left(\mathbf{V}_1\sqrt{\mathbf{Q}_1}\mathbf{V}_1^{*}\right)\left(\mathbf{V}_2\sqrt{\mathbf{Q}_2^{inv}}\mathbf{V}_2^{*}\right)$$

The matrices $\mathbf{V}_1$ and $\mathbf{Q}_1$ are determined as the singular value decomposition of the matrix $\mathbf{\Delta}_E$ as:

$$\mathbf{\Delta}_E = \mathbf{V}_1\mathbf{Q}_1\mathbf{V}_1^{*}$$

The matrices $\mathbf{V}_2$ and $\mathbf{Q}_2$ are determined as the singular value decomposition of the matrix $\mathbf{E}_{\mathbf{Y}}^{\text{wet}}$ as:

$$\mathbf{E}_{\mathbf{Y}}^{\text{wet}} = \mathbf{V}_2\mathbf{Q}_2\mathbf{V}_2^{*}$$

9.5.4.2.5 Regularized inverse operation

The regularized inverse matrix **J** approximating $\mathbf{J} \approx \mathbf{\Delta}^{-1}$ is calculated as:

$$\mathbf{J} = \mathbf{V}\mathbf{\Lambda}^{inv}\mathbf{V}^{*}.$$

The matrices **V** and $\mathbf{\Lambda}$ are determined as the singular value decomposition of the matrix Δ as:

$$\mathbf{V}\mathbf{\Lambda}\mathbf{V}^{*} = \mathbf{\Delta}.$$

The regularized inverse $\mathbf{\Lambda}^{inv}$ of the diagonal singular value matrix $\mathbf{\Lambda}$ is computed according to subclause 9.5.4.2.6.

In the case the matrix Δ is used in the calculation of the parametric un-mixing matrix **U**, the operations described are applied for all sub-matrices $\mathbf{\Delta}_q$. A sub-matrix $\mathbf{\Delta}_q$ of size $N_{\mathrm{g}}^{q} \times N_{\mathrm{g}}^{q}$, with elements $\mathbf{\Delta}_q\left(idx_1, idx_2\right)$, is obtained by selecting the elements $\mathbf{\Delta}\left(ch_1, ch_2\right)$ corresponding to the downmix channels ch_1 and ch_2 assigned to the group $\mathbf{g}_q$ (i.e., $\mathbf{g}_q(idx_1) = ch_1$ and $\mathbf{g}_q(idx_2) = ch_2$).

The group $\mathbf{g}_q$ of size $1 \times N_{\mathrm{g}}^{q}$ is defined by the smallest set of downmix channels with the following properties.

k) The input signals contained in the downmix channels of group $\mathbf{g}_q$ are not contained in any other downmix channel. An input signal is not contained in a downmix channel if the corresponding downmix gain is given by the smallest allowed value of the quantization index see idxDMG, as defined in ISO/IEC 23003-2.

l) All input signals *i* contained in the downmix channels of group $\mathbf{g}_q$ are not related to any input signal *j* contained in any downmix channel of any other group (i.e., **bsRelatedTo**[i][j] == 0).

The results of the independent regularized inversion operations $\mathbf{J}_q \approx \mathbf{\Delta}_q^{-1}$ are combined for obtaining the matrix **J** as:

$$\mathbf{J}\left(ch_1, ch_2\right) = \begin{cases} \mathbf{J}_q\left(idx_1, idx_2\right) & \text{, if } \mathbf{g}_q\left(idx_1\right) = ch_1 \text{ and } \mathbf{g}_q\left(idx_2\right) = ch_2, \\ 0 & \text{, otherwise.} \end{cases}$$

9.5.4.2.6 Regularization of singular values

The regularized inverse operation $(\cdot)^{inv}$ used for the diagonal singular value matrix $\mathbf{\Lambda}$ is determined as:

$$\mathbf{\Lambda}^{inv} = \lambda_{i,j}^{-1} = \begin{cases} \dfrac{1}{\lambda_{i,i}} & \text{, if } i = j \text{ and } abs\left(\lambda_{i,i}\right) \geq T_{reg}^{\Lambda}, \\ 0 & \text{, otherwise.} \end{cases}$$

The relative regularization scalar T_{reg}^{Λ} is determined using absolute threshold T_{reg} and maximal value of $\mathbf{\Lambda}$ as follows:

$$T_{reg}^{\Lambda} = \max_i \left(\text{abs}\left(\lambda_{i,i} \right) \right) T_{reg}, \quad \text{with } T_{reg} = 10^{-2}$$

9.5.5 Dual mode

The SAOC 3D decoder can use an alternative scheme for calculation of the parameters **U** and **P** for the upper frequency range, defined by parameter bands $\textbf{bsBandsLow} \le pb < numBands$. This scheme is particularly useful for downmix signals where the upper frequency range is coded by a non-waveform preserving coding algorithm e.g. SBR in high efficiency AAC. The matrices **U**, $\mathbf{P}_{\text{dry}}$ and $\mathbf{P}_{\text{wet}}$ are determined as:

$\mathbf{U} = \mathbf{GT}$,

$\mathbf{P}_{\text{dry}} = \mathbf{I}$,

$\mathbf{P}_{\text{wet}} = \mathbf{0}$.

The matrix **T** is defined as:

$$\mathbf{T} = \mathbf{RD}^{*}\mathbf{J}$$

The matrix $\mathbf{J} \approx \mathbf{\Delta}^{-1}$ of size $N_{dmx} \times N_{dmx}$ for $\mathbf{\Delta} = \mathbf{DD}^{*}$ is derived according to subclause 9.5.4.2.5.

The gain matrix **G** of size $N_{\text{out}} \times N_{\text{out}}$ is given by:

$$\mathbf{G} = g_{i,j} = \begin{cases} \sqrt{\dfrac{c_{i,i}}{e_i^{upmix}}} & \text{, if } i = j, \\ 0 & \text{, otherwise.} \end{cases}$$

The energy upmix vector e_i^{upmix} of size $1 \times N_{out}$ is given by:

$$e_i^{upmix} = diag\left(\mathbf{T} \left[matdiag\left(\mathbf{DED}^{*} \right) \right] \mathbf{T}^{*} + \varepsilon \mathbf{I} \right)$$

The unmixing matrix **U** contains the rendering matrix **R** thus the rendering block in Figure 49 is omitted for the upper frequency range.

The matrix $(\mathbf{U})^{l,k}$, computed for every parameter time-slot *l*, shall be interpolated over all time-slots *n* as defined in ISO/IEC 23003-1:2007, 6.5.2.1, considering:

$$\mathbf{W}_1^{l,k} = \mathbf{U}^{l,k} \quad \text{for computing } \mathbf{U}^{n,k} = \mathbf{M}_1^{n,k}$$

where $\mathbf{W}_1^{l,k}$ and $\mathbf{M}_1^{n,k}$ shall be variables as specified in ISO/IEC 23003-1.

10 Generic loudspeaker rendering/format conversion

10.1 Description

The loudspeaker renderer converts multichannel signals from the transmitted channel configuration to the desired reproduction format. It is therefore also referred to as a "format converter". If the channel configuration of the channels routed to the format converter exactly matches the signalled reproduction layout (i.e. the target channel configuration), the format converter shall be bypassed. The format converter consists of two major building blocks:

— an initialization algorithm that takes into account static parameters like the input and output format;

— a signal adaptive downmixing process that operates in a subband domain.

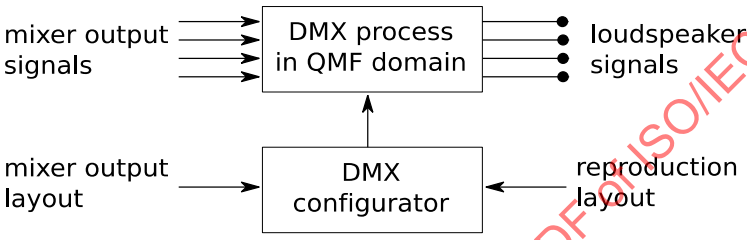


Figure 50 — Main building blocks of the generic format converter

In the initialization phase the format converter automatically generates optimized downmixing parameters (like the downmixing matrix) for the given combination of input and output formats. It applies an algorithm that selects for each input loudspeaker the most appropriate mapping rule from a list of rules that has been designed to incorporate psychoacoustic considerations. Each rule describes the mapping from one input channel to one or several output loudspeaker channels.

Input channels are

— either mapped to a single output channel;

— or panned to two output channels;

— or (in case of the 'Voice of God' channel) distributed over a larger number of output channels.

The optimal mapping for each input channel is selected depending on the list of output loudspeakers that are available in the desired output format. Each mapping defines downmix gains for the input channel under consideration as well as potentially also an equalizer that is applied to the input channel under consideration.

Output setups with non-standard loudspeaker positions can be signalled to the system by providing the azimuth and elevation deviations from a regular loudspeaker setup.

The actual downmixing of the audio signals is performed on a hybrid QMF subband representation of the signals. The algorithm makes use of two mechanisms to avoid signal deteriorations like comb-filtering, colouration, or modulation artifacts.

— Phase-alignment of the multichannel input signals: Correlated input signals that differ in phase are aligned prior to downmixing them. The alignment process makes use of an attraction measure to only align the relevant channels for the relevant time-frequency tiles and to avoid

modifications to other parts of the input signal. The alignment is further regularized to avoid artifacts due to rapid changes to the phase alignment modification terms. The phase-alignment improves the output signal quality by avoiding narrow spectral notches due to out-of-phase signal cancellations that cannot be compensated for by energy normalization because of a limited frequency resolution. It further reduces the need to boost signals in the energy preserving normalization, thus minimizes modulation artifacts.

— Normalization of the downmix process to preserve the input energies (except for the desired energy scaling that may be inherent in the downmix matrix).

10.2 Definitions

10.2.1 General remarks

Audio signals that are fed into the format converter are referred to as *input signals* in the following. Audio signals that are the result of the format conversion process are referred to as *output signals*. Note that the audio input signals of the format converter are audio output signals from the core decoder.

Vectors and matrices are denoted by bold-faced symbols. Vector elements or matrix elements are denoted as italic variables supplemented by indices indicating the row/column of the vector/matrix element in the vector/matrix, e.g. $[y_1 \cdots y_a \cdots y_N] = \mathbf{y}$ denotes a vector and its elements. Similarly, $M_{a,b}$ denotes the element in the ath row and bth column of a matrix **M**.

10.2.2 Variable definitions

N_{in}	Number of channels in the input channel configuration.
N_{out}	Number of channels in the output channel configuration.
$\mathbf{M}_{\text{DMX}}$	Downmix matrix containing real-valued non-negative downmix coefficients (downmix gains). $\mathbf{M}_{\text{DMX}}$ is of dimension ($N_{\text{out}} \times N_{\text{in}}$).
$\mathbf{G}_{\text{EQ}}$	Matrix consisting of gain values per processing band determining frequency responses of equalizing filters.
$\mathbf{I}_{\text{EQ}}$	Vector signalling which equalizer filters to apply to the input channels (if any).
L	Frame length measured in the time domain audio samples.
ν	Time domain sample index.
n	QMF time slot index (= subband sample index).
L_n	Frame length measured in QMF slots.
F	Frame index (frame number).
K	Number of hybrid QMF frequency bands, K= 71.
k	QMF band index (1..64) or hybrid QMF band index (1.. K).
A, B	Channel indices.
eps	Numerical constant, $\text{eps} = 10^{-35}$.

10.3 Processing

10.3.1 Application of transmitted downmix matrices

10.3.1.1 General

MPEG-H 3D audio allows transmission of downmix definitions for specific target channel configurations. downmixIds are assigned to the transmitted downmix specifications, allowing DRC to adapt to the downmix specification applied in the MPEG-H decoder (e.g. to select an appropriate DRC gain sequence). Further, loudness metadata values may be coupled with downmixIds. downmixIds are transmitted in the bitstream together with the downmixType as well as the nominal loudspeaker layouts for which the embedded downmix matrices (and/or DRC and loudness data) have been designed.

In the MPEG-H 3D audio decoder the selection of a downmixIds thus:

— determines whether transmitted downmix coefficients (downmixType=1) or decoder side generated downmix coefficients (downmixType=0) are applied in the downmix process;

— influences DRC/loudness processing.

Even moderately displaced reproduction layouts benefit from the transmission and application of transmitted downmix coefficients as follows.

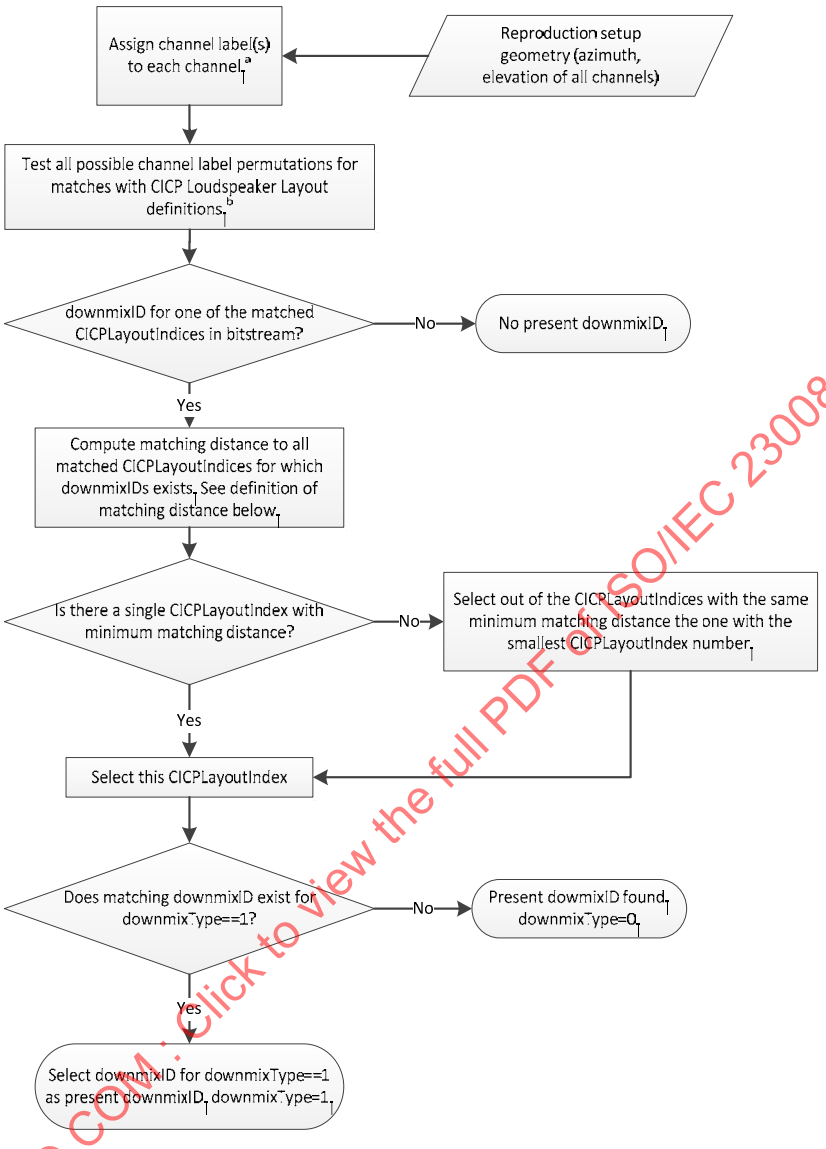
— The selection of a significantly different downmix matrix (transmitted vs. decoder generated) results in large perceptual changes of the downmix result.

— The artistic intent of transmitted downmix coefficients would be lost if a transmitted downmix matrix is not applied.

Of course, in the situation that the loudspeaker displacements of the reproduction setup are too large, the application of the transmitted downmix coefficients may result in a larger perceptual distance from the intended reproduction than the application of decoder generated downmix coefficients. As a consequence, the allowed loudspeaker displacement values are restricted in the following matching scheme.

10.3.1.2 Loudspeaker layout matching scheme

The downmixId matching algorithm is specified by the flowchart of Figure 51. It takes as input the geometry of the actual reproduction setup and outputs the present downmixId (if applicable). The present downmixId determines the downmix processing in the decoder as shown in subclause 10.3.1.4. Further, the present downmixId may affect DRC and loudness processing.



a Channel labels shall be assigned according to Table 158.

b CICP Loudspeaker Layout definitions according to Table 163.

Figure 51 — Flowchart for loudspeaker layout matching and to determine downmixId

The *matching distance* is defined as the sum of all absolute azimuth and elevation angle differences between the channel positions of the reproduction layout and the tested CICP loudspeaker layout, summed over all channels of the reproduction setup, excluding the LFE channels:

$$d_{\text{matching}} = \sum_{ch \in \left\{ \begin{matrix} \text{all channels} \\ \text{except LFEs} \end{matrix} \right\}} \left| \varphi_{\text{CICPLayout},ch} - \varphi_{\text{reprodcutionLayout},ch} \right| + \left| \vartheta_{\text{CICPLayout},ch} - \vartheta_{\text{reprodcutionLayout},ch} \right|$$

where φ denotes azimuth angles and ϑ denotes elevation angles.

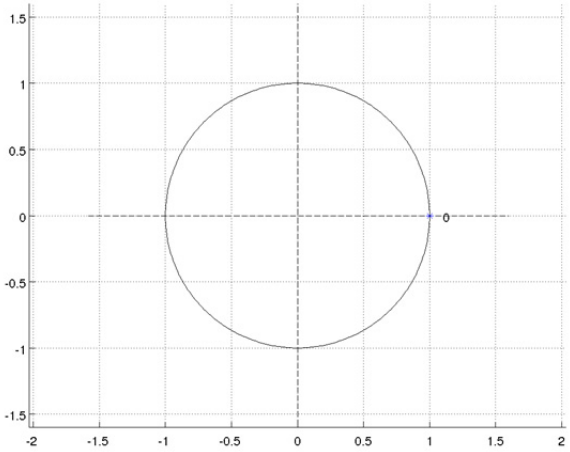
Table 158 — Channel matching tolerances for matching downmixIds to reproduction layouts

Loudspeaker Geometry (as defined in ISO/IEC 23001-8)	Channel	Azimuth [deg]	Elevation [deg]	Azimuth start angle of sector [deg]	Azimuth end angle of sector [deg]	Elevation start angle of sector [deg]	Elevation end angle of sector [deg]	Ch. is LFE	Position is relative
	CH_EMPTY	n/a	n/a	n/a	n/a	n/a	n/a	0	0
0	CH_M_L030	30	0	15	45	-15	15	0	0
1	CH_M_R030	-30	0	-45	-15	-15	15	0	0
2	CH_M_000	0	0	-10	10	-15	15	0	0
3	CH_LFE1	0	n/a	n/a	n/a	n/a	n/a	1	0
4	CH_M_L110	110	0	90	130	-15	15	0	0
5	CH_M_R110	-110	0	-130	-90	-15	15	0	0
6	CH_M_L022	22	0	7	37	-15	15	0	0
7	CH_M_R022	-22	0	-37	-7	-15	15	0	0
8	CH_M_L135	135	0	120	150	-15	15	0	0
9	CH_M_R135	-135	0	-150	-120	-15	15	0	0
10	CH_M_180	180	0	170	190	-15	15	0	0
13	CH_M_L090	90	0	70	110	-15	15	0	0
14	CH_M_R090	-90	0	-110	-70	-15	15	0	0
15	CH_M_L060	60	0	40	80	-15	15	0	0
16	CH_M_R060	-60	0	-80	-40	-15	15	0	0
17	CH_U_L030	30	35	15	45	15	55	0	0
18	CH_U_R030	-30	35	-45	-15	15	55	0	0
19	CH_U_000	0	35	-15	15	15	55	0	0
20	CH_U_L135	135	35	115	155	15	55	0	0
21	CH_U_R135	-135	35	-155	-115	15	55	0	0
22	CH_U_180	180	35	165	195	15	55	0	0
23	CH_U_L090	90	35	70	110	15	55	0	0
24	CH_U_R090	-90	35	-110	-70	15	55	0	0
25	CH_T_000	0	90	-180	180	60	90	0	0
26	CH_LFE2	45	n/a	n/a	n/a	n/a	n/a	1	0
27	CH_L_L045	45	-15	25	65	-40	0	0	0
28	CH_L_R045	-45	-15	-65	-25	-40	0	0	0
29	CH_L_000	0	-15	-15	15	-40	0	0	0
30	CH_U_L110	110	35	90	130	15	55	0	0
31	CH_U_R110	-110	35	-130	-90	15	55	0	0
32	CH_U_L045	45	35	30	60	15	55	0	0
33	CH_U_R045	-45	35	-60	-30	15	55	0	0
34	CH_M_L045	45	0	30	60	-15	15	0	0
35	CH_M_R045	-45	0	-60	-30	-15	15	0	0
36	CH_LFE3	-45	n/a	n/a	n/a	n/a	n/a	1	0
37	CH_M_LSCR	60	0	15	80	-15	15	0	1
38	CH_M_RSCR	-60	0	-80	-15	-15	15	0	1
39	CH_M_LSCH	30	0	7	40	-15	15	0	1
40	CH_M_RSCH	-30	0	-40	-7	-15	15	0	1
41	CH_M_L150	150	0	135	165	-15	15	0	0
42	CH_M_R150	-150	0	-165	-135	-15	15	0	0

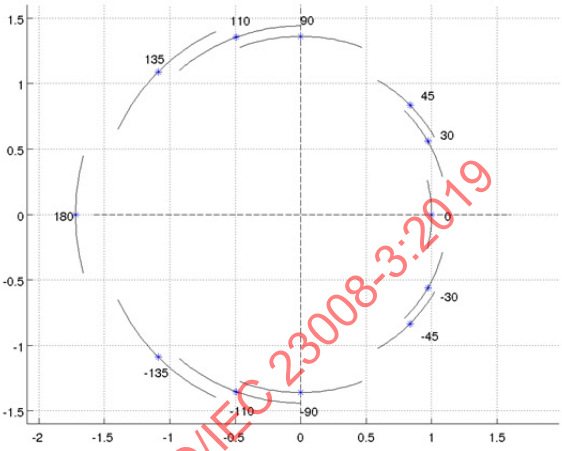
NOTE Azimuth and elevation tolerance intervals are defined as sectors, where azimuth start and end values are connected in counterclockwise direction and elevation start and end values are connected in ascending elevations. Start and end values of the sectors are considered part of the sectors.

10.3.1.3 Visualization of azimuth tolerances

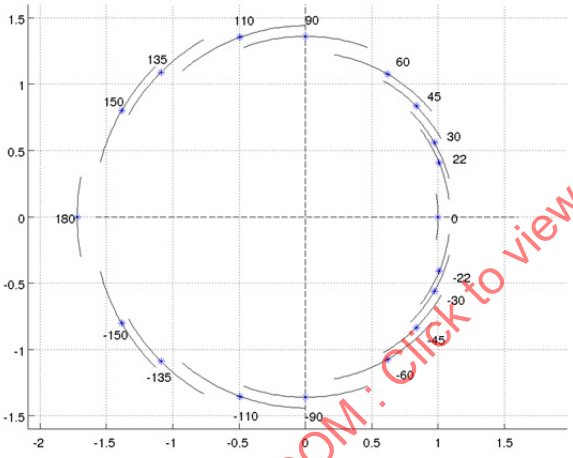
The following figures depict the azimuth matching sectors for each loudspeaker. The tolerances reflect the non-isotropic sound localization performance of the human auditory system.



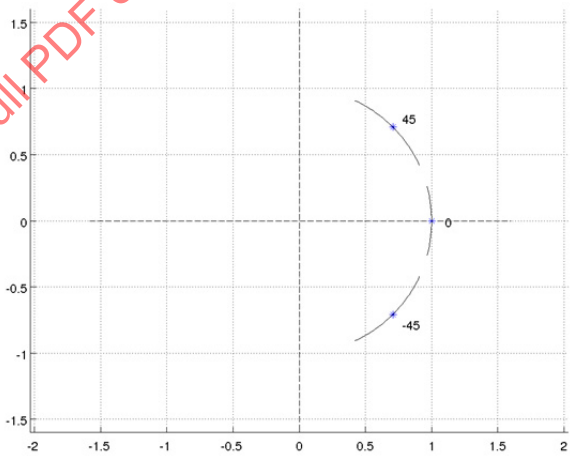
a) Top layer



b) Upper layer



c) Middle layer



d) Lower/bottom layer

NOTE Arcs have been plotted on different radii just for clarity of presentation. Nominal positions of the channels have been marked with asterisks and labels indicating the nominal azimuth angle in degrees.

Figure 52 — Visualization of azimuth tolerances

10.3.1.4 Determination of downmix processing depending on present downmixId

The downmix processing in the MPEG-H decoder is determined by the present downmixId as follows.

— If there is no present downmixId for the current reproduction setup, the downmix coefficients shall be derived as specified in the format converter initialization.

— If downmixType==0 for the present downmixId, the downmix coefficients shall be derived as specified in the format converter initialization.

— If downmixType==1 for the present downmixId, the downmix coefficients transmitted with the downmixId shall be applied in the downmix.

10.3.2 Application of transmitted equalizer settings

Equalizer settings may be transmitted together with downmix matrices, as indicated by the **EqualizerPresent** bitstream element. In case equalizer settings have been transmitted together with a downmix matrix that is applied in the format converter, the equalizers shall be applied to this downmix matrix as follows.

The transmitted equalizer parameters shall be decoded into frequency dependent gains (i.e. into equalizer frequency responses) according to subclause 10.3.4.6.4. Next, the transmitted downmix matrix gains shall be multiplied by the frequency dependent equalizer gains to arrive at the final frequency dependent downmix matrix that shall be applied in the downmix. Note that the assignment of equalizer gains to downmix matrix elements is given by the vector **equalizerIndex** that is derived according to Table 31: **equalizerIndex** tells for each input channel whether an equalizer (and if any, which) shall be applied to an input channel by applying the corresponding equalizer gains to all downmix matrix coefficients associated with the respective input channel.

10.3.3 Downmix processing involving multiple channel groups

10.3.3.1 General

In case multiple channel groups are transmitted in the MPEG-H 3D audio bitstream and routed to the format converter, one instance of the format converter shall perform a downmix of all input channels routed to the format converter to the desired target channel configuration. Therefore, all channels routed to the format converter are compiled in a group of channels that constitutes the input to the format conversion process.

If downmix matrices to one or more target channel configurations are transmitted in the bitstream, downmix matrices for those target channel configurations shall be transmitted for all channel groups/channel elements present in the bitstream.

10.3.3.2 Downmix processing with decoder generated downmix gains

In case no appropriate downmix matrices have been transmitted for the signalled target loudspeaker configuration, the downmix gains are generated during the initialization of the format converter according to subclause 10.3.4. The channels are fed to the format converter as a group of all input channels routed to the format converter. The input channel configuration signalled to the format converter shall reflect the channel geometry of this group of channels.

10.3.3.3 Downmix processing with transmitted downmix gains

In case downmix matrices have been transmitted that are applicable to the desired target channel configuration, those downmix matrices shall be applied in the format converter instead of generating downmix gains in the format converter initialization process. Whether a downmix matrix is applicable for a desired target setup, or not, is determined according to subclause 10.3.1.

All channels (and/or groups of channels) shall be compiled in one group of channels, where this group of channels consists of blocks of channels that are the channels (and/or groups of channels) routed to the format converter.

The transmitted downmix matrices assigned to the channels routed to the format converter shall be concatenated according to the order of the blocks of channels in the group of channels that forms the

input to the format converter. The concatenated downmix matrix shall then be applied in the format converter to derive the downmixed signal in the desired target channel configuration.

10.3.4 Initialization of the format converter

10.3.4.1 General description of the initialization

The initialization of the format converter is carried out before processing of the audio samples delivered by the core decoder takes place.

The initialization takes into account as input parameters.

— The sampling rate of the audio data to process.

— The channel configuration of the audio data to process with the format converter (number and geometric positions of input channels).

— The channel configuration of the desired output format (number and geometric positions of output channels).

— Optional: Parameters signaling the deviation of the output loudspeaker positions from a standard loudspeaker setup (random setup functionality).

It returns

— a frequency dependent downmix matrix $\mathbf{M}_{DMX}$ that is applied in the audio signal processing of the format converter. $\mathbf{M}_{DMX}$ is also taken into account in the core decoding process, see subclause 5.5.3.1.2.

The input parameters to the initialization algorithm are listed in Table 159.

Table 159 — Format converter initialization input parameters

	Input format: number of channels and nominal channel setup geometry.
	Output format: number of channels and nominal channel setup geometry.
f_s	Sampling frequency in Hertz.
$r_{azi,A}$	For each output channel A, an azimuth angle is specified, determining the deviation from the standard format loudspeaker azimuth.
$R_{ele,A}$	For each output channel A, an elevation angle is specified, determining the deviation from the standard format loudspeaker elevation.

Table 160 lists the output parameters that are derived during the initialization of the format converter.

Table 160 — Format converter initialization output parameters

$\mathbf{M}_{DMX}$	Downmix matrix [linear gains]

10.3.4.2 Assignment of format converter channel labels to input/output format channels

The format converter initialization is based on a system of rules that are defined in terms of *format converter channel labels*, see Table 162. To allow the application of the initialization rules, the channel labels have to be assigned to the channels of the input and output formats. Each format converter channel label is associated with a segment of the surface of the unit sphere, as defined in Table 162. The segments are designed to be non-overlapping.

The assignment of channel labels to channels is achieved by geometrically matching the segments to the position data associated with the channels of the input and output formats. The azimuth and elevation

angles in degrees of the position data associated with the channels shall be rounded towards the nearest integer number before performing the channel label assignment. Note that the *nominal* channel positions shall be applied in the following manner for matching to channel label sectors, i.e. the azimuth and elevation angles *without* taking into account potential angle deviations signalled in $r_{azi,A}$ and/or $r_{ele,A}$.

For each channel that is not an LFE (low frequency enhancement) channel.

If the nominal position of the current channel, defined by its azimuth angle and elevation angle, is within or on the border of one of the segments defined in Table 162 then:

- Assign the corresponding channel label (e.g. CH_M_L030) associated with the matching segment.
- Add the angle differences between the nominal position of the current channel and the nominal position associated with the matching segment (i.e. the angles in the second and third column of Table 162) to the angle deviations stored in $r_{azi,A}$ and $r_{ele,A}$.

Else (i.e. no matching sector found), then:

- Assign the CH_EMPTY label.

If an input or output format contains exactly one LFE channel, then the label CH_LFE2 shall be assigned to this channel.

If an input or output format contains exactly two LFE channels, then the labels CH_LFE2 and CH_LFE3 shall be assigned to the two LFE channels in the order that minimizes the maximum azimuth distance of the two LFE channels to the assigned CH_LFE2 and CH_LFE3 nominal azimuth positions.

If an input or output format contains more than 2 LFE channels, then two of the LFE channels from the considered setup shall be selected and assigned that minimize the maximum azimuth distance to the CH_LFE2 and CH_LFE3 nominal azimuth positions. The labels CH_LFE2 and CH_LFE3 shall be assigned as in the case of two LFE channels. The remaining LFE channels shall not be considered further in the calculation of downmix coefficients, i.e. the corresponding lines/columns of the downmix matrix shall remain filled with zeros.

10.3.4.3 Handling for unknown input channels

If the label CH_EMPTY is assigned to an input channel, this channel shall be considered unknown to the rules-based initialization and the downmix coefficients for mapping this input channel to the output channels shall be derived as specified in subclause 10.3.4.6.7.

10.3.4.4 Handling for unknown output formats

If the output format contains at least one channel with the label CH_EMPTY assigned to it, or if at least one channel label is assigned to more than one channel of the output format, the output format shall be considered unknown and the derivation of the downmixing coefficients shall be carried out as specified in subclause 10.3.4.6.7. The rules-based derivation of downmix coefficients shall not be applied for unknown output formats.

10.3.4.5 Handling of deviations from standard loudspeaker positions

If any of the below conditions are not met, the rules-based initialization is considered to have failed, the output format shall be considered to be unknown, and the downmixing gains shall be obtained as defined in subclause 10.3.4.6.7.

— The absolute values of $r_{azi,A}$ and $r_{ele,A}$ shall not exceed 35 and 55 degrees, respectively. The minimum angle between any loudspeaker pair (without LFE channels) shall not be smaller than 15 degrees.

— The values of $r_{azi,A}$ shall be such that the ordering by azimuth angles of the horizontal loudspeakers does not change. Likewise, the ordering of the height and low loudspeakers shall not change.

— The values of $r_{ele,A}$ shall be such that the ordering by elevation angles of loudspeakers which are (approximately) above/below each other does not change. To verify this, the following procedure is applied:

  For each row of Table 167 which contains two or three channels of the output format, do:

  — Order the channels by elevation without randomization;

  — Order the channels by elevation with considering randomization;

  — If the two orderings differ, return an initialization error.

10.3.4.6 Rules-based initialization algorithm

10.3.4.6.1 General

The rules-based initialization algorithm is defined in the following subclauses. The algorithm shall not be applied if the output format is considered unknown as defined in the previous subclause. For clarity the following description makes use of intermediate parameters listed in Table 161 but an implementation may omit the explicit use of these intermediate parameters.

Table 161 — Format converter initialization intermediate parameters

S	Vector of converter source channels [input channel indices]
D	Vector of converter destination channels [output channel indices]
G	Vector of converter gains [linear]
E	Vector of converter EQ indices
$\mathbf{G}_{EQ}$	Matrix containing equalizer gain values for all EQ indices and frequency bands

The intermediate parameters describe the dowmixing parameters according to the mapping, i.e. as sets of parameters S_i, D_i, G_i, E_i, per mapping i.

The format converter initialization output parameters are derived as described in the following steps:

10.3.4.6.2 Random setups pre-processing

Random output loudspeaker setups, i.e. output setups that contain loudspeakers at positions deviating from the positions defined for the desired output format are signalled by specifying the loudspeaker position deviation angles as input parameters $r_{azi,A}$ and $r_{ele,A}$. The angle deviations are taken into account as a pre-processing step.

Modify the channels' azimuth and elevation angles according to Table 162 by adding $r_{azi,A}$ and $r_{ele,A}$ to the corresponding channels' azimuth and elevation angles.

10.3.4.6.3 Derivation of input channel/output channel mapping parameters

The parameters vectors **S**, **D**, **G**, **E** define the mapping of input channels to output channels. For each mapping i from an input channel to an output channel with non-zero downmix gain they define the downmix gain as well as an equalizer index that indicates which equalizer curve has to be applied to the input channel under consideration in mapping i.

The elements of the parameter vectors **S**, **D**, **G**, **E** are derived by the following algorithm:

Initialize the mapping counter i: $i = 1$.

For each input channel, ignoring channels with label CH_EMPTY assigned to them:

If the input channel also exists in the output format (e.g. input channel under consideration is CH_M_R030 and channel CH_M_R030 exists in the output format), then:

- S_i = index of source channel in input
  EXAMPLE channel CH_M_R030 in ChannelConfiguration 6 is at second place according to Table 163, i.e. has index 2 in this format.
- D_i = index of same channel in output
- G_i = 1.0
- E_i = 0
- $i = i + 1$

Else (i.e. if the input channel does not exist in the output format)

- search the first entry of this channel in the **Source** column of Table 164, for which the channels in the corresponding row of the **Destination** column exist. The ALL_U destination shall be considered valid (i.e. the relevant output channels exist) if the output format contains at least one "CH_U_" channel. The ALL_M destination shall be considered valid (i.e. the relevant output channels exist) if the output format contains at least one "CH_M_" channel. If for no entry in Table 164 corresponding to the input channel the channels in the **Destination** column exist, the rules-based initialization shall terminate and the downmix gains shall be derived according to subclause 10.3.4.6.7.

  If **Destination** column contains ALL_U, then:

  For each output channel x with "CH_U_" in its name, do:

  - S_i = index of source channel in input
  - D_i = index of channel x in output
  - G_i = (value of Gain column) / sqrt(number of "CH_U_" output channels)
  - E_i = value of EQ column
  - $i = i + 1$

  Else if **Destination** column contains ALL_M, then:

  For each output channel x with "CH_M_" in its name, do:

  - S_i = index of source channel in input
  - D_i = index of channel x in output
  - G_i = (value of Gain column) / sqrt(number of "CH_M_" output channels)

— E_i = value of EQ column
— $i = i + 1$

Else If there is one channel in the **Destination** column, then:

— S_i = index of source channel in input
— D_i = index of destination channel in output
— G_i = value of Gain column
— E_i = value of EQ column
— $i = i + 1$

Else (two channels in **Destination** column)

— S_i = index of source channel in input
— D_i = index of first destination channel in output
— G_i = (value of Gain column) × g_1
— E_i = value of EQ column
— $i = i + 1$

— $S_i = S_{i-1}$
— D_i = index of second destination channel in output
— G_i = (value of Gain column) × g_2
— $E_i = E_{i-1}$
— $i = i + 1$

The gains g_1 and g_2 are computed by applying tangent law amplitude panning in the following way.

— Unwrap source destination channel azimuth angles to be positive.

— The azimuth angles of the destination channels are α_1 and α_2 (see Table 162).

— The azimuth angle of the source channel (= panning target) is α_{src}.

— $\alpha_0 = \frac{|\alpha_1 - \alpha_2|}{2}$

— $\alpha_{center} = \frac{\alpha_1 + \alpha_2}{2}$

— $\alpha = (\alpha_{center} - \alpha_{src}) \cdot \mathrm{sgn}(\alpha_2 - \alpha_1)$

— $g_1 = \frac{g}{\sqrt{1+g^2}}, \quad g_2 = \frac{1}{\sqrt{1+g^2}} \qquad \text{with} \qquad g = \frac{\tan\alpha_0 - \tan\alpha + 10^{-10}}{\tan\alpha_0 + \tan\alpha + 10^{-10}}$

10.3.4.6.4 Derivation of equalizer gains G_{EQ}

G_{EQ} comprises a set of gain values for each frequency band k and equalizer index e. The 5 predefined equalizers are combinations of different peak filters. Each equalizer is a serial cascade of one or more peak filters and an associated gain:

$$G_{\mathrm{EQ},e}^{k} = 10^{\frac{g}{20}} \prod_{n=1}^{N} \mathrm{peak}\left(band(k)\frac{f_s}{2}, P_{f,n}, P_{Q,n}, P_{g,n} \right)$$

where *band(k)* is the normalized centre frequency of frequency band k, specified in Table 165, f_s is the sampling frequency, and function *peak()* is for negative G:

$$\mathrm{peak}\left(b, f, Q, \mathrm{G}\right) = \sqrt{\frac{b^4 + \left(\frac{1}{Q^2} - 2\right) f^2 b^2 + f^4}{b^4 + \left(\frac{10^{\frac{-G}{10}}}{Q^2} - 2\right) f^2 b^2 + f^4}}$$

and otherwise

$$\mathrm{peak}\left(b, f, Q, \mathrm{G}\right) = \sqrt{\frac{b^4 + \left(\frac{10^{\frac{G}{10}}}{Q^2} - 2\right) f^2 b^2 + f^4}{b^4 + \left(\frac{1}{Q^2} - 2\right) f^2 b^2 + f^4}}$$

The parameters for the equalizers are specified in Table 166.

10.3.4.6.5 Post-processing for random setups

Once the output parameters are computed, they are modified according to the specific random azimuth and elevations angles. This step only has to be carried out, if not all $r_{\mathrm{ele},A}$ are zero. The post-processing algorithm proceeds as follows.

For each element i in D_i, do:

if the output channel with index D_i is a horizontal channel by definition (i.e. output channel label contains the label '_M_'), **and**

if this output channel is now a height channel (elevation in range 0..60 degrees), **and**

if input channel with index S_i is a height channel (i.e. label contains '_U_'), **then**

- h = min(elevation of randomized output channel, 35)/35;
- $G_{\mathrm{comp}} = h \cdot \frac{1}{0.85} + (1-h)$;
- Apply compensation gain to DMX gain: $G_i = G_i \cdot G_{\mathrm{comp}}$;
- Define new equalizer with a new index e, where $G_{EQ,e}^{k} = h + (1-h) \cdot \mathrm{G}_{EQ,E_i}^{k}$;
- $E_i = e$;

else if input channel with index S_i is a horizontal channel (label contains '_M_')

- h = min(elevation of randomized output channel, 35)/35;
- Define new equalizer with a new index e, where ;

$$G^k_{EQ,e} = h \cdot \mathrm{G}^k_{EQ,5} + (1-h) \cdot \mathrm{G}^k_{EQ,E_i};$$

— $E_i = e$.

Explanation of the post-processing steps defined above:

h is a normalized elevation parameter indicating the elevation of a nominally horizontal output channel ('_M_') due to a random setup elevation offset $r_{ele,A}$. For zero elevation offset *h*=0 follows and effectively no post-processing is applied.

The rules table (Table 164) in general applies a gain of 0.85 when mapping an upper input channel ('_U_' in channel label) to one or several horizontal output channels ('_M_' in channel label(s)). In case the output channel gets elevated due to a random setup elevation offset $r_{ele,A}$, the gain of 0.85 is partially (0<*h*<1) or fully (*h*=1) compensated for. Similarly the equalizer definitions fade towards a flat EQ-curve ($G^k_{EQ,e} = const. = 1.0$) for *h* approaching *h* =1.

In case a horizontal input channel gets mapped to an output channel that gets elevated due to a random setup elevation offset $r_{ele,A}$, the equalizer $G^k_{EQ,5}$ is partially (0<*h*<1) or fully (*h*=1) applied.

10.3.4.6.6 Derivation of rules-based initialization downmix matrix:

$\mathbf{M}_{\mathrm{DMX}}$ is derived by rearranging the temporary parameters from the mapping-oriented representation (enumerated by mapping counter *i*) to a channel-oriented representation as defined in the following:

Initialize $\mathbf{M}^k_{\mathrm{DMX}}$ as an $N_{\mathrm{out}} \times N_{\mathrm{in}}$ zero matrix for all processing bands *k*.

For each *i* do:

If ($E_i = 0$)

$$M^k_{\mathrm{DMX},A,B} = G_i \quad \text{for} \quad A = D_i,\ B = S_i,\ 0 \le k < K$$

Else

$$M^k_{\mathrm{DMX},A,B} = G_i \cdot G^k_{EQ,E_i} \quad \text{for} \quad A = D_i,\ B = S_i,\ 1 \le k < K$$

where $\mathbf{M}^k_{\mathrm{DMX},A,B}$ denotes the matrix element in the *A*th row and *B*th column of $\mathbf{M}^k_{\mathrm{DMX}}$. Note that after the rules-based initialization this matrix of downmix coefficients will contain columns of zeros, if unknown channels are present in the input format. Those columns are filled with downmix gains as specified in subclause 10.3.4.6.7.

10.3.4.6.7 VBAP-based downmix coefficients derivation

This subclause defines how downmix gains are derived in a generic manner using VBAP in the case of unknown output formats or unknown input channels. The following restrictions apply.

— If the target setup contains at least one LFE, then map each LFE channel directly to the LFE of the target setup that minimizes the azimuth angle deviation. No VBAP-based downmix coefficients derivation shall be applied for the LFE channels. The downmix coefficient for the direct mapping shall be set to unity gain, i.e. to 1.0.

— Otherwise apply the VBAP-based downmix coefficients derivation defined in the following also to the LFE channels.

Handling of unknown output formats:

In case the output format is considered unknown, the downmix coefficients for all input channels shall be derived as follows.

Each channel of the input setup is regarded as a static audio object at the position defined by the azimuth and elevation angles associated with the input channel. For each input channel the mixing gains to all output loudspeakers are calculated as VBAP panning gains $\mathbf{g}_{\mathbf{scaled}}$ according to subclause 8.4.4, where the same output format shall be signalled to the VBAP algorithm as to the format converter. The panning gain vectors $\mathbf{g}_{\mathbf{scaled}}$ shall be post-processed according to subclause 10.3.4.6.8.

The downmix matrix $\mathbf{M}_{\mathrm{DMX}}^{k}$ is finally derived by filling each matrix column with the post-processed panning gain vector elements of the corresponding input channel, independently of the processing band index k.

Handling of unknown input channels:

In case the input format contains unknown input channels, the downmix coefficients for these channels shall be derived as follows.

Each unknown channel of the input setup is regarded as a static audio object at the position defined by the azimuth and elevation angles associated with the input channel. For each unknown input channel the mixing gains to all output loudspeakers are calculated as VBAP panning gains $\mathbf{g}_{\mathbf{scaled}}$ according to subclause 8.4.4, where the same output format shall be signalled to the VBAP algorithm as to the format converter. The panning gain vectors $\mathbf{g}_{\mathbf{scaled}}$ shall be post-processed according to subclause 10.3.4.6.8.

The downmix matrix $\mathbf{M}_{\mathrm{DMX}}^{k}$ is finally derived by filling each matrix column, corresponding to an unknown input channel, with the post-processed panning gain vector elements of the corresponding unknown input channel, independently of the processing band index k.

10.3.4.6.8 VBAP gains post-processing

The mixing gains obtained from the VBAP rendering algorithm shall be post-processed to avoid excessive use of phantom sources. Therefore, small matrix gains are set to zero, followed by a renormalization of the panning gains to ensure energy-preservation.

For each panning gain vector $\mathbf{g}_{\mathbf{scaled}}$ do:

— If the vector contains at least one panning gain that exceeds the threshold value 0.3, then;

— Set all vector elements smaller or equal to 0,3 to the value 0,0;

— Normalize the gain vector such that the sum of squares of the vector elements remains the same as before the post-processing.

10.3.4.7 Format converter initialization tables

Table 162 lists channel labels, corresponding azimuth and elevation angles, and associated sectors. The sectors are defined as points on the unit sphere, whose azimuth/elevation angles are within or on the borders of the intervals given by the azimuth/elevation start and end values in the table, connecting

azimuth start and end values in a counter-clockwise direction and connecting elevation start and end values in the direction of increasing elevation angles.

Table 162 — Channels definitions: Channel labels, corresponding azimuth and elevation angles, and associated sectors

Loudspeaker Geometry (as defined in ISO/IEC 23001-8)	**Channel**	**Azimuth** [deg]	**Elevation** [deg]	**Azimuth start angle of sector** [deg]	**Azimuth end angle of sector** [deg]	**Elevation start angle of sector** [deg]	**Elevation end angle of sector** [deg]	**Ch. is LFE**	**Position is relative**
	CH_EMPTY	n/a	n/a	n/a	n/a	n/a	n/a	0	0
0	CH_M_L030	+30	0	+23	+37	-9	+20	0	0
1	CH_M_R030	-30	0	-37	-23	-9	+20	0	0
2	CH_M_000	0	0	-7	+7	-9	+20	0	0
3	CH_LFE1	0	n/a	n/a	n/a	n/a	n/a	1	0
4	CH_M_L110	+110	0	+101	+124	-45	+20	0	0
5	CH_M_R110	-110	0	-124	-101	-45	+20	0	0
6	CH_M_L022	+22	0	+8	+22	-9	+20	0	0
7	CH_M_R022	-22	0	-22	-8	-9	+20	0	0
8	CH_M_L135	+135	0	125	142	-45	+20	0	0
9	CH_M_R135	-135	0	-142	-125	-45	+20	0	0
10	CH_M_180	180	0	158	-158	-45	+20	0	0
13	CH_M_L090	+90	0	+76	+100	-45	+20	0	0
14	CH_M_R090	-90	0	-100	-76	-45	+20	0	0
15	CH_M_L060	+60	0	+53	+75	-9	+20	0	0
16	CH_M_R060	-60	0	-75	-53	-9	+20	0	0
17	CH_U_L030	+30	+35	+11	+37	+21	+60	0	0
18	CH_U_R030	-30	+35	-37	-11	+21	+60	0	0
19	CH_U_000	0	+35	-10	+10	+21	+60	0	0
20	CH_U_L135	+135	+35	+125	+157	+21	+60	0	0
21	CH_U_R135	-135	+35	-157	-125	+21	+60	0	0
22	CH_U_180	180	+35	+158	-158	+21	+60	0	0
23	CH_U_L090	+90	+35	+67	+100	+21	+60	0	0
24	CH_U_R090	-90	+35	-100	-67	+21	+60	0	0
25	CH_T_000	0	+90	-180	+180	+61	+90	0	0
26	CH_LFE2	+45	n/a	n/a	n/a	n/a	n/a	1	0
27	CH_L_L045	+45	-15	+11	+75	-45	-10	0	0
28	CH_L_R045	-45	-15	-75	-11	-45	-10	0	0
29	CH_L_000	0	-15	-10	+10	-45	-10	0	0
30	CH_U_L110	+110	+35	+101	+124	+21	+60	0	0
31	CH_U_R110	-110	+35	-124	-101	+21	+60	0	0
32	CH_U_L045	+45	+35	+38	+66	+21	+60	0	0
33	CH_U_R045	-45	+35	-66	-38	+21	+60	0	0
34	CH_M_L045	+45	0	+38	+52	-9	+20	0	0
35	CH_M_R045	-45	0	-52	-38	-9	+20	0	0
36	CH_LFE3	-45	n/a	n/a	n/a	n/a	n/a	1	0
37	CH_M_LSCR	+60	0	n/a	n/a	n/a	n/a	0	1
38	CH_M_RSCR	-60	0	n/a	n/a	n/a	n/a	0	1
39	CH_M_LSCH	+30	0	n/a	n/a	n/a	n/a	0	1
40	CH_M_RSCH	-30	0	n/a	n/a	n/a	n/a	0	1
41	CH_M_L150	+150	0	143	157	-45	+20	0	0
42	CH_M_R150	-150	0	-157	-143	-45	+20	0	0

Table 163 — Formats with corresponding number of channels and channel ordering

Loudspeaker layout index or ChannelConfiguration as defined in ISO/IEC 23001-8	Number of channels	Channels (with ordering)
1	1	CH_M_000
2	2	CH_M_L030, CH_M_R030
3	3	CH_M_L030, CH_M_R030, CH_M_000
4	4	CH_M_L030, CH_M_R030, CH_M_000, CH_M180
5	5	CH_M_L030, CH_M_R030, CH_M_000, CH_M_L110, CH_M_R110
6	6	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110
7	8	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_M_L060, CH_M_R060
8		n.a.
9	3	CH_M_L030, CH_M_R030, CH_M_180
10	4	CH_M_L030, CH_M_R030, CH_M_L110, CH_M_R110
11	7	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_M_180
12	8	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_M_L135, CH_M_R135
13	24	CH_M_L060, CH_M_R060, CH_M_000, CH_LFE2, CH_M_L135, CH_M_R135, CH_M_L030, CH_M_R030, CH_M_180, CH_LFE3, CH_M_L090, CH_M_R090, CH_U_L045, CH_U_R045, CH_U_000, CH_T_000, CH_U_L135, CH_U_R135, CH_U_L090, CH_U_R090, CH_U_180, CH_L_000, CH_L_L045, CH_L_R045
14	8	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_U_L030, CH_U_R030
15	12	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE2, CH_M_L135, CH_M_R135, CH_LFE3, CH_M_L090, CH_M_R090, CH_U_L045, CH_U_R045, CH_U_180
16	10	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_U_L030, CH_U_R030, CH_U_L110, CH_U_R110
17	12	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_U_L030, CH_U_R030, CH_U_000, CH_U_L110, CH_U_R110, CH_T_000
18	14	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_M_L150, CH_M_R150, CH_U_L030, CH_U_R030, CH_U_000, CH_U_L110, CH_U_R110, CH_T_000
19	12	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L135, CH_M_R135, CH_M_L090, CH_M_R090, CH_U_L030, CH_U_R030, CH_U_L135, CH_U_R135
20	14	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L135, CH_M_R135, CH_M_L090, CH_M_R090, CH_U_L045, CH_U_R045, CH_U_L135, CH_U_R135, CH_M_LSCR, CH_M_RSCR

Table 164 — Converter rules matrix

Source	Destination	Gain	EQ index
CH_M_000	CH_M_L022, CH_M_R022	1.0	0 (off)
CH_M_000	CH_M_L030, CH_M_R030	1.0	0 (off)
CH_M_L022	CH_M_000, CH_M_L030	1.0	0 (off)
CH_M_L022	CH_M_L030	1.0	0 (off)
CH_M_R022	CH_M_000, CH_M_R030	1.0	0 (off)
CH_M_R022	CH_M_R030	1.0	0 (off)
CH_M_L045	CH_M_L030, CH_M_L060	1.0	0 (off)
CH_M_L045	CH_M_L030	1.0	0 (off)
CH_M_R045	CH_M_R030, CH_M_R060	1.0	0 (off)
CH_M_R045	CH_M_R030	1.0	0 (off)
CH_M_L060	CH_M_L045, CH_M_L090	1.0	0 (off)
CH_M_L060	CH_M_L030, CH_M_L090	1.0	0 (off)
CH_M_L060	CH_M_L045, CH_M_L110	1.0	0 (off)
CH_M_L060	CH_M_L030, CH_M_L110	1.0	0 (off)
CH_M_L060	CH_M_L030	0.8	0 (off)
CH_M_R060	CH_M_R045, CH_M_R090	1.0	0 (off)
CH_M_R060	CH_M_R030, CH_M_R090	1.0	0 (off)
CH_M_R060	CH_M_R045, CH_M_R110	1.0	0 (off)
CH_M_R060	CH_M_R030, CH_M_R110	1.0	0 (off)
CH_M_R060	CH_M_R030	0.8	0 (off)
CH_M_L090	CH_M_L060, CH_M_L110	1.0	0 (off)
CH_M_L090	CH_M_L045, CH_M_L110	1.0	0 (off)
CH_M_L090	CH_M_L030, CH_M_L110	1.0	0 (off)
CH_M_L090	CH_M_L030	0.8	0 (off)
CH_M_R090	CH_M_R060, CH_M_R110	1.0	0 (off)
CH_M_R090	CH_M_R045, CH_M_R110	1.0	0 (off)
CH_M_R090	CH_M_R030, CH_M_R110	1.0	0 (off)
CH_M_R090	CH_M_R030	0.8	0 (off)
CH_M_L110	CH_M_L135	1.0	0 (off)
CH_M_L110	CH_M_L090	0.8	0 (off)
CH_M_L110	CH_M_L045	0.8	0 (off)
CH_M_L110	CH_M_L030	0.8	0 (off)
CH_M_R110	CH_M_R135	1.0	0 (off)
CH_M_R110	CH_M_R090	0.8	0 (off)
CH_M_R110	CH_M_R045	0.8	0 (off)
CH_M_R110	CH_M_R030	0.8	0 (off)
CH_M_L135	CH_M_L110	1.0	0 (off)
CH_M_L135	CH_M_L150	1.0	0 (off)
CH_M_L135	CH_M_L090	0.8	0 (off)
CH_M_L135	CH_M_L045	0.8	0 (off)
CH_M_L135	CH_M_L030	0.8	0 (off)
CH_M_R135	CH_M_R110	1.0	0 (off)
CH_M_R135	CH_M_R150	1.0	0 (off)
CH_M_R135	CH_M_R090	0.8	0 (off)
CH_M_R135	CH_M_R045	0.8	0 (off)
CH_M_R135	CH_M_R030	0.8	0 (off)
CH_M_L150	CH_M_L135	1.0	0 (off)
CH_M_L150	CH_M_L110	1.0	0 (off)
CH_M_L150	CH_M_L045	0.8	0 (off)
CH_M_L150	CH_M_L030	0.8	0 (off)
CH_M_R150	CH_M_R135	1.0	0 (off)

Source	Destination	Gain	EQ index
CH_M_R150	CH_M_R110	1.0	0 (off)
CH_M_R150	CH_M_R045	0.8	0 (off)
CH_M_R150	CH_M_R030	0.8	0 (off)
CH_M_180	CH_M_R150, CH_M_L150	1.0	0 (off)
CH_M_180	CH_M_R135, CH_M_L135	1.0	0 (off)
CH_M_180	CH_M_R110, CH_M_L110	1.0	0 (off)
CH_M_180	CH_M_R090, CH_M_L090	0.8	0 (off)
CH_M_180	CH_M_R045, CH_M_L045	0.6	0 (off)
CH_M_180	CH_M_R030, CH_M_L030	0.6	0 (off)
CH_U_000	CH_U_L030, CH_U_R030	1.0	0 (off)
CH_U_000	CH_M_L030, CH_M_R030	0.85	0 (off)
CH_U_L045	CH_U_L030	1.0	0 (off)
CH_U_L045	CH_M_L045	0.85	1
CH_U_L045	CH_M_L030	0.85	1
CH_U_R045	CH_U_R030	1.0	0 (off)
CH_U_R045	CH_M_R045	0.85	1
CH_U_R045	CH_M_R030	0.85	1
CH_U_L030	CH_U_L045	1.0	0 (off)
CH_U_L030	CH_M_L030	0.85	1
CH_U_R030	CH_U_R045	1.0	0 (off)
CH_U_R030	CH_M_R030	0.85	1
CH_U_L090	CH_U_L030, CH_U_L110	1.0	0 (off)
CH_U_L090	CH_U_L030, CH_U_L135	1.0	0 (off)
CH_U_L090	CH_U_L045	0.8	0 (off)
CH_U_L090	CH_U_L030	0.8	0 (off)
CH_U_L090	CH_M_L045, CH_M_L110	0.85	2
CH_U_L090	CH_M_L030, CH_M_L110	0.85	2
CH_U_L090	CH_M_L030	0.85	2
CH_U_R090	CH_U_R030, CH_U_R110	1.0	0 (off)
CH_U_R090	CH_U_R030, CH_U_R135	1.0	0 (off)
CH_U_R090	CH_U_R045	0.8	0 (off)
CH_U_R090	CH_U_R030	0.8	0 (off)
CH_U_R090	CH_M_R045, CH_M_R110	0.85	2
CH_U_R090	CH_M_R030, CH_M_R110	0.85	2
CH_U_R090	CH_M_R030	0.85	2
CH_U_L110	CH_U_L135	1.0	0 (off)
CH_U_L110	CH_U_L090	0.8	0 (off)
CH_U_L110	CH_U_L045	0.8	0 (off)
CH_U_L110	CH_U_L030	0.8	0 (off)
CH_U_L110	CH_M_L110	0.85	2
CH_U_L110	CH_M_L045	0.85	2
CH_U_L110	CH_M_L030	0.85	2
CH_U_R110	CH_U_R135	1.0	0 (off)
CH_U_R110	CH_U_R090	0.8	0 (off)
CH_U_R110	CH_U_R045	0.8	0 (off)
CH_U_R110	CH_U_R030	0.8	0 (off)
CH_U_R110	CH_M_R110	0.85	2
CH_U_R110	CH_M_R045	0.85	2
CH_U_R110	CH_M_R030	0.85	2
CH_U_L135	CH_U_L110	1.0	0 (off)
CH_U_L135	CH_U_L090	0.8	0 (off)
CH_U_L135	CH_U_L045	0.8	0 (off)
CH_U_L135	CH_U_L030	0.8	0 (off)

Source	Destination	Gain	EQ index
CH_U_L135	CH_M_L110	0.85	2
CH_U_L135	CH_M_L045	0.85	2
CH_U_L135	CH_M_L030	0.85	2
CH_U_R135	CH_U_R110	1.0	0 (off)
CH_U_R135	CH_U_R090	0.8	0 (off)
CH_U_R135	CH_U_R045	0.8	0 (off)
CH_U_R135	CH_U_R030	0.8	0 (off)
CH_U_R135	CH_M_R110	0.85	2
CH_U_R135	CH_M_R045	0.85	2
CH_U_R135	CH_M_R030	0.85	2
CH_U_180	CH_U_R135, CH_U_L135	1.0	0 (off)
CH_U_180	CH_U_R110, CH_U_L110	1.0	0 (off)
CH_U_180	CH_M_180	0.85	2
CH_U_180	CH_M_R110, CH_M_L110	0.85	2
CH_U_180	CH_U_R030, CH_U_L030	0.8	0 (off)
CH_U_180	CH_M_R030, CH_M_L030	0.85	2
CH_T_000	ALL_U	0.8	3
CH_T_000	ALL_M	0.8	4
CH_L_000	CH_M_000	1.0	0 (off)
CH_L_000	CH_M_L030, CH_M_R030	1.0	0 (off)
CH_L_L045	CH_M_L045	1.0	0 (off)
CH_L_L045	CH_M_L030	1.0	0 (off)
CH_L_R045	CH_M_R045	1.0	0 (off)
CH_L_R045	CH_M_R030	1.0	0 (off)
CH_LFE2	CH_LFE3	1.0	0 (off)
CH_LFE2	CH_M_L030, CH_M_R030	1.0	0 (off)
CH_LFE3	CH_LFE2	1.0	0 (off)
CH_LFE3	CH_M_L030, CH_M_R030	1.0	0 (off)

Table 165 — Normalized centre frequencies of the 71 filter-bank bands

Normalized frequency [0, 1]
0,004 583 30
0,000 833 33
0,002 083 30
0,005 875 00
0,009 791 70
0,014 292 00
0,019 792 00
0,027 000 00
0,035 417 00
0,042 625 00
0,056 750 00
0,072 375 00
0,088 000 00
0,103 620 00
0,119 250 00
0,134 870 00
0,150 500 00
0,166 120 00
0,181 750 00
0,197 370 00
0,213 000 00

Normalized frequency [0, 1]
0,228 620 00
0,244 250 00
0,259 880 00
0,275 500 00
0,291 130 00
0,306 750 00
0,322 380 00
0,338 000 00
0,353 630 00
0,369 250 00
0,384 880 00
0,400 500 00
0,416 130 00
0,431 750 00
0,447 380 00
0,463 000 00
0,478 630 00
0,494 250 00
0,509 870 00
0,525 500 00
0,541 120 00
0,556 750 00
0,572 370 00
0,588 000 00
0,603 620 00
0,619 250 00
0,634 870 00
0,650 500 00
0,666120 00
0,681 750 00
0,697 370 00
0,713 000 00
0,728 620 00
0,744 250 00
0,759 870 00
0,775 500 00
0,791 120 00
0,806 750 00
0,822 370 00
0,838 000 00
0,853 620 00
0,869 250 00
0,884 870 00
0,900 500 00
0,916 120 00
0,931 750 00
0,947 370 00
0,963 000 00
0,974 540 00
0,999 040 00

Table 166 — Equalizer parameters

Equalizer	P_f [Hz]	P_Q	P_g[dB]	g [dB]
$G_{EQ,1}$	12 000	0,3	-2	1,0
$G_{EQ,2}$	12 000	0,3	-3,5	1,0
$G_{EQ,3}$	200,1 300, 600	0,3, 0,5, 1,0	-6,5, 1,8, 2,0	0,7
$G_{EQ,4}$	5 000, 1 100	1,0, 0,8	4,5, 1,8	-3,1
$G_{EQ,5}$	35	0,25	-1,3	1,0

Table 167 — Vertically corresponding channels: Each row lists channels which are considered to be above/below each other

CH_L_000	CH_M_000	CH_U_000
CH_L_L045	CH_M_L030	CH_U_L030
CH_L_L045	CH_M_L030	CH_U_L045
CH_L_L045	CH_M_L045	CH_U_L030
CH_L_L045	CH_M_L045	CH_U_L045
CH_L_L045	CH_M_L060	CH_U_L030
CH_L_L045	CH_M_L060	CH_U_L045
CH_L_R045	CH_M_R030	CH_U_R030
CH_L_R045	CH_M_R030	CH_U_R045
CH_L_R045	CH_M_R045	CH_U_R030
CH_L_R045	CH_M_R045	CH_U_R045
CH_L_R045	CH_M_R060	CH_U_R030
CH_L_R045	CH_M_R060	CH_U_R045
CH_M_180	CH_U_180	
CH_M_L090	CH_U_L090	
CH_M_L110	CH_U_L110	
CH_M_L135	CH_U_L135	
CH_M_L090	CH_U_L110	
CH_M_L090	CH_U_L135	
CH_M_L110	CH_U_L090	
CH_M_L110	CH_U_L135	
CH_M_L135	CH_U_L090	
CH_M_L135	CH_U_L135	
CH_M_R090	CH_U_R090	
CH_M_R110	CH_U_R110	
CH_M_R135	CH_U_R135	
CH_M_R090	CH_U_R110	
CH_M_R090	CH_U_R135	
CH_M_R110	CH_U_R090	
CH_M_R110	CH_U_R135	
CH_M_R135	CH_U_R090	
CH_M_R135	CH_U_R135	

10.3.5 Audio signal processing

10.3.5.1 General

The audio processing block of the format converter obtains time domain audio samples for N_{in} channels from the core decoder and generates a downmixed time domain audio output signal consisting of N_{out} channels.

The processing takes as input

— the audio data decoded by the core decoder, and

— the static downmix matrix $\mathbf{M}_{DMX}$ returned by the initialization of the format converter.

It returns an N_{out}-channel time domain output signal for the OutConf channel configuration signalled during the initialization of the format converter.

The format converter operates on contiguous, non-overlapping frames of length L=2048 time domain samples of the input audio signals and outputs one frame of L samples per processed input frame of length L.

10.3.5.2 T/F-transform (hybrid QMF analysis)

As the first processing step, the converter transforms L=2048 samples of the N_{in} channel time domain input signal $\left[\tilde{y}^{\nu}_{\mathrm{ch},1} \cdots \tilde{y}^{\nu}_{\mathrm{ch},N_{\mathrm{in}}} \right] = \tilde{\mathbf{y}}^{\nu}_{\mathrm{ch}}$ to a hybrid QMF N_{in} channel signal representation consisting of $L_n = 32$ QMF time slots (slot index n) and $K = 71$ frequency bands (band index k). A QMF analysis according to ISO/IEC 14496-3:2009, subclause 4.6.18.4, is performed first:

$$\left[\hat{y}^{n,k}_{\mathrm{ch},1} \cdots \hat{y}^{n,k}_{\mathrm{ch},N_{\mathrm{in}}} \right] = \hat{\mathbf{y}}^{n,k}_{\mathrm{ch}} = \mathrm{QmfAnalysis}(\tilde{\mathbf{y}}^{\nu}_{\mathrm{ch}}) \text{ with } 0 \le \nu < L \text{ and } 0 \le n < L_n$$

followed by a hybrid analysis:

$$\left[y^{n,k}_{\mathrm{ch},1} \cdots y^{n,k}_{\mathrm{ch},N_{\mathrm{in}}} \right] = \mathbf{y}^{n,k}_{\mathrm{ch}} = \mathrm{HybridAnalysis}(\hat{\mathbf{y}}^{n,k}_{\mathrm{ch}})$$

The hybrid filtering shall be carried out as specified in ISO/IEC 14496-3:2009, 8.6.4.3 for the 10,20 bands configuration of parametric stereo, resulting in a 71-band hybrid QMF domain representation.

10.3.5.3 Covariance analysis

Note that for clarity the frequency band parameter (superscript k) is omitted in the following equations if it is not required for the presentation.

Let F be a monotonically increasing frame index denoting the current frame of input data, e.g. $\mathbf{y}^{F,n}_{\mathrm{ch}} = \mathbf{y}^{n}_{\mathrm{ch}}$ for frame F, starting at F = 0 for the first frame of input data after initialization of the format converter. An analysis frame of length $2L_n$ is formulated from the input hybrid QMF spectra as:

$$\mathbf{y}^{F,n}_{\mathrm{in,ch}} = \begin{cases} 0 & \text{,for } 0 \le n < L_n \quad \text{,} F = 0 \\ \mathbf{y}^{F-1,n+L_n}_{\mathrm{in,ch}} & \text{,for } 0 \le n < L_n \quad \text{,} F > 0 \\ \mathbf{y}^{F,n-L_n}_{\mathrm{ch}} & \text{,for } L_n \le n < 2L_n \quad \text{,} F \ge 0 \end{cases}$$

Note that $\mathbf{y}_{\text{in, ch}}^{F,n}$ is a row vector with N_{in} elements in case of N_{in} input channels. The covariance matrix is analysed from four quarter segments of $\mathbf{y}_{\text{in, ch}}^{F,n}$, so that:

$$\mathbf{C}_{y,q}^{F} = \sum_{n=16q}^{16q+15} \left(\mathbf{y}_{\text{in, ch}}^{F,n}\right)^{\text{T}} \left(\mathbf{y}_{\text{in, ch}}^{F,n}\right)^{*} \quad \text{,for} \quad q = 0,1,2,3\,,$$

where

$(\cdot)^{\text{T}}$ denotes the transpose;

$(\cdot)^{*}$ denotes the complex conjugate of a variable;

$\mathbf{C}_{y,q}^{F}$ is an $N_{\text{in}} \times N_{\text{in}}$ matrix for each q = 0,1,2,3.

$\mathbf{C}_{y,0}^{F}$ and $\mathbf{C}_{y,1}^{F}$ are the same as $\mathbf{C}_{y,2}^{F-1}$ and $\mathbf{C}_{y,3}^{F-1}$, correspondingly, and are not required to be re-calculated. The covariance matrices of the four quarter segments are added with centre weighting assuming a staircase shape:

$$\mathbf{C}_{y,\text{sum}}^{\text{F}} = \mathbf{C}_{y,0}^{F} + 4\mathbf{C}_{y,1}^{F} + 4\mathbf{C}_{y,2}^{F} + \mathbf{C}_{y,3}^{F}$$

The final estimation for the covariance matrix $\mathbf{C}_{y}^{F}$ is obtained by modifying the entries of $\mathbf{C}_{y,\text{sum}}^{F}$ with a small channel dependent offset

$$C_{y,a,b} = (1 + 0.0002\sqrt{ab})C_{y,\text{sum},a,b}$$

where the two indices in a notation $C_{y,a,b}$ denote the matrix element in the ath row and bth column of $\mathbf{C}_{y}$. From the covariance matrix $\mathbf{C}_{y}$ inter-channel correlation coefficients between the channels A and B are derived as:

$$ICC_{A,B} = \sqrt{\frac{\left|C_{y,A,B}\right|^{2}}{\text{eps} + C_{y,A,A} \cdot C_{y,B,B}}}$$

10.3.5.4 Phase-alignment matrix formulation

10.3.5.4.1 General

The $ICC_{A,B}$ values are mapped to an attraction measure matrix $\mathbf{T}$ with elements:

$$T_{A,B} = \begin{cases} \min\left(PasMax, \max\left(0, PasCurveSlope \cdot ICC_{A,B} + PasCurveShift\right)\right) & \text{,for} \quad A \neq B \\ \min\left(1, \max\left(0, 2.5 \cdot ICC_{A,B} - 1.2\right)\right) & \text{,for} \quad A = B \end{cases}$$

where *PasMax, PasCurveSlope, PasCurveShift* are derived from Table 168.

Table 168 — Phase attraction mapping curve parameters

phaseAlignStrength	PasMax	PasCurveSlope	PasCurveShift
0	0	0	0
1	0,071 4	0,170 1	-0,089 1
2	0,154 8	0,377 1	-0,189 6
3	0,25	0,625	-0,3
4	0,357 1	0,918 4	-0,418 4
5	0,476 2	1,262 3	-0,542 7
6	0,607 1	1,662 4	-0,670 7
7	0,75	2,125	-0,8
phaseAlignStrength shall be set to 3 if no other value has been signalled in the bitstream. If passiveDownmixFlag==1, then phaseAlignStrength shall be set to 0.			

An intermediate phase-aligning mixing matrix $\mathbf{M}_{\text{int}}$ is calculated. With

$P_{A,B} = T_{A,B} \cdot C_{y,A,B}$ and

$\mathbf{V} = \mathbf{M}_{\text{DMX}}\mathbf{P}$

the matrix elements are derived as:

$$M_{\text{int},B,A} = \frac{M_{\text{DMX},B,A}}{eps + \left|V_{B,A}\right|} V_{B,A}$$

The intermediate phase-aligning mixing matrix $\mathbf{M}_{\text{int}}$ is modified to avoid abrupt phase shifts, resulting in $\mathbf{M}_{\text{mod}}$. This is a recursive regularization process, running for each frame F, processing the frequency bands k in ascending order.

The regularization against phase shifts takes place in two stages: In the first stage, the regularization performs amplitude-weighted phase comparison against the previous frame, previous band, while also linking the phase-attracted channels. In the second stage, the regularization limits the update rate of the phase coefficients in comparison to the previous frame only.

Both regularization stages make use of a phase update limiting parameter, $\theta_{\text{diff},A}^{F,k}$, which is formulated as a function of an onset measure $o_A^{F,k}$ so that a low energy portion of a signal does not affect the phase processing after an onset:

$$o_A^{F,k} = \frac{C_{y,A,A}^{F,k}}{eps + C_{y,A,A}^{F,k} + C_{y,A,A}^{F-1,k}}$$

$$\theta_{\text{diff},A}^{F,k} = \max(0.15, 20.3 o_A^{F,k} - 19)\pi$$

10.3.5.4.2 Regularization stage 1

Stage 1 recursively takes into account comparison values $\mathbf{M}_{\text{cmp}}$ from the last frame index (F-1) as well as for the last processing band (k-1). $\mathbf{M}_{\text{cmp}}$ is derived from $\mathbf{M}_{\text{mod}}$ at the end of the regularization process. The first step of regularization stage 1 combines the comparison data across frequency and time as follows:

if (F=0)

$$\mathbf{M}_{\text{cmpFk}}^{F,k} = \begin{cases} 0 & \text{,for} \quad k = 1 \\ \mathbf{M}_{\text{cmp}}^{\text{F,k-1}} & \text{,for} \quad k > 1 \ \wedge \ k \neq 3 \\ \left(\mathbf{M}_{\text{cmp}}^{\text{F,k-1}}\right)^{*} & \text{,for} \quad k = 3 \end{cases}$$

else (i.e. for F>0)

$$\mathbf{M}_{\text{cmpFk}}^{F,k} = \begin{cases} \mathbf{M}_{\text{cmp}}^{F-1,k} & \text{,for} \quad k = 1 \\ \mathbf{M}_{\text{cmp}}^{F-1,k} + \mathbf{M}_{\text{cmp}}^{\text{F,k-1}} & \text{,for} \quad k > 1 \ \wedge \ k \neq 3 \\ \mathbf{M}_{\text{cmp}}^{F-1,k} + \left(\mathbf{M}_{\text{cmp}}^{\text{F,k-1}}\right)^{*} & \text{,for} \quad k = 3 \end{cases}$$

where the complex conjugate processing for the third band (k=3) accounts for the complex conjugate properties of the filterbank.

The frequency index k is omitted in the following since the inter-band dependency is now contained in the matrix $\mathbf{M}_{\text{cmpFk}}^{F,k}$. The phase change of the current unregularized phase-aligning matrix $\mathbf{M}_{\text{int}}^{F}$ relative to $\mathbf{M}_{\text{cmpFk}}^{F,k}$ is measured by amplitude weighting with $\sqrt{C_{y,A,A}^{F}}$ and comparison against $\mathbf{M}_{\text{cmpFk}}^{F}$, forming $\mathbf{M}_{\text{timeFreq}}^{F}$ with elements:

$$M_{\text{timeFreq},B,A}^{F} = M_{\text{cmpFk},B,A}^{F}\left(M_{\text{int},B,A}^{F}\right)^{*}\sqrt{C_{y,A,A}^{F}}$$

To also take into account the interdependent channels in the regularization, the relevant entries are intermixed with the attraction matrix $\mathbf{T}^{F}$:

$$\mathbf{M}_{\text{timeFreqChan}}^{F} = \mathbf{M}_{\text{timeFreq}}^{F}\mathbf{T}^{F}$$

The phase values of the elements of matrix $\mathbf{M}_{\text{timeFreqChans}}^{F}$ are:

$$\theta_{\text{timeFreqChan},B,A}^{F} = \arg\left(M_{\text{timeFreqChan},B,A}^{F}\right)$$

To avoid constant phase offsets, $\theta_{\text{timeFreqChan},B,A}^{F}$ is adjusted towards zero by $\theta_{\text{diff},A}^{F}$:

$$\theta_{\text{timeFreqChanStage1},B,A}^{F} = \text{sign}\left(\theta_{\text{timeFreqChan},B,A}^{F}\right)\max\left(0, abs(\theta_{\text{timeFreqChan},B,A}^{F}) - \theta_{\text{diff},A}^{F}\right)$$

10.3.5.4.3 Regularization stage 2

In stage 2 of the regularization another phase comparison parameter, only across time, is calculated:

$$\theta^{F}_{\text{time},B,A} = \arg\left(M^{F-1}_{\text{PA},B,A} \left(M^{F}_{\text{int},B,A} \right)^{*} \right)$$

The final regularization parameter is defined to be as close as possible to $\theta^{F}_{\text{timeFreqChanStage1},B,A}$, but not further than $\theta^{F}_{\text{diff},A}$ from $\theta^{F}_{\text{time},B,A}$. Let *unwrap()* be a function that maps any angular parameter to the corresponding angle in the interval $-\pi \ldots \pi$. The final phase parameter is calculated as:

$$\theta^{F}_{\text{update},B,A} = unwrap\left(\theta^{F}_{\text{timeFreqChanStage1},B,A} - \theta^{F}_{\text{time},B,A} \right),$$
$$\theta^{F}_{\text{mod},B,A} = \theta^{F}_{\text{time},B,A} + sign\left(\theta^{F,k}_{\text{update},B,A} \right) \max(0, abs(\theta^{F}_{\text{update},B,A}) - \theta^{F}_{\text{diff},A}),$$

and the modified, i.e. phase-regularized, mixing matrix elements are obtained as:

$$M^{F}_{\text{mod},B,A} = M^{F}_{\text{int},B,A} \cdot \exp\left(j \cdot \theta^{F}_{\text{mod},B,A} \right)$$

Finally, $\mathbf{M}_{\text{mcp}}$ is derived by amplitude weighting the regularized downmixing coefficients,

$$M^{F}_{\text{cmp,B,A}} = M^{F}_{\text{mod},B,A} \sqrt{C^{F}_{y,A,A}}$$

Note that $\mathbf{M}_{\text{mcp}}$ is used in the time and frequency recursive formulation of regularization stage 1.

10.3.5.4.4 Energy scaling

An energy scaling is applied to the mixing matrix to obtain the final phase-aligning mixing matrix $\mathbf{M}_{\text{PA}}$. With

$$\mathbf{M}_{Cy} = \mathbf{M}_{\text{mod}} \mathbf{C}_{y} \mathbf{M}^{\text{H}}_{\text{mod}}$$

where $(\cdot)^{\text{H}}$ denotes the conjugate transpose operator, and

$$S_{B} = \sqrt{\frac{\sum_{A=1}^{N_{in}} M_{\text{DMX},B,A} \cdot M_{\text{DMX},B,A} \cdot C_{y,A,A}}{\text{eps} + M_{Cy,B,B}}}$$

$$S_{\text{lim},B} = \min\left(S_{\max}, \max\left(S_{\min}, S_{B} \right) \right)$$

where the limits are defined as $S_{\max} = 10^{0.4}$ and $S_{\min} = 10^{-0.5}$, the final phase-aligning mixing matrix elements follow as:

$$\mathrm{M}_{PA,B,A} = (S_{\text{lim},B} \cdot AES + (1 - AES)) \cdot \mathrm{M}_{\text{mod},B,A}$$

where *AES* = (1 – **passiveDownmixFlag**).

10.3.5.5 Calculation of output data

The output signals for the current frame F are computed by linearly interpolating the mixing matrices from the previous frame to the current frame:

$$\mathbf{z}_{\mathrm{ch}}^{F,n} = \left(\left(\frac{n+1}{L_n} \mathbf{M}_{\mathrm{PA}}^{F} + \frac{L_n - n - 1}{L_n} \mathbf{M}_{\mathrm{PA}}^{F-1} \right) \left(\mathbf{y}_{\mathrm{in,\,ch}}^{F,n} \right)^{\mathrm{T}} \right)^{\mathrm{T}} \quad \text{for} \quad 0 \le n < L_n$$

Note that the input audio for the above mixing procedure is the first half of the analysis window.

10.3.5.6 F/T-transform (hybrid QMF synthesis)

Note that the processing steps described above have to be carried out for each hybrid QMF band k (recursively, for ascending k). In the following procedure the band index k is reintroduced, i.e. $\mathbf{z}_{\mathrm{ch}}^{F,n,k} = \mathbf{z}_{\mathrm{ch}}^{F,n}$ The hybrid QMF frequency domain output signal $\mathbf{z}_{\mathrm{ch}}^{F,n,k}$ is transformed to an N_{out}-channel time domain signal frame of length L time domain samples per output channel B, yielding the time domain output signal $\underset{\sim}{\mathbf{z}}_{\mathrm{ch}}^{F,\nu}$:

The hybrid synthesis:

$$\hat{\mathbf{z}}_{\mathrm{ch}}^{F,n,k} = \mathrm{HybridSynthesis}(\mathbf{z}_{\mathrm{ch}}^{F,n,k})$$

is carried out as defined in ISO/IEC 14496-3:2009, Figure 8.21, i.e. by summing the sub-subbands of the three lowest QMF subbands to obtain the three lowest QMF subbands of the 64band QMF representation. The subsequent QMF synthesis:

$$\underset{\sim}{\mathbf{z}}_{\mathrm{ch}}^{F,\nu} = \mathrm{QMFSynthesis}(\hat{\mathbf{z}}_{\mathrm{ch}}^{F,n,k})$$

shall be carried out as defined in ISO/IEC 14496-3:2009, 4.6.18.4.

11 Immersive loudspeaker rendering/format conversion

11.1 Description

For the 5.0 and 5.1 channel output layouts, immersive loudspeaker rendering is chosen to provide overhead sound images using surround channel loudspeakers. The immersive loudspeaker renderer is a downmixer that converts multichannel signals from transmitted channel configurations with N_{in} channels to desired reproduction format of either 5.1 or 5.0 system. It has a switching scheme between 3D rendering and 2D rendering using different elevation rendering for height input channels depending on the transmitted bitstream **rendering3DType** to provide overhead sound image properly. It is thus also called 'immersive format converter'. The system consists of two major building blocks:

— an initialization algorithm that takes into account static parameters like the input and output format;

— a signal adaptive downmixing process that operates in a subband domain with a switching scheme according to the transmitted flag **rendering3DType**.

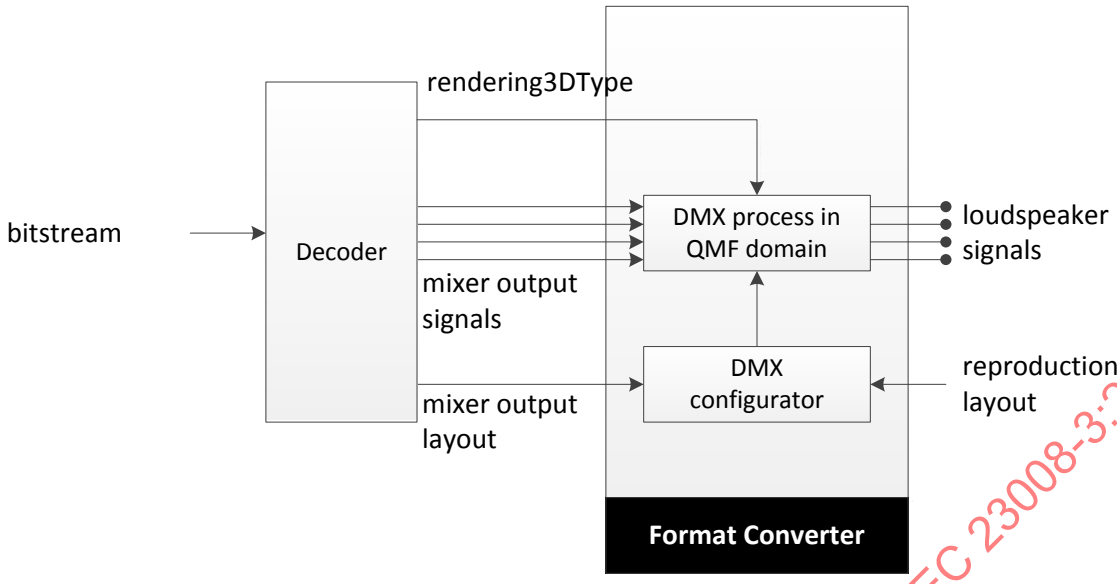


Figure 53 — Main building blocks of the immersive format converter

In the initialization phase the format converter automatically generates optimized downmixing parameters (like the downmixing matrix) for the given combination of input and output formats. It applies an algorithm that selects for each input loudspeaker the most appropriate mapping rule from a list of rules that has been designed to incorporate psychoacoustic considerations. Each rule describes the mapping from one input channel to one or several output loudspeaker channels.

Input channels are

— either mapped to a single output channel;

— or panned to two output channels;

— or (in case of the 'Voice of God' channel) distributed over a larger number of output channels;

— or (in case of the 'elevation rendering') panned to multiple output channels with different panning coefficients over frequency.

The optimal mapping for each input channel is selected depending on the list of output loudspeakers that are available in the desired output format. Each mapping defines downmix gains for the input channel under consideration as well as potentially also an equalizer that is applied to the input channel under consideration.

Output setups with non-standard loudspeaker positions can be signalled to the system by providing the azimuth and elevation deviations from a regular loudspeaker setup. Further, distance variations of the desired target loudspeaker positions are taken into account.

For each frame, a bit called rendering3DType is decoded by the decoder and passed to the immersive format converter. The rendering3DType indicates whether the sound scene is appropriate for the 3D rendering or 2D rendering over either 5.0 or 5.1 channel layout. For the height input channels, the immersive format converter uses the "spatial elevation rendering" for 3D rendering when the rendering3DType is TRUE and "timbral elevation rendering" for the 2D rendering when the rendering3DType is FALSE. For the non-height input channels, non-elevation rendering uses the same downmix coefficients regardless of the rendering3DType.

The actual downmixing of the audio signals is performed on a hybrid QMF subband representation of the signals. The algorithm makes use of two mechanisms to avoid signal deteriorations like comb-filtering, coloration, or modulation artifacts.

— Phase-alignment of the multichannel input signals: Correlated input signals that differ in phase are aligned prior to downmixing them. The alignment process makes use of an attraction measure to only align the relevant channels for the relevant time-frequency tiles and to avoid modifications to other parts of the input signal. The alignment is further regularized to avoid artifacts due to rapid changes of the alignment phase modification terms. The phase-alignment improves the output signal quality by avoiding narrow spectral notches due to out-of-phase signal cancellations that could not be compensated for by energy normalization because of a limited frequency resolution. It further reduces the need of boosting signals in the energy preserving normalization, thus minimizes modulation artifacts. For the downmix cases with the 'elevation rendering', the frequency components in the range 2,8 kHz ~ 10 kHz are not aligned in order to provide accurate synchronization of the rendered output signal. This stage is explained in subclause 11.4.2.5.

— Normalization of the downmix process to preserve the input energies (except for the desired energy scaling that may be inherent in the downmix matrix).

11.2 Syntax

The FormatConverterFrame defines the proper rendering type for the immersive format converter.

Table 169 — Syntax of FormatConverterFrame()

Syntax	No. of bits	Mnemonic

FormatConverterFrame()
{
rendering3DType;
}
	1	**uimsbf**

The flag **rendering3DType** is created at the encoder based on the audio scene. When the audio scene is wideband and highly decorrelated at a frame, the flag **rendering3DType** becomes false and rendering is achieved by the secondary downmix matrix $\mathbf{M}_{DMX2}$. In all other cases, the flag becomes true and rendering is achieved by the primary downmix matrix $\mathbf{M}_{DMX}$, which provides elevated sound images.

11.3 Definitions

11.3.1 General remarks

Audio signals that are fed into the format converter are referred to as *input signals* in the following. Audio signals that are the result of the format conversion process are referred to as *output signals*. Note that the audio input signals of the format converter are audio output signals from the core decoder.

Vectors and matrices are denoted by bold-faced symbols. Vector elements or matrix elements are denoted as italic variables supplemented by indices indicating the row/column of the vector/matrix element in the vector/matrix, e.g. $[y_1 \cdots y_a \cdots y_N] = \mathbf{y}$ denotes a vector and its elements. Similarly, $M_{a,b}$ denotes the element in the ath row and bth column of a matrix **M**.

11.3.2 Variable definitions

N_{in}	Number of channels in the input channel configuration.
N_{out}	Number of channels in the output channel configuration.
$\mathbf{M}_{DMX}$	Primary downmix matrix containing real-valued non-negative downmix coefficients (downmix gains) for the non-elevation rendering and spatial elevation rendering for 3D rendering over 2D layout. $\mathbf{M}_{DMX}$ is basically of dimension (N_{out} × N_{in}) but possibly increased to (N_{out} × (N_{in} +5)) depending on the input and output layouts. See how the dimension is changed in subclause 11.4.2.3.
$\mathbf{M}_{DMX2}$	Secondary downmix matrix containing real-valued non-negative downmix coefficients (downmix gains) for the non-elevation rendering and timbral elevation rendering for 2D rendering over 2D layout. The downmix coefficients for the horizontal input channels are identical to those in $\mathbf{M}_{DMX}$. The $\mathbf{M}_{DMX}$ is dimension of (N_{out} x N_{in}) but possibly increased to (N_{out} x (N_{in}+5)) depending on the input and output layouts. See how the dimension is changed in subclause 11.4.2.3.
$\mathbf{G}_{EQ}$	Matrix consisting of gain values per processing band determining frequency responses of equalizing filters for all rendering mapping. $\mathbf{G}_{EQ,1\sim5}$ are used for the non-elevation rendering and timbral elevation rendering, $\mathbf{G}_{EQ,7\sim14}$ are used for spatial elevation rendering, $\mathbf{G}_{EQ,15\sim20}$ are used for spatial coloration filter, and $\mathbf{G}_{EQ,21\sim}$ are used for modified EQ for randomized setup in subclause 11.4.1.6.5 and coloration filter in subclause 11.4.1.6.7.6.
$\mathbf{I}_{EQ}$	Vector signalling which equalizer filters to apply to the input channels (if any).
L	Frame length measured in the time domain audio samples.
ν	Time domain sample index.
n	QMF time slot index (= subband sample index).
L_n	Frame length measured in QMF slots.
F	Frame index (frame number).
K	Number of hybrid QMF frequency bands, $K = 71$.
k	QMF band index (1..64) or hybrid QMF band index (1.. K).
A, B	Channel indices.
eps	Numerical constant, $\text{eps} = 10^{-35}$.

rendering3DType	Flag from the bitstream identifying the rendering type for the elevation rendering. True for the general audio scene and false for the highly decorrelated wideband scene (e.g. applause). Accordingly, the primary downmix matrix $\mathbf{M}_{DMX}$ is chosen when the **rendering3DType** is TRUE and the secondary downmnix matrix $\mathbf{M}_{DMX2}$ is chosen when the **rendering3DType** is FALSE in the immersive format converter.
i_{in}	Label of the input channel to be rendered by immersive format converter (e.g. CH_U_000).
$G_{vH0,1\sim6}$ (i_{in})	Spatial elevation panning coefficients for the input channel (i_{in}) for the 2.8~10kHz in order to provide overhead image. Note that the coefficient is always normalized to preserve the input power after mixing. The number index represents output channels as shown in Table 178.
$G_{vL0,1\sim6}$ (i_{in})	Spatial elevation panning coefficients for the input channel (i_{in}) for below 2.8 kHz and above 10kHz. Note that the coefficient is always normalized to reserve the input power after mixing. The number index represents output channels as shown in Table 178.
COLOR_A_B	Tone coloration filter to the output at the azimuth of ±B degrees from the input at the azimuth of ±A degrees based on the ratio between HRTF at A and HRTF at B. It is determined by a frequency dependent dynamic cue which represents loudspeaker-to-listener orientation.

11.4 Processing

11.4.1 Initialization of the format converter

11.4.1.1 General description of the initialization

The initialization of the format converter is carried out before audio samples delivered by the core decoder are processed.

The initialization takes the following input parameters into account.

- The sampling rate of the audio data to process.
- The channel configuration of the audio data to process with the format converter (number and geometric positions of input channels).
- The channel configuration of the desired output format (number and geometric positions of output channels).
- Optional: Parameters signaling the deviation of the output loudspeaker positions from a standard loudspeaker setup (random setup functionality).

It returns:

- The primary frequency dependent downmix matrix $\mathbf{M}_{DMX}$ that is applied in the audio signal processing of the format converter when the rendering3DType is TRUE. Note that the $\mathbf{M}_{DMX}$ is independent variable in the format converter and shall not be taken into account in the core decoding process in subclause 5.5.3.1.2;

— The secondary frequency dependent downmix matrix $\mathbf{M}_{DMX2}$ that is applied in the audio signal processing of the format converter when the rendering3DType is FALSE. The $\mathbf{M}_{DMX2}$ is identical to $\mathbf{M}_{DMX}$ if there is no 'height' input channel or 'spatial elevation rendering' is not possible.

The input parameters to the initialization algorithm are listed in Table 170.

Table 170 — Format converter initialization input parameters

	Input format: number of channels and nominal channel setup geometry
	Output format: number of channels and nominal channel setup geometry
f_s	Sampling frequency in Hertz.
$r_{azi,A}$	For each output channel A, an azimuth angle is specified, determining the deviation from the standard format loudspeaker azimuth.
$R_{ele,A}$	For each output channel A, an elevation angle is specified, determining the deviation from the standard format loudspeaker elevation.

Table 171 lists the output parameters that are derived during the initialization of the format converter.

Table 171 — Format converter initialization output parameters

$\mathbf{M}_{DMX}$	Primary Downmix matrix [linear gains] for spatial elevation rendering (for rendering3DType == 1)
$\mathbf{M}_{DMX2}$	Secondary Downmix matrix [linear gains] for timbral elevation rendering (for rendering3DType == 0)

Note that the $\mathbf{M}_{DMX1}$ and $\mathbf{M}_{DMX2}$ include the same input-output downmix matrix for non-elevation rendering input channels.

11.4.1.2 Assignment of format converter channel labels to input/output format channels

The format converter initialization is based on a system of rules that are defined in terms of *format converter channel labels*, see Table 180. To allow the application of the initialization rules, the channel labels have to be assigned to the channels of the input and output formats. Each format converter channel label is associated with a segment of the surface of the unit sphere, as defined in Table 180. The segments are designed to be non-overlapping.

The assignment of channel labels to channels is done by geometrically matching the segments to the position data associated with the channels of the input and output formats. The azimuth and elevation angles in degrees of the position data associated with the channels shall be rounded towards the nearest integer number before performing the channel label assignment. Note that the *nominal* channel positions shall be applied in the following matching to channel label sectors, i.e. the azimuth and elevation angles *without* taking into account potential angle deviations signalled in $r_{azi,A}$ and/or $r_{ele,A}$.

For each channel that is not an LFE (low frequency enhancement) channel:

If the nominal position of the current channel, defined by its azimuth angle and elevation angle, is within or on the border of one of the segments defined in Table 180 then:

— Assign the corresponding channel label (e.g. CH_M_L030) associated with the matching segment.

— Add the angle differences between the nominal position of the current channel and the nominal position associated with the matching segment (i.e. the angles in the second and third column of Table 180) to the angle deviations stored in $r_{azi,A}$ and $r_{ele,A}$.

Else (i.e. no matching sector found), then:

— Assign the CH_EMPTY label.

If an input or output format contains exactly one LFE channel, then the label CH_LFE1 shall be assigned to this channel.

If an input or output format contains exactly two LFE channels, then the labels CH_LFE1 and CH_LFE2 shall be assigned to the two LFE channels in the order that minimizes the maximum azimuth distance from the channels to the assigned CH_LFE1 and CH_LFE2 nominal azimuth positions.

If an input or output format contains more than 2 LFE channels, then those 2 LFE channels out of the considered setup shall be selected that minimize the maximum azimuth distance to the CH_LFE1 and CH_LFE2 nominal azimuth positions. The labels CH_LFE1 and CH_LFE2 shall be assigned as in the case of two LFE channels. The remaining LFE channels shall not be considered further in the calculation of downmix coefficients, i.e. the corresponding lines/columns of the downmix matrix shall remain filled with zeros.

11.4.1.3 Handling for unknown input channels

If the label CH_EMPTY is assigned to an input channel, this channel shall be considered unknown to the rules-based initialization and the downmix coefficients for mapping this input channel to the output channels shall be derived as specified in subclause 10.3.4.6.7.

11.4.1.4 Handling for unknown output formats

If the output format contains at least one channel with the label CH_EMPTY assigned to it, or if at least one channel label is assigned to more than one channel of the output format, the output format shall be considered unknown and the derivation of the downmixing coefficients shall be carried out as specified in subclause 11.4.1.6.9. The rules-based derivation of downmix coefficients shall not be applied for unknown output formats.

11.4.1.5 Handling of deviations from standard loudspeaker positions

If the below conditions are not met, the rules-based initialization is considered to have failed, the output format shall be considered to be unknown, and the downmixing gains shall be obtained as defined in subclause 11.4.1.6.9.

The absolute values of $r_{azi,A}$ and $r_{ele,A}$ shall not exceed 35 and 55 degrees, respectively. The minimum angle between any loudspeaker pair (without LFE channels) shall not be smaller than 15 degrees.

The values of $r_{azi,A}$ shall be such that the ordering by azimuth angles of the horizontal loudspeakers does not change. Likewise, the ordering of the height and low loudspeakers shall not change.

The values of $r_{ele,A}$ shall be such that the ordering by elevation angles of loudspeakers which are (approximately) above/below each other does not change. To verify this, the following procedure is applied:

For each row of Table 185 which contains two or three channels of the output format, do:

— Order the channels by elevation without randomization;

— Order the channels by elevation with considering randomization;

— If the two orderings differ, return an initialization error.

If the below conditions are not met, converter initialization is considered to have failed, and an error shall be returned.

11.4.1.6 Rules-based initialization algorithm

11.4.1.6.1 General

The rules-based initialization algorithm is defined in the following subclauses. The algorithm shall not be applied if the output format is considered unknown as defined in the previous subclause. For clarity the following description makes use of intermediate parameters listed in Table 172 but an implementation may omit the explicit use of these intermediate parameters.

Table 172 — Format converter initialization intermediate parameters

$\mathbf{S}$, $\mathbf{S}^P$, $\mathbf{S}^S$	Vector of converter source channels [input channel indices]
$\mathbf{D}$, $\mathbf{D}^P$, $\mathbf{D}^S$	Vector of converter destination channels [output channel indices]
$\mathbf{G}$, $\mathbf{G}^P$, $\mathbf{G}^S$	Vector of converter gains [linear]
$\mathbf{E}$, $\mathbf{E}^P$, $\mathbf{E}^S$	Vector of converter EQ indices
$\mathbf{G}_{EQ}$	Matrix containing equalizer gain values for all EQ indices and frequency bands

The superscript S/P is the discriminator for the elevation rendering type. Those with designated superscript P are initialized to be used for the 'spatial elevation rendering' and used to create the primary downmix matrix $\mathbf{M}_{DMX}$, those with designated superscript S are for the 'timbral elevation rendering' and used to create the secondary downmix matrix, and those without superscript are for the non-elevation rendering and used to create both the primary and secondary downmix matrixes.

The intermediate parameters describe the downmixing parameters according to the mapping, i.e. as sets of parameters S_i, D_i, G_i, E_i, per mapping i.

The format converter initialization output parameters are derived as described in Figure 54.

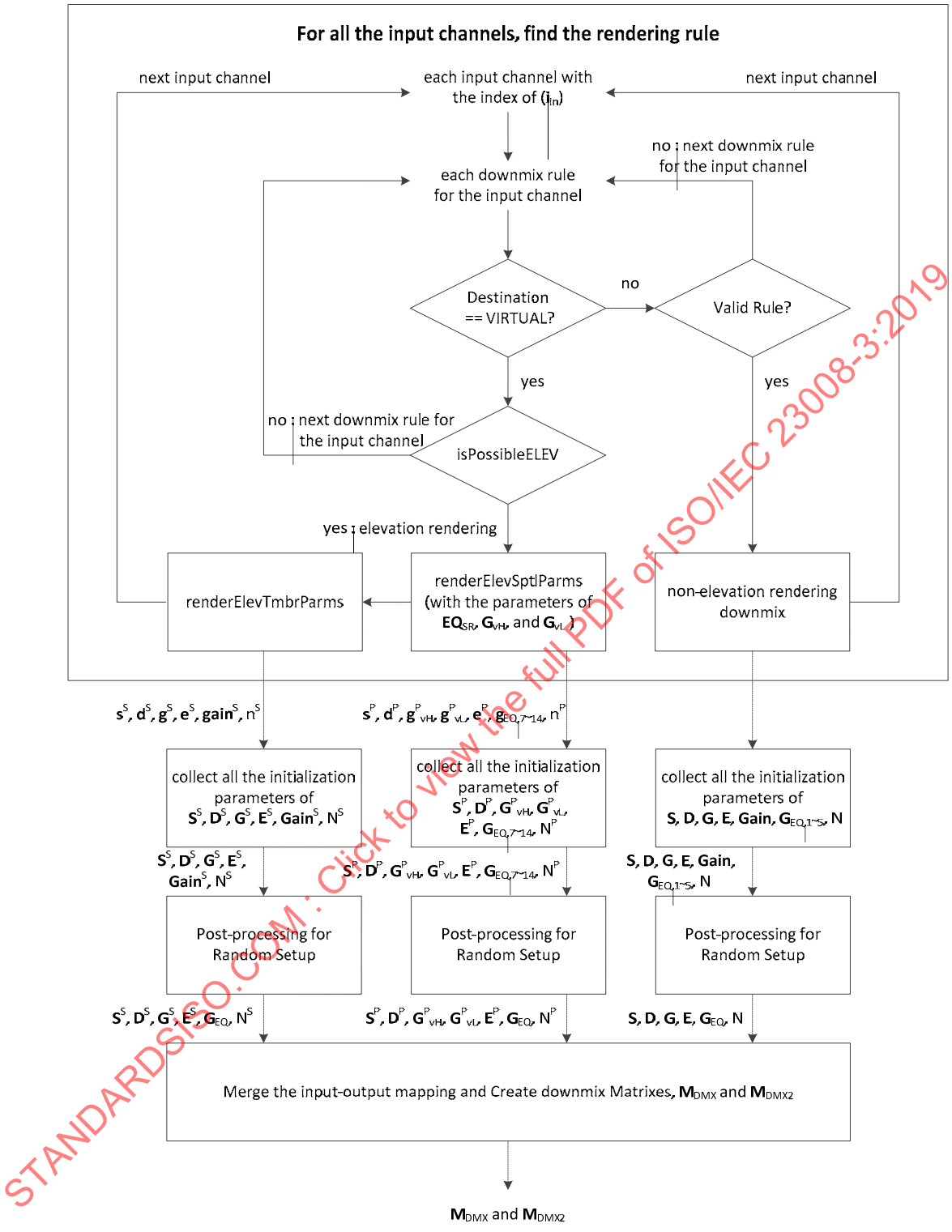


Figure 54 — Rule-based downmix initialization flow

With following steps for each channel:

— If the destination of the downmix rule for the input channel (i_{in}) is VIRTUAL, isPossibleElev defines whether the input channel should be rendered by the elevation rendering defined in subclause 11.4.1.6.7.

- If isPossibleElev returns TRUE,
  - A set of parameters of $\mathbf{s}^{P}$, $\mathbf{d}^{P}$, $\mathbf{g}^{P}{}_{H}$, $\mathbf{g}^{P}{}_{L}$, $\mathbf{e}^{P}$, $\mathbf{G}_{EQ,eq(i_{in})}$, and n^{P} and another set of parameters of $\mathbf{s}^{S}$, $\mathbf{d}^{S}$, $\mathbf{g}^{S}$, $\mathbf{e}^{S}$, $\mathbf{gain}^{S}$, and n^{S} are initialized by the renderElevParms and renderTmbrParms, respectively. The parameters of n^{P} and n^{S} indicate the number of output loudspeakers required for the input channel (i_{in}), $\mathbf{s}^{P}$, $\mathbf{d}^{P}$, $\mathbf{g}^{P}{}_{H}$, $\mathbf{g}^{P}{}_{L}$, and $\mathbf{e}^{P}$ are n^{P} column vectors, $\mathbf{s}^{S}$, $\mathbf{d}^{S}$, $\mathbf{g}^{S}$, $\mathbf{e}^{S}$, and $\mathbf{gain}^{S}$ are n^{S} column vectors, $\mathbf{G}_{EQ,eq(i_{in})}$ is a 71 row vector representing the EQ coefficients for the 71 bands of the input channel (i_{in}).
  - The initialization parameters are collected among the group of parameters, each primary or secondary, and the downmix rules for next input channel are investigated until all the input channel mapping is found.
- If isPossibleElev returns FALSE, the current downmix rule shall be ignored and the next downmix rule shall be investigated.

— If the destination is not VIRTUAL, the downmix rule shall be investigated to determine whether the rule is valid; by checking whether the output layout includes all of the channels in the destination column.
- If the downmix rule is valid,
  - A set of parameters shall be initialized by the non-elevation rendering initialization and added directly to the **S**, **D**, **G**, **E**, and **Gain** as specified in subclause 11.4.1.6.3 and to the $\mathbf{G}_{EQ}$ as specified in subclause 11.4.1.6.4.
  - The downmix rules for the next input channel shall be investigated until all of the input channel mappings are found.
- If the downmix rule is invalid, the current downmix rule shall be ignored and the next downmix rule shall be investigated.

— After collecting all of the initialization parameters for the 'spatial elevation rendering', 'timbral elevation rendering' and 'non-elevation rendering' post-processing for random setup shall be applied.

— Create the primary downmix matrix, combining the 'spatial elevation rendering' and 'non-elevation rendering' parameters, and the secondary downmix matrix, combining the 'timbral elevation rendering' and 'non-elevation rendering' parameters.

11.4.1.6.2 Random setups pre-processing

Random output loudspeaker setups, i.e. output setups that contain loudspeakers at positions deviating from the positions defined for the desired output format, are signalled by specifying the loudspeaker position deviation angles as input parameters $r_{azi,A}$ and $r_{ele,A}$. The angle deviations are taken into account as a pre-processing step.

Modify the channels' azimuth and elevation angles according to Table 180 by adding $r_{azi,A}$ and $r_{ele,A}$ to the corresponding channels' azimuth and elevation angles.

11.4.1.6.3 Derivation of input channel/output channel mapping parameters

The parameters vectors **S**, **D**, **G**, **E** define the mapping of input channels to output channels. For each mapping *i* from an input channel to an output channel with non-zero downmix gain they define the downmix gain as well as an equalizer index that indicates which equalizer curve has to be applied to the input channel under consideration in mapping *i*.

The elements of the parameter vectors **S**, **D**, **G**, **E** are derived by the following algorithm:

— Initialize the mapping counter i: i=1, i^P=1, i^S=1;

— Initialize the EQ counter e: e=21 (EQ slots for e from 1 to 5 are occupied by subclause 11.4.1.6.4, that for e from 7 to 14 are occupied by Table 177, and that for e from 15 to 20 are occupied by Table 174. The EQ counter e will be shared in subclauses 11.4.1.6.5 and 11.4.1.6.7.6 in an incremental way).

For each input channel, ignoring channels with label CH_EMPTY assigned to them:

If the input channel also exists in the output format (e.g. input channel under consideration is CH_M_R030 and channel CH_M_R030 exists in the output format), then:

— S_i = index of source channel in input
EXAMPLE channel CH_M_R030 in ChannelConfiguration 6 is at second place according to Table 181, i.e. has index 2 in this format.
— D_i = index of same channel in output
— G_i = 1,0
— E_i = 0
— $i = i + 1$

Else (i.e. if the input channel does not exist in the output format)

— search the first entry of this channel in the **Source** column of Table 182, for which the channels in the corresponding row of the **Destination** column exist. The **VIRTUAL** destination shall be considered valid if the **isPossibleElev** returns **TRUE**, which indicates the output format contains required channels for the elevation rendering of the input channel. The **isPossibleElev**, **renderElevSptlParms,** and **renderElevTmbrParms** are defined in subclause 11.4.1.6.7. The ALL_U destination shall be considered valid (i.e. the relevant output channels exist) if the output format contains at least one "CH_U_" channel. The ALL_M destination shall be considered valid (i.e. the relevant output channels exist) if the output format contains at least one "CH_M_" channel. If for no entry in Table 182 corresponding to the input channel the channels in the **Destination** column exist, the rules-based initialization shall terminate and the downmix gains shall be derived according to subclause 11.4.1.6.9.

If **Destination** column contains **VIRTUAL**, then:

[$\mathbf{s}^P$, $\mathbf{d}^P$, $\mathbf{g}^P_H$, $\mathbf{g}^P_L$, $\mathbf{e}^P$, n^P]= **renderElevSptlParms** (specified in subclause 11.4.1.6.7.4)

for n = 1 to n^P

$m = i^P$

$S^P_m = s^P_n(\mathrm{i_{in}})$

$D^P_m = d^P_n(\mathrm{i_{in}})$

$G^P_{mH} = g^P_{nH}(\mathrm{i_{in}})$

$G^P_{mL} = g^P_{nH}(\mathrm{i_{in}})$

$E^{P}_{m} = e^{P}_{n}(i_{in})$

$C^{P}_{mi} = (i_{in})$

C^{P}_{mo} = label of the output channel $d^{P}_{n}(i_{in})$

$i^{P} = i^{P} + 1$

[$\mathbf{s}^{S}$, $\mathbf{d}^{S}$, $\mathbf{g}^{S}$, $\mathbf{e}^{S}$, **gain**, n^{S}]= **renderElevSptlParms** (described in subclause 11.4.1.6.7.5)

for n = 1 to n^{S}

$m = i^{S}$

$S^{S}_{m} = s^{S}_{n}(i_{in})$

$D^{S}_{m} = d^{S}_{n}(i_{in})$

$G^{S}_{m} = g^{S}_{n}(i_{in})$

$E^{S}_{m} = e^{S}_{n}(i_{in})$

$C^{S}_{mi} = (i_{in})$

C^{S}_{mo} = label of the output channel $d^{P}_{n}(i_{in})$

$Gain^{S}_{m} = gain_{n}$

$i^{S} = i^{S} + 1$

where i_{in} js the input channel label.

Else, if **Destination** column contains ALL_U, then:

For each output channel x with “CH_U_” in its name, do:

- S_i = index of source channel in input
- D_i = index of channel x in output
- G_i = (value of Gain column) / sqrt(number of “CH_U_” output channels)
- E_i = value of EQ column
- $Gain_i$ = (value of Gain column)
- $i = i + 1$

Else if **Destination** column contains ALL_M, then:

For each output channel x with “CH_M_” in its name, do:

- S_i = index of source channel in input
- D_i = index of channel x in output
- G_i = (value of Gain column) / sqrt(number of “CH_M_” output channels)
- E_i = value of EQ column
- $Gain_i$ = (value of Gain column)

— $i = i + 1$

Else If there is one channel in the **Destination** column, then:

— S_i = index of source channel in input
— D_i = index of destination channel in output
— G_i = value of Gain column
— E_i = value of EQ column
— $Gain_i$ = (value of Gain column)
— $i = i + 1$

Else (two channels in **Destination** column):

— S_i = index of source channel in input
— D_i = index of first destination channel in output
— G_i = (value of Gain column) * g_1
— E_i = value of EQ column
— $Gain_i$ = (value of Gain column)
— $i = i + 1$

— $S_i = S_{i-1}$
— D_i = index of second destination channel in output
— G_i = (value of Gain column) * g_2
— $E_i = E_{i-1}$
— $Gain_i$ = (value of Gain column)
— $i = i + 1$

The gains g_1 and g_2 are computed by applying tangent law amplitude panning in the following way.

— Unwrap source destination channel azimuth angles to be positive.
— The azimuth angles of the destination channels are α_1 and α_2 (see Table 180).
— The azimuth angle of the source channel (= panning target) is α_{src}.
— $\alpha_0 = \frac{|\alpha_1 - \alpha_2|}{2}$
— $\alpha_{center} = \frac{\alpha_1 + \alpha_2}{2}$
— $\alpha = (\alpha_{center} - \alpha_{src}) \cdot \mathrm{sgn}(\alpha_2 - \alpha_1)$
— $g_1 = \frac{g}{\sqrt{1+g^2}}, \quad g_2 = \frac{1}{\sqrt{1+g^2}} \quad \text{with} \quad g = \frac{\tan\alpha_0 - \tan\alpha + 10^{-10}}{\tan\alpha_0 + \tan\alpha + 10^{-10}}$

11.4.1.6.4 Derivation of equalizer gains $\mathbf{G}_{EQ}$

$\mathbf{G}_{EQ}$ comprises a set of gain values for each frequency band k and equalizer index e. The 5 predefined equalizers are combinations of different peak filters across different values of e for $1 \le e \le 5$. Each equalizer is a serial cascade of one or more peak filters and a gain:

$$G_{\mathrm{EQ},e}^{k} = 10^{\frac{g}{20}} \prod_{n=1}^{N} \mathrm{peak}\left(band(k)\frac{f_s}{2}, P_{f,n}, P_{Q,n}, P_{g,n} \right)$$

where *band(k)* is the normalized centre frequency of frequency band k, specified in Table 183, f_s is the sampling frequency, and function *peak()* is for negative G:

$$\mathrm{peak}\left(b, f, Q, \mathrm{G}\right) = \sqrt{\frac{b^4 + \left(\frac{1}{Q^2} - 2\right) f^2 b^2 + f^4}{b^4 + \left(\frac{10^{\frac{-G}{10}}}{Q^2} - 2\right) f^2 b^2 + f^4}}$$

and otherwise:

$$\mathrm{peak}\left(b, f, Q, \mathrm{G}\right) = \sqrt{\frac{b^4 + \left(\frac{10^{\frac{G}{10}}}{Q^2} - 2\right) f^2 b^2 + f^4}{b^4 + \left(\frac{1}{Q^2} - 2\right) f^2 b^2 + f^4}}$$

The parameters for the equalizers are specified in Table 184.

11.4.1.6.5 Post-processing for random setups

Once the output parameters are computed, they are modified according to the specific random azimuth and elevations angles. This step only has to be carried out, if not all $r_{ele,A}$ are zero. The post-processing algorithm for non-elevation rendering is defined as follows.

For each element i in D_i do:

if the output channel with index D_i is a horizontal channel by definition (i.e. output channel label contains the label '_M_'), **and**

if this output channel is now a height channel (elevation in range 0..60 degrees), **and**

if input channel with index S_i is a height channel (i.e. label contains '_U_'), **then**

- h = min(elevation of randomized output channel, 35)/35
- $G_{comp} = h \cdot \frac{1}{Gain_i} + (1 - h)$
- Apply compensation gain to DMX gain: $G_i = G_i \cdot G_{\mathrm{comp}}$
- Define new equalizer $\mathbf{G}_{EQ,e}$ with the index e, where $G_{\mathrm{EQ},e}^{k} = h + (1-h) \cdot G_{\mathrm{EQ},E_i}^{k}$

— $E_i = e$
— $e = e + 1$

else if input channel with index S_i is a horizontal channel (label contains '_M_')

— h = min(elevation of randomized output channel, 35)/35
— Define new equalizer $\mathbf{G}_{\mathrm{EQ},e}$ with the index e,
where $G_{EQ,e}^{k} = \max\left(0.6310,\ \min\left(1.5849, \left(h \cdot IEQ(D_i)^k + (1-h)\right) \cdot G_{EQ,E_i}^{k}\right)\right)$
and $IEQ(D_i)^k$ is defined in Table 173
— $E_i = e$
— $e = e + 1$

Table 173 — Inverse spatial elevation filter

Azimuth of D_i	**Front centre (-15 15)**	**Front (-90 -15) or (15 90)**	**Side/Rear [-180 -90] or [90 180]**
$IEQ(D_i)^k$	$\frac{1}{G_{EQ,9}^{k}}$	$\frac{1}{G_{EQ,7}^{k}}$	$\frac{1}{G_{EQ,8}^{k}}$
NOTE $\mathrm{EQ}_{0,\mathrm{lin}}$ is defined in Table 177.			

Explanation of the post-processing steps defined above:

h is a normalized elevation parameter indicating the elevation of a nominally horizontal output channel ('_M_') due to a random setup elevation offset $r_{\mathrm{ele},A}$. For zero elevation offset h=0 follows and effectively no post-processing is applied.

The rule table (Table 182) in general applies a gain of the value in the gain column when mapping an upper input channel ('_U_' in channel label) to one or several horizontal output channels ('_M_' in channel label(s)). In case the output channel gets elevated due to a random setup elevation offset $r_{\mathrm{ele},A}$, the gain is partially ($0<h<1$) or fully ($h = 1$) compensated for. Similarly the equalizer definitions fade towards a flat EQ-curve ($G_{\mathrm{EQ},e}^{k} = const. = 1.0$) for h approaching $h = 1$.

In the case that a horizontal input channel is mapped to an output channel that is elevated, due to a random setup elevation offset $r_{\mathrm{ele},A}$, the equalizer G_{EQ,E_i}^{k} is fully applied and $IEQ(D_i)^k$, an inverse form of spatial elevation filter defined in Table 173, is partially ($0<h<1$) or fully (h=1) applied. As the spatial elevation filter is designed to provide the tone color of overhead loudspeakers on horizontal loudspeakers, the inverse of the spatial elevation filter is used to provide the tone color of horizontal loudspeakers on overhead loudspeakers. The modified EQ is thresholded within the level of 4 dB, [0.6310, 1.5849].

11.4.1.6.6 Spatial coloration filter for the horizontal input channels

When a horizontal input channel at side or rear is panned by two output loudspeakers, e.g. CH_M_L090 is panned by CH_M_L030 and CH_M_L110, the tone color changes. In order to avoid such a change in tone color, a set of horizontal coloration filters of $\mathbf{G}_{\mathrm{EQ},15\sim20}$ are defined as in Table 174. Here, the filter name COLOR_A_B means the coloration filter for the output at the azimuth of ±B from the input at the azimuth of ±A. The filtering algorithm is:

For each element i in **S** do:

A : the magnitude of the azimuth of the input channel with index S_i

B : the magnitude of the azimuth of the output channel with index D_i

If both S_i and D_i are horizontal channels and there exists a filter COLOR_A_B_ in Table 174,

If the output channel has no deviation in azimuth and elevation, ($r_{azi,D_i} = 0$ and r_{ele,D_i}=0)

If A==60 && B == 30 E_i = 15 (COLOR_60_30)

Elseif A==90 && B == 30 E_i = 16 (COLOR_90_30)

Elseif A==60 && B == 110 E_i = 17 (COLOR_60_110)

Elseif A==90 && B == 110 E_i = 18 (COLOR_90_110)

Elseif A==135 && B == 110 E_i = 19 (COLOR_135_110)

Elseif A==180 && B == 110 E_i = 20 (COLOR_180_110)

Table 174 — Spatial coloration filters for horizontal channels

Hybrid QMF band	COLOR_180_110 $G_{EQ,20}$	COLOR_090_030 $G_{EQ,16}$	COLOR_060_110 $G_{EQ,17}$	COLOR_135_110 $G_{EQ,19}$	COLOR_090_110 $G_{EQ,18}$	COLOR_060_030 $G_{EQ,15}$
0	1.2207277	0.82575887	1.0478644	1.0402648	0.9383601	0.92212301
1	1.2207277	0.82575887	1.0478644	1.0402648	0.9383601	0.92212301
2	1.2207277	0.82575887	1.0478644	1.0402648	0.9383601	0.92212301
3	1.2207277	0.82575887	1.0478644	1.0402648	0.9383601	0.92212301
4	0.79365081	1.1754434	1.01643	0.9288547	1.0755285	1.1108547
5	0.79365081	1.1754434	1.01643	0.9288547	1.0755285	1.1108547
6	0.79365081	1.1754434	1.01643	0.9288547	1.0755285	1.1108547
7	0.79365081	1.1585163	1.0149473	0.90169001	0.98411781	1.1948091
8	0.79365081	1.1585163	1.0149473	0.90169001	0.98411781	1.1948091
9	0.79365081	1.26	0.92721885	1.0003462	1.0241666	1.26
10	0.79365081	1.208599	0.81262708	0.90693218	0.87535268	1.1219938
11	0.79365081	0.97740591	1.0512857	1.0171721	0.97889137	1.0496904
12	0.85933852	0.84453988	1.26	0.97543412	1.2534994	0.90018034
13	0.90390807	0.94619727	1.26	0.91973215	1.26	1.0997185
14	0.88763344	1.0500977	1.26	0.86256438	1.26	1.1820639
15	0.80779493	1.0856131	1.26	0.79612881	1.26	1.26
16	0.80421317	0.99716073	1.26	0.79365081	1.26	1.1002132
17	0.85141277	0.89153987	1.26	0.79365081	1.26	0.98295653
18	0.83064753	0.94386107	1.26	0.79365081	1.26	1.0519516
19	0.81076753	0.99713677	1.26	0.79365081	1.26	1.1632811
20	0.79365081	1.1471872	1.26	0.79365081	1.26	1.26

Hybrid QMF band	COLOR_180_110 $G_{EQ.20}$	COLOR_090_030 $G_{EQ.16}$	COLOR_060_110 $G_{EQ.17}$	COLOR_135_110 $G_{EQ.19}$	COLOR_090_110 $G_{EQ.18}$	COLOR_060_030 $G_{EQ.15}$
21	0.79365081	1.26	1.26	0.79365081	1.26	1.26
22	0.79365081	1.26	1.26	0.79365081	1.26	1.26
23	0.79365081	1.26	1.26	0.79365081	1.26	1.26
24	0.79365081	1.26	1.1467814	0.79365081	1.26	1.26
25	0.79365081	1.26	1.0131826	0.79365081	1.233322	1.26
26	0.79365081	1.26	0.90711004	0.79365081	1.1852934	1.26
27	0.79365081	1.26	0.81770629	0.79365081	1.0953486	1.26
28	0.79365081	1.26	0.79365081	0.79365081	1.0236902	1.26
29	0.79365081	1.26	0.79365081	0.79365081	0.98883402	1.26
30	0.79365081	1.26	0.79365081	0.79365081	0.92739761	1.26
31	0.79365081	1.26	0.79365081	0.82225895	0.9255684	1.26
32	0.79365081	1.26	0.82704854	0.84583306	0.97202152	1.26
33	0.79365081	1.26	1.0166014	0.79365081	1.030863	1.26
34	0.79365081	1.26	1.2003248	0.79365081	1.1947373	1.26
35	0.79365081	1.26	1.2016704	0.79365081	1.26	1.26
36	0.79365081	1.26	1.0472132	0.79365081	1.26	1.26
37	0.79365081	1.26	0.92362517	0.79365081	1.218715	1.26
38	0.79365081	1.26	0.8078649	0.80314553	1.2225702	1.26
39	0.79365081	1.26	0.79365081	0.825122	1.1719567	1.1482208
40	0.79365081	1.26	0.79365081	0.82577235	1.0976481	0.95546353
41	0.79365081	1.26	0.79365081	0.82158899	1.0299088	0.81616014
42	0.79365081	1.26	0.79365081	0.81514698	0.98753756	0.79365081
43	0.79365081	1.1031394	0.79365081	0.84100527	0.99518955	0.79365081
44	0.79365081	0.95639628	0.79365081	0.89222193	1.0115879	0.79365081
45	0.79365081	0.79763001	0.79365081	0.88660526	0.97977269	0.79365081
46	0.79365081	0.79365081	0.89757115	0.85831517	0.99019438	0.79365081
47	0.79365081	0.79365081	1.0443652	0.86586368	1.0341145	0.79365081
48	0.79365081	0.79365081	1.1831371	0.84102261	1.0767646	0.79365081
49	0.79365081	0.79365081	1.26	0.86726952	1.1670161	0.79365081
50	0.79365081	0.79365081	1.26	0.89460015	1.26	0.79365081
51	0.79365081	0.79365081	1.26	0.80968457	1.26	0.79365081
52	0.79365081	0.79365081	1.26	0.79365081	1.26	0.80474436
53	0.79365081	0.79365081	1.26	0.79365081	1.26	0.87647426
54	0.79365081	0.87519455	1.26	0.79365081	1.26	0.93920386
55	0.79365081	0.98095393	1.26	0.79365081	1.26	0.99611974
56	0.79365081	1.0163895	1.26	0.79365081	1.26	0.97187245
57	0.79365081	1.0274936	1.26	0.79365081	1.26	0.93501246

Hybrid QMF band	COLOR_180_110 $G_{EQ,20}$	COLOR_090_030 $G_{EQ,16}$	COLOR_060_110 $G_{EQ,17}$	COLOR_135_110 $G_{EQ,19}$	COLOR_090_110 $G_{EQ,18}$	COLOR_060_030 $G_{EQ,15}$
58	0.79365081	1.0639353	1.26	0.79365081	1.26	0.95474315
59	0.79365081	1.1517589	1.26	0.79365081	1.26	1.0320153
60	0.79365081	1.22469	1.26	0.79365081	1.26	1.1006937
61	0.79365081	1.26	1.2404966	0.79365081	1.26	1.1356862
62	0.79365081	1.26	1.226367	0.79365081	1.26	1.1986256
63	0.79365081	1.26	1.2287724	0.79365081	1.26	1.26
64	0.79365081	1.26	1.223058	0.79365081	1.26	1.26
65	0.79365081	1.26	1.1695373	0.79365081	1.26	1.26
66	0.79365081	1.26	1.1292651	0.79365081	1.26	1.26
67	0.79365081	1.26	1.0702422	0.79365081	1.1744785	1.26
68	0.79365081	1.26	0.96692592	0.79365081	1.0516917	1.26
69	0.79365081	1.26	0.96692592	0.79365081	1.0516917	1.26
70	0.79365081	1.26	0.99395496	0.79365081	0.97151875	1.26

11.4.1.6.7 Elevation rendering

11.4.1.6.7.1 General

Elevation rendering is intended to provide an overhead sound image when using a 5.1 channel layout. When the **Destination** column contains **VIRTUAL**, the variable **isPossibleElev** determines whether the input channel shall be rendered by the elevation rendering and is determined by comparing the output channel configuration and the required output channels for the spatial elevation rendering.

Table 175 — Detail information of the elevation rendering initialization parameters

$\mathbf{S}^P$	Vector of converter source channels [input channel indices]	Initialization parameters for spatial elevation rendering (renderElevSptlParms)
$\mathbf{D}^P$	Vector converter destination channels [output channel indices]	
$\mathbf{G}^P{}_H$	Vector of converter gains for 2.8~10kHz components	
$\mathbf{G}^P{}_L$	Vector of converter gains below 2.8kHz and above 10kHz components	
$\mathbf{E}^P$	Vector of converter EQ indices	
$\mathbf{S}^S$	Vector of converter source channels [input channel indices]	Initialization parameters for timbral elevation rendering (renderElevTmbrParms)
$\mathbf{D}^S$	Vector converter destination channels [output channel indices]	
$\mathbf{G}^S$	Vector of converter gains [linear]	
$\mathbf{E}^S$	Vector of converter EQ indices	
$\mathbf{G}_{EQ}$	Matrix containing equalizer gain values for all EQ indices and frequency bands	EQ matrix for all rendering types

When **isPossibleElev** is **TRUE**, two sets of parameters for the spatial elevation rendering and timbral elevation rendering are initialized using **renderElevSptlParms** and **renderElevTmbrParms**. The parameters of $\mathbf{S}^P$, $\mathbf{D}^P$, $\mathbf{G}^P{}_H$, $\mathbf{G}^P{}_L$, and $\mathbf{E}^P$, initialized by **renderElevSptlParms**, are for the spatial elevation rendering, and $\mathbf{S}^S$, $\mathbf{D}^S$, $\mathbf{G}^S$, and $\mathbf{E}^S$, initialized by **renderElevTmbrParms**, are for the timbral elevation rendering. The intermediate parameters describe the dowmixing parameters per mapping index i, i.e. as sets of parameters $S^P{}_i$, $D^P{}_i$, $G^P{}_{iH}$, $G^P{}_{iL}$, and $E^P{}_i$.

11.4.1.6.7.2 isPossibleElev : Decision whether elevation rendering is valid for the output layout

The **isPossibleElev** boolean function returns **TRUE** when all of the required output channels exist. The required output channels are defined in Table 176, indicating 1 for the required channel and 0 for the unnecessary channel. For example, CH_M_L030, CH_M_L110, and CH_M_R110 are required in order to use elevation rendering for the input channel CH_U_L030. If any of the required output channels for the input channel is not in the output channel configuration, **isPossibleElev** shall return **FALSE** and elevation rendering shall not be applied for that input channel.

Table 176 — Required output channels for elevation rendering for isPossibleElev

Input Channel	Required Output Channels					
	CH_M_L030	CH_M_R030	CH_M_000	CH_LFE1	CH_M_L110	CH_M_R110
CH_U_000	1	1	1	0	1	1
CH_U_L045	1	1	0	0	1	1
CH_U_R045	1	1	0	0	1	1
CH_U_L030	1	1	0	0	1	1
CH_U_R030	1	1	0	0	1	1
CH_U_L090	1	0	0	0	1	1
CH_U_R090	0	1	0	0	1	1
CH_U_L110	1	0	0	0	1	1
CH_U_R110	0	1	0	0	1	1
CH_U_L135	1	0	0	0	1	1
CH_U_R135	0	1	0	0	1	1
CH_U_180	1	1	0	0	1	1
CH_T_000	1	1	1	0	1	1

11.4.1.6.7.3 initElevSptlParms : Initialization of elevation rendering parameters based on the input channel elevation

The initial values of the spatial elevation filters and the elevation panning coefficients are defined in Table 177, Table 178, and Table 179 for the 'height' input channels, except CH_T_000. When a 'height' input channel (except CH_T_000) has the elevation higher than 35 degrees, the spatial elevation filter coefficients and elevation panning coefficients shall be updated according to the degree of the elevation. Note that the elevation sector is defined from +21 to +60 degrees for the height channels in Table 182.

For the elevation rendering of an input channel, an EQ is selected from the EQ column of the Table 182. For convenience, $eq(i_{in})$ is defined as the EQ column, e.g. if the i_{in} is CH_U_000, $eq(i_{in})$ is 9 (EQVFC).

Table 177 — Spatial elevation filter initial values (for the 35 degrees in elevation)

Hybrid QMF (71 bands)	$EQ_{0,lin}$ (.)							
	7 (EQVF)	8 (EQVB)	9 (EQVFC)	10 (EQVBC)	11 (EQVOG)	12 (EQVS)	13 (EQBTM)	14 (EQVBA)
0	0.841395	0.819093	0.68508532	0.770312	0.62528018	0.877674	0.922571	0.877674
1	0.841395	0.819093	0.68508532	0.770312	0.62528018	0.877674	0.922571	0.877674
2	0.874312	0.922571	0.71188768	0.940445	0.62528018	0.901571	1.143756	0.988553
3	0.874312	0.922571	0.71188768	0.940445	0.62528018	0.901571	1.143756	0.988553
4	0.926119	0.980995	0.83	1.015469	0.67516235	1	1.117721	1.051155
5	0.926119	0.980995	0.83	1.015469	0.67516235	1	1.117721	1.051155
6	0.944061	0.887837	0.76867877	1	0.72902365	0.905038	1.023293	0.951335
7	0.944061	0.887837	0.76867877	1	0.72902365	0.905038	1.023293	0.951335
8	0.944061	0.887837	0.76867877	1	0.72902365	0.905038	1.023293	0.951335
9	0.908518	0.958665	0.76867877	0.887837	0.70157643	1.039122	0.922571	1.027228
10	0.908518	0.958665	0.76867877	0.887837	0.70157643	1.039122	0.922571	1.027228
11	0.944061	0.936843	0.79875133	0.838172	0.72902365	1.023293	0.940445	0.860994
12	0.944061	0.936843	0.79875133	0.838172	0.72902365	1.023293	0.940445	0.860994
13	0.944061	0.936843	0.79875133	0.838172	0.72902365	1.023293	0.940445	0.860994
14	0.980995	0.962351	0.83	0.922571	0.64974255	0.940445	0.940445	0.992354
15	0.980995	0.962351	0.83	0.922571	0.64974255	0.940445	0.940445	0.992354
16	0.980995	0.962351	0.83	0.922571	0.64974255	0.940445	0.940445	0.992354
17	0.980995	0.962351	0.83	0.922571	0.64974255	0.940445	0.940445	0.992354
18	0.887153	0.916638	0.72234319	0.916638	0.63446387	0.9525	0.829592	0.90963
19	0.875087	0.899232	0.71251848	0.899232	0.62583427	0.934412	0.813839	0.892357
20	0.933012	0.863327	0.7596824	0.830823	0.66726037	0.917998	0.883436	0.856726
21	0.922109	0.849236	0.75080472	0.817263	0.65946291	0.903015	0.869017	0.842743
22	0.912036	0.836276	0.74260266	0.804791	0.65225907	0.889235	0.855756	0.829882
23	0.902688	0.824301	0.73499156	0.793266	0.64557314	0.876501	0.843501	0.817998
24	0.893979	0.816315	0.72790004	0.813189	0.61527373	0.864685	0.922978	0.874698
25	0.885822	0.805907	0.72125921	0.802821	0.60966047	0.853661	0.91121	0.863545
26	0.878166	0.796173	0.71502508	0.793123	0.60439092	0.843349	0.900203	0.853114
27	0.90502	0.802272	0.7368906	0.78401	0.64724144	0.866273	0.93538	0.917603
28	0.897936	0.793493	0.7311221	0.775431	0.6421749	0.856795	0.925145	0.907563
29	0.7943	0.785196	0.64673932	0.815915	0.56805749	0.815915	1.003794	0.918991
30	0.788618	0.777344	0.6421129	0.807755	0.56399394	0.807755	0.993756	0.909801
31	0.783206	0.738062	0.63770643	0.710275	0.53903577	0.769886	0.84093	0.790848
32	0.778048	0.731266	0.63350663	0.703734	0.53548544	0.762797	0.833186	0.783566
33	0.773116	0.724785	0.62949109	0.697497	0.53209122	0.756036	0.825802	0.776621
34	0.659052	0.545115	0.53661741	0.483973	0.48977268	0.566441	0.691546	0.584101

Hybrid QMF (71 bands)	$EQ_{0,lin}$ (.)							
	7 (EQVF)	8 (EQVB)	9 (EQVFC)	10 (EQVBC)	11 (EQVOG)	12 (EQVS)	13 (EQBTM)	14 (EQVBA)
35	0.655171	0.540626	0.53345677	0.479988	0.48688833	0.561776	0.685851	0.579291
36	0.651447	0.536328	0.53042478	0.476172	0.48412123	0.557311	0.6804	0.574686
37	0.647865	0.532204	0.52750816	0.47251	0.48145865	0.553025	0.675167	0.570266
38	0.644418	0.528244	0.52470193	0.468995	0.47889724	0.54891	0.670144	0.566024
39	0.641096	0.524436	0.52199696	0.465614	0.47642896	0.544953	0.665313	0.561943
40	0.637889	0.520767	0.51938578	0.462356	0.39127933	0.54114	0.660658	0.558012
41	0.634793	0.517232	0.51686507	0.459218	0.38937988	0.537467	0.656173	0.554224
42	0.631798	0.513818	0.5144257	0.456187	0.38754274	0.53392	0.651843	0.550566
43	0.628901	0.510523	0.51206767	0.453261	0.3857659	0.530496	0.647662	0.547035
44	0.626093	0.507335	0.50978102	0.450431	0.38404333	0.527183	0.643618	0.543619
45	0.623373	0.504251	0.50756575	0.447693	0.38237503	0.523978	0.639706	0.540315
46	0.620731	0.501262	0.50541522	0.445039	0.3807543	0.520873	0.635914	0.537112
47	0.618168	0.498367	0.5033286	0.442469	0.37918248	0.517864	0.632241	0.53401
48	0.615676	0.495557	0.50129925	0.439974	0.37765354	0.514944	0.628675	0.530998
49	0.613254	0.49283	0.49932717	0.437553	0.37616815	0.51211	0.625216	0.528077
50	0.610896	0.490179	0.49740738	0.435199	0.37472162	0.509356	0.621853	0.525236
51	0.608602	0.487604	0.49553988	0.432913	0.37331462	0.50668	0.618586	0.522477
52	0.606366	0.485097	0.49371886	0.430687	0.37194246	0.504075	0.615406	0.519791
53	0.604187	0.482659	0.49194432	0.428522	0.37060648	0.501541	0.612313	0.517178
54	0.602061	0.480282	0.49021294	0.426412	0.36930199	0.499072	0.609298	0.514632
55	0.599987	0.477968	0.48852472	0.424358	0.36803033	0.496667	0.606362	0.512152
56	0.597962	0.475711	0.48687551	0.422354	0.36678748	0.494321	0.603498	0.509733
57	0.595984	0.47351	0.48526614	0.4204	0.36557478	0.492034	0.600706	0.507375
58	0.594051	0.47136	0.48369163	0.418491	0.36438888	0.489801	0.59798	0.505072
59	0.592162	0.469263	0.48215364	0.416629	0.36322978	0.487622	0.595319	0.502825
60	0.590313	0.467213	0.48064802	0.414809	0.36209614	0.485492	0.592718	0.500628
61	0.588505	0.465211	0.4791756	0.413032	0.36098729	0.483411	0.590178	0.498483
62	0.586734	0.463252	0.47773389	0.411293	0.35990055	0.481376	0.587694	0.496384
63	0.585001	0.461338	0.47632289	0.409593	0.35883726	0.479386	0.585265	0.494332
64	0.583302	0.459463	0.47493928	0.407929	0.35779541	0.477439	0.582887	0.492324
65	0.581638	0.45763	0.47358472	0.406301	0.356775	0.475533	0.58056	0.490359
66	0.580006	0.455833	0.47225589	0.404705	0.35577402	0.473666	0.578281	0.488434
67	0.578407	0.454074	0.47095362	0.403144	0.35479247	0.471838	0.57605	0.486549
68	0.576837	0.45235	0.46967542	0.401613	0.35382968	0.470046	0.573862	0.484701
69	0.575697	0.451098	0.46874665	0.400502	0.3531302	0.468746	0.572275	0.48336
70	0.573327	0.448501	0.46681773	0.398196	0.35167697	0.466048	0.56898	0.480578

When the 'height' input channel i_{in} has higher elevation than 35 degrees, the spatial elevation filter is calculated from $EQ^{k}_{0,lin}(eq(i_{in}))$ in Table 177 by:

If the input channel, i_{in}, is frontal side : azimuth is in the range of (-90, 90)

$$EQ^{k}_{1,db}(eq(i_{in})) = \begin{cases} 20 \cdot \log_{10}\left(EQ^{k}_{0,lin}(eq(i_{in}))\right) + 0.05 \cdot \log_2(f_k \cdot f_s/6000) + 1 & \text{for } f_k \cdot f_s > 8000 \\ 20 \cdot \log_{10}\left(EQ^{k}_{0,lin}(eq(i_{in}))\right) + 1 & \text{otherwise} \end{cases}$$

$$EQ^{k}_{2,db}(eq(i_{in})) = EQ^{k}_{1,db}(eq(i_{in})) \cdot (1 + \min(\max(elv - 35, 0), 25) \cdot 0.05)$$

$$\mathbf{G}^{k}_{EQ,eq(i_{in})} = \begin{cases} 10^{\left(EQ^{k}_{2,db}(eq(i_{in}))-1\right)/20 - 0.05 \cdot \log_2(f_k \cdot f_s/6000)} & \text{for } f_k \cdot f_s > 8000 \\ 10^{\left(EQ^{k}_{2,db}(eq(i_{in}))-1\right)/20} & \text{otherwise} \end{cases}$$

else (the input channel is rear side : azimuth is either in [-180, -90] or in [90, 180])

$$EQ^{k}_{1,db}(eq(i_{in})) = \begin{cases} 20 \cdot \log_{10}\left(EQ^{k}_{0,lin}(eq(i_{in}))\right) + 0.07 \cdot \log_2(f_k \cdot f_s/6000) + 1 & \text{for } f_k \cdot f_s > 8000 \\ 20 \cdot \log_{10}\left(EQ^{k}_{0,lin}(eq(i_{in}))\right) + 1 & \text{otherwise} \end{cases}$$

$$EQ^{k}_{2,db}(eq(i_{in})) = EQ^{k}_{1,db}(eq(i_{in})) \cdot (1 + \min(\max(elv - 35, 0), 25) \cdot 0.05)$$

$$\mathbf{G}^{k}_{EQ,eq(i_{in})} = \begin{cases} 10^{\left(EQ^{k}_{2,db}(eq(i_{in}))-1\right)/20 - 0.07 \cdot \log_2(f_k \cdot f_s/6000)} & \text{for } f_k \cdot f_s > 8000 \\ 10^{\left(EQ^{k}_{2,db}(eq(i_{in}))-1\right)/20} & \text{otherwise} \end{cases}$$

where f_k is the normalized centre frequency of frequency band k, specified in Table 183, f_s is the sampling frequency, and elv is the elevation of the input channel.

When the 'height' input channel i_{in} does not have higher elevation than 35 degrees, use the initial filter coefficients according to:

$$\mathbf{G}^{k}_{EQ,eq(i_{in})} = EQ^{k}_{0,lin}(eq(i_{in}))$$

The spatial elevation panning coefficients shall also be updated for the 'height' input channels, except CH_T_000 and CH_U_180, for the different elevation degrees. Table 178 and Table 179 show the initial panning coefficients for the input channels with the elevation of 35 degrees. When the elevation of the input channel is higher than 35 degrees, the ipsilateral gain applied to the input channel shall be reduced and the contralateral channel shall be boosted with the gain difference $g_I(elv)$ and $g_C(elv)$.

Table 178 — Initial spatial localization panning coefficients for 2,8 kHz ~ 10 kHz (35 degrees in elevation)

Channel label, i_{in}	$G_{vH0,1\sim6}(i_{in})$					
	CH_M_L030 ($G_{vH0,1}(i_{in})$)	CH_M_R030 ($G_{vH0,2}(i_{in})$)	CH_M_000 ($G_{vH0,3}(i_{in})$)	CH_LFE1 ($G_{vH0,4}(i_{in})$)	CH_M_L110 ($G_{vH0,5}(i_{in})$)	CH_M_R110 ($G_{vH0,6}(i_{in})$)
CH_U_000	0.49146774	0.49146774	0.34746769	0	0.44507593	0.44507593
CH_U_L045	0.70918131	0.27444959	0	0	0.56982642	0.31150770
CH_U_R045	0.27444959	0.70918131	0	0	0.31150770	0.56982642
CH_U_L030	0.70918131	0.27444959	0	0	0.56982642	0.31150770
CH_U_R030	0.27444959	0.70918131	0	0	0.31150770	0.56982642
CH_U_L090	0.56040317	0	0	0	0.81550622	0.14456093
CH_U_R090	0	0.56040317	0	0	0.14456093	0.81550622
CH_U_L110	0.34278116	0	0	0	0.91200900	0.22525696
CH_U_R110	0	0.34278116	0	0	0.22525696	0.91200900
CH_U_L135	0.34278116	0	0	0	0.91200900	0.22525696
CH_U_R135	0	0.34278116	0	0	0.22525696	0.91200900
CH_U_180	0.22851810	0.22851810	0	0	0.66916323	0.66916323
CH_T_000	0.45328009	0.45328009	0.33519593	0	0.48822021	0.48822021

Table 179 — Initial spatial localization panning coefficients below 2,8 kHz and above 10 kHz (35 degrees in elevation)

Channel label, i_{in}	$G_{vL0,1\sim6}(i_{in})$					
	CH_M_L030 ($G_{vL0,1}(i_{in})$)	CH_M_R030 ($G_{vL0,2}(i_{in})$)	CH_M_000 ($G_{vL0,3}(i_{in})$)	CH_LFE1 ($G_{vL0,4}(i_{in})$)	CH_M_L110 ($G_{vL0,5}(i_{in})$)	CH_M_R110 ($G_{vL0,6}(i_{in})$)
CH_U_000	0.61940062	0.61940062	0.43791625	0	0	0
CH_U_L045	1	0	0	0	0	0
CH_U_R045	0	1	0	0	0	0
CH_U_L030	1	0	0	0	0	0
CH_U_R030	0	1	0	0	0	0
CH_U_L090	0.36730000	0	0	0	0.93010002	0
CH_U_R090	0	0.36730000	0	0	0	0.93010002
CH_U_L110	0	0	0	0	1	0
CH_U_R110	0	0	0	0	0	1
CH_U_L135	0.34278116	0	0	0	0.91200900	0.22525696
CH_U_R135	0	0.34278116	0	0	0.22525696	0.91200900
CH_U_180	0.22851810	0.22851810	0	0	0.66916323	0.66916323
CH_T_000	0.45328009	0.45328009	0.33519593	0	0.48822021	0.48822021

For all height input channel i_{in}, the $G_{vH,1\sim6}$ and $G_{vL,1\sim6}$ shall be initialized with $G_{vH0,1\sim6}$:

$$G_{vH,1\sim6}(i_{in}) = G_{vH0,1\sim6}(i_{in})$$

$$G_{vL,1\sim6}(i_{in}) = G_{vL0,1\sim6}(i_{in})$$

For each height input channel i_{in}, the $G_{vH,1\sim6}$ shall be calculated from $G_{vH0,1\sim6}$:

If the input channel is CH_U_000,

$$G_{vH,5}(i_{in}) = 10^{(0.25\cdot\min(\max(elv-35,0),25))/20} \cdot G_{vH0,5}(i_{in})$$

$$G_{vH,6}(i_{in}) = 10^{(0.25\cdot\min(\max(elv-35,0),25))/20} \cdot G_{vH0,6}(i_{in})$$

Elseif the input channel is not CH_U_180

if the input channel is side : azimuth is either in (-110, -70) or in (70, 110),

$$g_I(elv) = 10^{(-0.05522\cdot\min(\max(elv-35,0),25))/20}$$

$$g_C(elv) = 10^{(0.41879\cdot\min(\max(elv-35,0),25))/20}$$

else (the input channel is frontal or rear in [-70, 70], [-180, -110], or [110 180])

$$g_I(elv) = 10^{(-0.047401\cdot\min(\max(elv-35,0),25))/20}$$

$$g_C(elv) = 10^{(0.14985\cdot\min(\max(elv-35,0),25))/20}$$

For each output channels ipsilateral to the input channel,
(e.g. CH_M_L030 and CH_M_L110 for CH_U_L045)

$$G_{vH,I}(i_{in}) = g_I(elv) \cdot G_{vH0,I}(i_{in})$$

where the I of $G_{vH,I}(i_{in})$ represents the indices ipsilateral to the i_{in}

For each output channels contralateral to the input channel,
(e.g. CH_M_R030 and CH_M_R110 for CH_U_L045)

$$G_{vH,C}(i_{in}) = g_C(elv) \cdot G_{vH0,C}(i_{in})$$

where the C of $G_{vH,C}(i_{in})$ represents the indices contralateral to the i_{in}

$$G_{vH,C}(i_{in}) = g_C(elv) \cdot G_{vH0,C}(i_{in})$$

where the C of $G_{vH,C}(i_{in})$ represents the indices contralateral to the i_{in}

A set of spatial localization panning coefficients for the input channel is normalized to make the sum of powers be 1.

$$P_{G_{vH}}(i_{in}) = \sqrt{\sum_{o=1}^{6} G_{vH,o}^2(i_{in})}$$

$$G_{vH,1\sim6}(i_{in}) = \frac{1}{P_{G_{vH}}} G_{vH,1\sim6}(i_{in})$$

If the input channel is in [-160, -110) or (110 160], $G_{vL,C}(i_{in})$ shall be updated :

$$G_{vL,I}(i_{in}) = g_I(elv) \cdot G_{vL0,I}(i_{in})$$

$$G_{vL,C}(i_{in}) = g_C(elv) \cdot G_{vL0,C}(i_{in})$$

$$P_{G_{vL}}(i_{in}) = \sqrt{\sum_{o=1}^{6} G_{vL,o}^2(i_{in})}$$

$$G_{vL,1\sim6}(i_{in}) = \frac{1}{P_{G_{vL}}} G_{vL,1\sim6}(i_{in})$$

Note that only the panning coefficients of the $G_{vL,1\sim6}$ for the rear input channels change and those for the frontal/side input channel do not.

As a result of **initElevSptlParms** following parameters are derived:

$\mathbf{G}^{k}_{EQ,eq(i_{in})}$: Updated spatial elevation filter coefficient vector (71band) for the input channel i_{in}.

$G_{vH,1\sim6}(i_{in})$: Updated spatial elevation panning coefficients for the input signal in the range of 2.8~10 kHz

$G_{vL,1\sim6}(i_{in})$: Updated spatial elevation panning coefficients for the input signal below 2.8 kHz and above 10 kHz

11.4.1.6.7.4 renderElevSptlParms : Derivation of input-output channel mapping and equalizer for spatial elevation rendering

renderElevSptlParms initializes the input-output channel mapping for spatial elevation rendering for an input channel (i_{in}).

Initialize the mapping counter i=1;

For m=1 to 6

- If $G_{vH,m}(i_{in}) > 1$
  - $s^P{}_i$ = index of source channel i_{in}
  - $d^P{}_i$ = index of channel m in output
  - $g^P{}_{iH}$ = (value of Gain column) * $G_{vH,m}(i_{in})$
  - $g^P{}_{iL}$ = (value of Gain column) * $G_{vL,m}(i_{in})$
  - $e^P{}_i$ = eq(i_{in})
  - $i = i + 1$

$n^P = i - 1$;

return { $\mathbf{s}^P$, $\mathbf{d}^P$, $\mathbf{g}^P{}_H$, $\mathbf{g}^P{}_L$, $\mathbf{e}^P$, and n^P }

Note that the initialization does not add the input-output channel mapping that the gain $g^P{}_{iH}$ is zero and eq(i_{in}) is from 7 to 15.

By applying the downmix rules defined in Table 182 with the spatial elevation filter and spatial localization panning coefficients, the spatial elevation rendering parameters for each input channel is summarized as:

- CH_U_L135, CH_U_R135, CH_U_180, or CH_T_000

- Spatial elevation filter: the HRTF-based EQs (EQVB , EQVBC, or EQVOG)
- Spatial elevation panning: Filtered signal is multiplied by the same panning coefficients over all frequency (the $\mathbf{g}^{P}{}_{H}$ is identical to the $\mathbf{g}^{P}{}_{L}$)

— CH_U_L110, and CH_U_R110

- Spatial elevation filter: the HRTF-based EQ (EQVBA)
- Spatial elevation panning: Filtered EQ signal is multiplied by the different panning coefficients over frequency range
  - $\mathbf{g}^{P}{}_{H}$ for the elevation-effective range (2,8 k ~ 10 kHz)
  - $\mathbf{g}^{P}{}_{L}$ for the rest frequency range: Use the “add-to-the-closest channel” method in order to provide enough envelopment and keep the audio channel wide enough.

— CH_U_L090 and CH_U_R090

- Spatial elevation filter: the HRTF-based EQs (EQVS)
- Spatial elevation panning: Filtered EQ signal is multiplied by the different panning coefficients over frequency range
  - $\mathbf{g}^{P}{}_{H}$ for the elevation-effective range (2,8 k ~ 10 kHz)
  - $\mathbf{g}^{P}{}_{L}$ for the rest frequency range: Panned at 90 degrees using front and surround loudspeakers in order to provide enough envelopment and keep the audio channel wide enough.

— CH_U_L030, CH_U_R030, CH_U_L045, CH_U_R045

- Spatial elevation filter: the HRTF-based EQs (EQVF)
- Spatial elevation panning: Filtered EQ signal is multiplied by the different panning coefficients over frequency range
  - $\mathbf{g}^{P}{}_{H}$ for the elevation-effective range (2,8 k ~ 10 kHz)
  - $\mathbf{g}^{P}{}_{L}$ for the rest frequency range: Use the “add-to-the-closest channel” method in order to provide enough envelopment and keep the audio channel wide enough.
- Signals for the surround loudspeakers, CH_M_L110 and CH_M_R110, are delayed by approximately 3 msec in order to avoid front-back confusion using the precedence effect. For details see subclause 11.4.2.3

— CH_U_000

- Spatial elevation filter: the HRTF-based EQ (EQVFC)
- Spatial elevation panning: Filtered EQ signal is multiplied by the different panning coefficients over frequency range
  - $\mathbf{g}^{P}{}_{H}$ for the elevation-effective range (2,8 k ~ 10 kHz)
  - $\mathbf{g}^{P}{}_{L}$ for the rest frequency range: panned among three frontal output channels, CH_M_L030, CH_M_000, and CH_M_R030.

— Signals for the surround loudspeakers, CH_M_L110 and CH_M_R110, are delayed by approximately 3 msec in order to avoid front-back confusion using the precedence effect. For details see subclause 11.4.2.3

11.4.1.6.7.5 renderElevTmbrParms : Derivation of input-output channel mapping and equalizer for timbral elevation rendering

In the same manner as **renderElevSptlParms**, **renderElevTmbrParms** initializes another input-output channel mapping for timbral elevation rendering for the same input channel i_{in}. The parameters are initialized following subclause 11.4.1.6.3 but ignoring the downmix rules with a destination field of **VIRTUAL**. As a result, a set of $\mathbf{s}^S$, $\mathbf{d}^S$, $\mathbf{g}^S$, $\mathbf{e}^S$, and $\mathbf{gain}^S$ are defined for the input channel i_{in} by the process:

Initialize the mapping counter i=1;

Search the first entry of the input channel in the Source column of the Table 182 and the channels in the Destination column exist below the entry with VIRTUAL destination.

If **Destination** column contains ALL_U, then:

For each output channel x with "CH_U_" in its name, do:

— s^S_i = index of source channel in input

— d^S_i = index of channel x in output

— g^S_i = (value of Gain column)/sqrt(number of "CH_U_" output channels)

— e^S_i = value of EQ column

— $gain_i$ = (value of Gain column)

— $i = i + 1$

Else if **Destination** column contains ALL_M, then:

For each output channel x with "CH_M_" in its name, do:

— s^S_i = index of source channel in input

— d^S_i = index of channel x in output

— g^S_i = (value of Gain column)/sqrt(number of "CH_M_" output channels)

— e^S_i = value of EQ column

— $gain_i$ = (value of Gain column)

— $i = i + 1$

Else If there is one channel in the **Destination** column, then:

— s^S_i = index of source channel in input

— d^S_i = index of destination channel in output

— g^S_i = value of Gain column

— e^S_i = value of EQ column

— if e^S_i == 13 (EQBTM)

— e^S_i = 0 (No Process)

— $gain_i$ = (value of Gain column)

— $i = i + 1$

Else (two channels in **Destination** column)

— s^S_i = index of source channel in input

— d^S_i = index of first destination channel in output

— g^S_i = (value of Gain column) * g_1

— e^S_i = value of EQ column

— if e^S_i == 13 (EQBTM)

— e^S_i = 0 (No Process)

— $gain_i$ = (value of Gain column)

— $i = i + 1$

— $s^S_i = s^S_{i-1}$

— d^S_i = index of second destination channel in output

— g^S_i = (value of Gain column) * g_2

— $e^S_i = e^S_{i-1}$

— $gain_i$ = (value of Gain column)

— $i = i + 1$

$n^S = i - 1$;

return { $\mathbf{s}^S$, $\mathbf{d}^S$, $\mathbf{g}^S$, $\mathbf{e}^S$, **gain** and n^S }

The gains g_1 and g_2 are computed by applying tangent law amplitude panning in the following way.

1) Unwrap source destination channel azimuth angles to be positive.

2) The azimuth angles of the destination channels are α_1 and α_2 (see Table 180).

3) The azimuth angle of the source channel (= panning target) is α_{src}.

4) $\alpha_0 = \dfrac{|\alpha_1 - \alpha_2|}{2}$

5) $\alpha_{\text{center}} = \dfrac{\alpha_1 + \alpha_2}{2}$

6) $\alpha = (\alpha_{\text{center}} - \alpha_{\text{src}}) \cdot \text{sgn}(\alpha_2 - \alpha_1)$

7) $g_1 = \dfrac{g}{\sqrt{1+g^2}}, \quad g_2 = \dfrac{1}{\sqrt{1+g^2}} \quad \text{with} \quad g = \dfrac{\tan\alpha_0 - \tan\alpha + 10^{-10}}{\tan\alpha_0 + \tan\alpha + 10^{-10}}$

11.4.1.6.7.6 Post-processing for random setups with elevation rendering

After the parameters of $\mathbf{S}^P$, $\mathbf{D}^P$, $\mathbf{G}^P_H$, $\mathbf{G}^P_L$, and $\mathbf{E}^P$ are initialized based on channel information, they are modified according to the azimuth and elevation deviations. For the convenience, $C^P{}_{ii}$ refers the label of

S^P_i, C^P_{io} refers the label of D^P_i, r_{ele,D_i}refers the elevation deviation of the C^P_{io}, and r_{azi,D_i}refers the azimuth deviation of the C^P_{io},

1) Elevation post-processing 1 : Find the “practically identical” channel

For each element *i* in $\mathbf{S}^P$, do

$flag(i)$=0

For each element *i* in $\mathbf{S}^P$, do

If $r_{ele,D_i} > 20$ and $r_{azi,D_i} \leq 15$
if (C^P_{io} == CH_M_L030 and (C^P_{ii} == CH_U_L030 || C^P_{ii} == CH_M_L045)) ||
(C^P_{io} == CH_M_R030 and (C^P_{ii} == CH_U_R030 || C^P_{ii} == CH_M_R045)) ||
(C^P_{io} == CH_M_000 and C^P_{ii} == CH_U_000) ||
{
G^P_{iH} = 1
G^P_{iL} = 1
$flag(i)$= 1
For each element *j* in $\mathbf{S}^P$, do
If $C^P_{ii} == C^P_{ji}$ && $i \neq j$
G^P_{jH} = 0
G^P_{jL} = 0
$flag(j)$= 1
}
If $r_{ele,D_i} > 20$ and $r_{azi,D_i} \leq 25$
if (C^P_{io} == CH_M_L110 and (C^P_{ii} == CH_U_L110 || C^P_{ii} == CH_M_L135)) ||
(C^P_{io} == CH_M_R110 and (C^P_{ii} == CH_U_R110 || C^P_{ii} == CH_M_R135))
{
G^P_{iH} = 1
G^P_{iL} = 1
$flag(i)$= 1
For each element *j* in $\mathbf{S}^P$, do
If $C^P_{ii} == C^P_{ji}$ && $i \neq j$
G^P_{jH} = 0
G^P_{jL} = 0
$flag(j)$= 1
}
For each element *i* in $\mathbf{S}^P$, do
If $r_{ele,D_i} > 20$ and C^P_{io} == CH_M_L110 and C^P_{ii} == CH_U_L090
For each element *j* in $\mathbf{S}^P$, do
If $r_{ele,D_j} > 20$ and C^P_{jo} == CH_M_L030 and C^P_{ii} == CH_U_L090
For each element *k* in $\mathbf{S}^P$, do
If C^P_{ki} == CH_U_L090
G^P_{kH} = 0
G^P_{kL} = 0

flag(*k*)= 1
G^{P}_{iH} = g_1
G^{P}_{iL} = g_1
G^{P}_{jH} = g_2
G^{P}_{jL} = g_2
flag(*i*)= 1
flag(*j*)= 1

If r_{ele,D_i} > 20 and C^{P}_{io} == CH_M_R110 and C^{P}_{ii} == CH_U_R090
For each element *j* in $\mathbf{S}^{P}$, do
If r_{ele,D_j} > 20 and C^{P}_{jo} == CH_M_R030 and C^{P}_{ii} == CH_U_R090
For each element *k* in $\mathbf{S}^{P}$, do
If C^{P}_{ki} == CH_U_R090
G^{P}_{kH} = 0
G^{P}_{kL} = 0
flag(*k*)= 1
G^{P}_{iH} = g_1
G^{P}_{iL} = g_1
G^{P}_{jH} = g_2
G^{P}_{jL} = g_2
flag(*i*)= 1
flag(*j*)= 1

The gains g_1 and g_2 are computed by applying tangent law amplitude panning in the following way.

— Unwrap source destination channel azimuth angles to be positive.

— The azimuth with the deviation for C^{P}_{io} and C^{P}_{jo} are α_1 and α_2.

— The azimuth angle of the source channel (= panning target) is α_{src}.

— $\alpha_0 = \dfrac{|\alpha_1 - \alpha_2|}{2}$

— $\alpha_{\text{center}} = \dfrac{\alpha_1 + \alpha_2}{2}$

— $\alpha = (\alpha_{\text{center}} - \alpha_{\text{src}}) \cdot \text{sgn}(\alpha_2 - \alpha_1)$

— $g_1 = \dfrac{g}{\sqrt{1+g^2}}, \quad g_2 = \dfrac{1}{\sqrt{1+g^2}}$ with $g = \dfrac{\tan\alpha_0 - \tan\alpha + 10^{-10}}{\tan\alpha_0 + \tan\alpha + 10^{-10}}$

2) Elevation post-processing 2 on panning coefficients : Find the “practically dual mono” channel

For each element *i* in $\mathbf{S}^{P}$, if *flag*(*i*)==0

If both CH_M_L030 and CH_M_R030 have elevation deviations more than 20 degrees
if C^{P}_{ii} == CH_U_000
if C^{P}_{io} == CH_M_L030 || CH_M_R030
G^{P}_{iH} = 1

G^{P}_{iL} = 1
$flag(i)$=1
else
G^{P}_{iH} = 0
G^{P}_{iL} = 0
$flag(i)$=1
If both CH_M_L110 and CH_M_R110 are elevated
if C^{P}_{ii} == CH_U_180
if C^{P}_{io} == CH_M_L110 || CH_M_R110
G^{P}_{iH} = 1
G^{P}_{iL} = 1
$flag(i)$=1
else
G^{P}_{iH} = 0
G^{P}_{iL} = 0
$flag(i)$=1

3) Elevation post-processing 3 on panning coefficients : Keep the central image

For each element i in $\mathbf{S}^{P}$, if $flag(i)$==0

if C^{P}_{ii} == CH_U_000 || CH_T_000 || CH_U_180
if only one of the output channels of CH_M_L030 or CH_M_R030 has an elevation deviation more than 20 degrees, **then**
if C^{P}_{ii} == CH_U_000 || CH_T_000
if C^{P}_{io} == CH_M_L030 || CH_M_R030
if C^{P}_{io} is elevated
$G^{P}_{iH} = G^{P}_{iH} \times 10^{\frac{3}{20}\cdot\frac{r_{ele,D_i}}{35}}$
if C^{P}_{ii} == CH_U_000
$G^{P}_{iL} = G^{P}_{iL} \times 10^{\frac{2}{20}\cdot\frac{r_{ele,D_i}}{35}}$
if C^{P}_{ii} == CH_T_000
$G^{P}_{iL} = G^{P}_{iL} \times 10^{\frac{3}{20}\cdot\frac{r_{ele,D_i}}{35}}$
$flag(i)$=1
else
$G^{P}_{iH} = G^{P}_{iH} \times 10^{\frac{2}{20}\cdot\frac{r_{ele,D_i}}{35}}$
$G^{P}_{iL} = G^{P}_{iL} \times 10^{\frac{2}{20}\cdot\frac{r_{ele,D_i}}{35}}$
$flag(i)$=1
if C^{P}_{io} == CH_M_000
$G^{P}_{iH} = G^{P}_{iH} \times 10^{\frac{2}{20}\cdot\frac{r_{ele,D_i}}{35}}$
$G^{P}_{iL} = G^{P}_{iL} \times 10^{\frac{2}{20}\cdot\frac{r_{ele,D_i}}{35}}$
$flag(i)$=1
if C^{P}_{ii} == CH_U_180
if C^{P}_{io} == CH_M_L030 || CH_M_R030
if C^{P}_{io} is not elevated

$G^{P}_{iH} = G^{P}_{iH} \times 10^{-\frac{2}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$G^{P}_{iL} = G^{P}_{iL} \times 10^{-\frac{2}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$flag(i)=1$

if only one of the output channels of CH_M_L110 or CH_M_R110 has an elevation deviation more than 20 degrees, **then**

if C^{P}_{ii} == CH_U_180 || CH_T_000

if C^{P}_{io} == CH_M_L110 || CH_M_R110

if C^{P}_{io} is elevated

$G^{P}_{iH} = G^{P}_{iH} \times 10^{\frac{2}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$G^{P}_{iL} = G^{P}_{iL} \times 10^{\frac{2}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$flag(i)=1$

if C^{P}_{ii} == CH_U_000

if C^{P}_{io} == CH_M_L110 || CH_M_R110

if C^{P}_{io} is not elevated

$G^{P}_{iH} = G^{P}_{iH} \times 10^{-\frac{2}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$G^{P}_{iL} = G^{P}_{iL} \times 10^{-\frac{2}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$flag(i)=1$

4) Elevation post-processing 4 on panning coefficients : Keep the L/R balance when the contralateral frontal channel elevated

For each element i in $\mathbf{S}^{P}$, if $flag(i)==0$

if CH_M_L030 is elevated more than 20 degrees and CH_M_R030 is not

if C^{P}_{ii} == CH_U_R030 || C^{P}_{ii} == CH_U_R045, do

if C^{P}_{io} == CH_M_L030

$G^{P}_{iH} = G^{P}_{iH} \times 10^{-\frac{1}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$G^{P}_{iL} = G^{P}_{iL} \times 10^{-\frac{1}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$flag(i)=1$

elseif C^{P}_{io} == CH_M_L110

$G^{P}_{iH} = G^{P}_{iH} \times 10^{-\frac{4}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$G^{P}_{iL} = G^{P}_{iL} \times 10^{-\frac{4}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$flag(i)=1$

elseif C^{P}_{ii} == CH_U_L090, do

if C^{P}_{io} == CH_M_L030

$G^{P}_{iH} = G^{P}_{iH} \times 10^{\frac{8.7}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$flag(i)=1$

elseif C^{P}_{io} == CH_M_L110

$G^{P}_{iH} = G^{P}_{iH} \times 10^{-\frac{4}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$flag(i) =1$

elseif C^{P}_{io} == CH_M_R110

$G^{P}_{iH} = 0$

$flag(i)$=1

elseif C^{P}_{ii} == CH_U_L110 || C^{P}_{ii} == CH_U_L135, do

if C^{P}_{io} == CH_M_L030

$G^{P}_{iH} = G^{P}_{iH} \times 10^{\frac{5.6}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$G^{P}_{iL} = G^{P}_{iL} \times 10^{\frac{5.6}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$flag(i)$=1

elseif C^{P}_{io} == CH_M_L110

$G^{P}_{iH} = G^{P}_{iH} \times 10^{-\frac{4.6}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$G^{P}_{iL} = G^{P}_{iL} \times 10^{-\frac{4,6}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$flag(i)$=1

elseif C^{P}_{io} == CH_M_R110

$G^{P}_{iH} = G^{P}_{iH} \times 10^{-\frac{1}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$G^{P}_{iL} = G^{P}_{iL} \times 10^{-\frac{1}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$flag(i)$=1

elseif CH_M_R030 is elevated more than 20 degrees and CH_M_L030 is not

if C^{P}_{ii} == CH_U_L030 || C^{P}_{ii} == CH_U_L045, do

if C^{P}_{io} == CH_M_R030

$G^{P}_{iH} = G^{P}_{iH} \times 10^{-\frac{1}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$G^{P}_{iL} = G^{P}_{iL} \times 10^{-\frac{1}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$flag(i)$=1

elseif C^{P}_{io} == CH_M_R110

$G^{P}_{iH} = G^{P}_{iH} \times 10^{-\frac{4}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$G^{P}_{iL} = G^{P}_{iL} \times 10^{-\frac{4}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$flag(i)$=1

elseif C^{P}_{ii} == CH_U_R090, do

if C^{P}_{io} == CH_M_R030

$G^{P}_{iH} = G^{P}_{iH} \times 10^{\frac{8.7}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$flag(i)$=1

elseif C^{P}_{io} == CH_M_R110

$G^{P}_{iH} = G^{P}_{iH} \times 10^{-\frac{4}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$flag(i)$ =1

elseif C^{P}_{io} == CH_M_R110

$G^{P}_{iH} = 0$

$flag(i)$=1

elseif C^{P}_{ii} == CH_U_R110 || C^{P}_{ii} == CH_U_R135, do

if C^{P}_{io} == CH_M_R030

$G^{P}_{iH} = G^{P}_{iH} \times 10^{\frac{5.6}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$G^{P}_{iL} = G^{P}_{iL} \times 10^{\frac{5.6}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$flag(i)$=1

elseif C^P_{io} == CH_M_R110

$G^P_{iH} = G^P_{iH} \times 10^{-\frac{4.6}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$G^P_{iL} = G^P_{iL} \times 10^{-\frac{4,6}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$flag(i)$=1

elseif C^P_{io} == CH_M_L110

$G^P_{iH} = G^P_{iH} \times 10^{-\frac{1}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$G^P_{iL} = G^P_{iL} \times 10^{-\frac{1}{20}\cdot\frac{r_{ele,D_i}}{35}}$

$flag(i)$=1

5) Azimuth post-processing on panning coefficients

$fbias = r_{azi,A}$(CH_M_L030)+ $r_{azi,A}$(CH_M_R030)
$bbias = r_{azi,A}$(CH_M_L110)+ $r_{azi,A}$(CH_M_R110)
For each element i in $\mathbf{S}^P$,
if $fbias > 10$
if (C^P_{ii} == CH_U_000 || C^P_{ii} == CH_T_000) && (C^P_{io} == CH_M_L030)

$G^P_{iH} = G^P_{iH} \times 10^{-\frac{1}{20}\cdot\frac{(fbias)}{15}}$

if C^P_{ii} == CH_U_000

$G^P_{iL} = G^P_{iL} \times 10^{-\frac{2}{20}\cdot\frac{(fbias)}{15}}$

elseif C^P_{ii} == CH_T_000

$G^P_{iL} = G^P_{iL} \times 10^{-\frac{1}{20}\cdot\frac{(fbias)}{15}}$

$flag(i)$=1

if $fbias < -10$
if (C^P_{ii} == CH_U_000 || C^P_{ii} == CH_T_000) && (C^P_{io} == CH_M_R030)

$G^P_{iH} = G^P_{iH} \times 10^{\frac{1}{20}\cdot\frac{(fbias)}{15}}$

if C^P_{ii} == CH_U_000

$G^P_{iL} = G^P_{iL} \times 10^{\frac{2}{20}\cdot\frac{(fbias)}{15}}$

elseif C^P_{ii} == CH_T_000

$G^P_{iL} = G^P_{iL} \times 10^{\frac{1}{20}\cdot\frac{(fbias)}{15}}$

$flag(i)$=1

if $bbias > 10$
if (C^P_{ii} == CH_U_180 || C^P_{ii} == CH_T_000) && (C^P_{io} == CH_M_L110)

$G^P_{iH} = G^P_{iH} \times 10^{\frac{1}{20}\cdot\frac{(bbias)}{15}}$

$G^P_{iL} = G^P_{iL} \times 10^{\frac{1}{20}\cdot\frac{(bbias)}{15}}$

$flag(i)$=1

if $bbias < -10$
if (C^P_{ii} == CH_U_180 || C^P_{ii} == CH_T_000) && (C^P_{io} == CH_M_R110)

$G^P_{iH} = G^P_{iH} \times 10^{-\frac{1}{20}\cdot\frac{(bbias)}{15}}$

$G^P_{iL} = G^P_{iL} \times 10^{-\frac{1}{20}\cdot\frac{(bbias)}{15}}$

$flag(i)$=1

6) Spatial elevation coefficient normalization

For each element i in $\mathbf{S}^P$, if $flag(i)$==1
P_{iL} = 0
P_{iH} = 0
For each element i in $\mathbf{S}^P$, if $flag(i)$==1
For each element k in $\mathbf{S}^P$
If $C^P{}_{ii} == C^P{}_{ki}$
$P_{iL} = P_{iL} + (G^P{}_{iL})^2$
$P_{iH} = P_{iH} + (G^P{}_{iH})^2$
$flag(i)$==0
For each element k in $\mathbf{S}^P$
If $C^P{}_{ii} == C^P{}_{ki}$
$G^P{}_{iL} = G^P{}_{iL} \times \sqrt{P_{iL}}$
$G^P{}_{iH} = G^P{}_{iH} \times \sqrt{P_{iH}}$
$flag(i)$==0

7) Elevation post-processing on spatial elevation filters

For each element i in $\mathbf{S}^P$

If $C^P{}_{io}$ is elevated ($r_{ele,A}(C^P{}_{io}) > 20$)

$E^P{}_i$ = 0 (no EQ)

8) Update the $\mathbf{E}^S$, $\mathbf{G}_{EQ}$ by same process defined in subclause 11.4.1.6.5 as below.

For each element i in $D^S{}_i$, do:

if the output channel with index $D^S{}_i$ is a horizontal channel by definition (i.e. output channel label contains the label '_M_'), **and**

if this output channel is now a height channel (elevation in range 0..60 degrees), **and**

if input channel with index $S^S{}_i$ is a height channel (i.e. label contains '_U_'), **then**

— h = min(elevation of randomized output channel, 35) / 35
— $G_{comp} = h \cdot \frac{1}{Gain_i^S} + (1 - h)$
— Apply compensation gain to DMX gain: $G^S{}_i = G^S{}_i \times G_{comp}$
— Define new equalizer $\mathbf{G}_{EQ,e}$ with the index e, where $G^k_{EQ,e} = h + (1-h) \cdot G^k_{EQ,E_i}$
— $E^S{}_i = e$
— $e = e + 1$

else if input channel with index $S^S{}_i$ is a horizontal channel (label contains '_M_')

— h = min(elevation of randomized output channel, 35) / 35

— Define new equalizer $\mathbf{G}_{EQ,e}$ with the index e,
where $G^k_{EQ,e} = h \cdot G^k_{EQ,5} + (1-h) \cdot G^k_{EQ,E_i}$
— $E^S_i = e$
— $e = e + 1$

11.4.1.6.7.7 Merge general downmix rules and elevation rules

After the elevation rendering parameters are initialized, the vectors of $\mathbf{S}^P$, $\mathbf{D}^P$, $\mathbf{G}^P{}_H$, $\mathbf{G}^P{}_L$, and $\mathbf{E}^P$ that cover the elevation rendered height input channel shall be merged with **S**, **D**, **G**, **E**, and $\mathbf{G}_{EQ}$ that cover the rest of the input channels by:

$$\mathbf{S}^P = \begin{bmatrix}\mathbf{S}^P \\ \mathbf{S}\end{bmatrix}, \quad \mathbf{D}^P = \begin{bmatrix}\mathbf{D}^P \\ \mathbf{D}\end{bmatrix}, \quad \mathbf{G}^P{}_H = \begin{bmatrix}\mathbf{G}^P{}_H \\ \mathbf{G}\end{bmatrix}, \quad \mathbf{G}^P{}_L = \begin{bmatrix}\mathbf{G}^P{}_L \\ \mathbf{G}\end{bmatrix}, \quad \mathbf{E}^P = \begin{bmatrix}\mathbf{E}^P \\ \mathbf{E}\end{bmatrix}$$

$\mathbf{S}^S$, $\mathbf{D}^S$, $\mathbf{G}^S$, and $\mathbf{E}^S$ are also merged with **S**, **D**, **G**, and **E** by:

$$\mathbf{S}^S = \begin{bmatrix}\mathbf{S}^S \\ \mathbf{S}\end{bmatrix}, \quad \mathbf{D}^S = \begin{bmatrix}\mathbf{D}^S \\ \mathbf{D}\end{bmatrix}, \quad \mathbf{G}^S = \begin{bmatrix}\mathbf{G}^S \\ \mathbf{G}\end{bmatrix}, \quad \mathbf{E}^S = \begin{bmatrix}\mathbf{E}^S \\ \mathbf{E}\end{bmatrix},$$

When no input channel is rendered by elevation rendering, the vectors of $\mathbf{S}^P$, $\mathbf{D}^P$, $\mathbf{G}^P{}_H$ (or $\mathbf{G}^P{}_L$), and $\mathbf{E}^P$ are identical to $\mathbf{S}^S$, $\mathbf{D}^S$, $\mathbf{G}^S$, and $\mathbf{E}^S$ and also to **S**, **D**, **G**, and **E**.

11.4.1.6.8 Derivation of rules-based initialization downmix matrix:

$\mathbf{M}_{DMX}$ and $\mathbf{M}_{DMX2}$ are derived by rearranging the temporary parameters from the mapping-oriented representation (enumerated by mapping counter i) to a channel-oriented representation as defined in the following.

Initialize $\mathbf{M}_{DMX}$ and $\mathbf{M}_{DMX2}$ as $N_{out} \times N_{in}$ zero matrixes for all processing bands k.

For each i in $\mathbf{S}^P$ do:

If ($E^P{}_i = 0$)

$M^k_{DMX,A,B} = G^P{}_{iL}$ for $A = D^P{}_i$, $B = S^P{}_i$, for $f_k \times f_s / 2 < 2\,800$

$M^k_{DMX,A,B} = G^P{}_{iH}$ for $A = D^P{}_i$, $B = S^P{}_i$, for $2\,800 \le f_k \times f_s / 2 \le 10\,000$

$M^k_{DMX,A,B} = G^P{}_{iL}$ for $A = D^P{}_i$, $B = S^P{}_i$, for $f_k \times f_s / 2 > 10\,000$

Else

$M^k_{DMX,A,B} = G^P{}_{iL} \times G^k{}_{EQ,E_i^P}$ for $A = D^P{}_i$, $B = S^P{}_i$, $f_k \times f_s / 2 < 2\,800$

$M^k_{DMX,A,B} = G^P{}_{iH} \times G^k{}_{EQ,E_i^P}$ for $A = D^P{}_i$, $B = S^P{}_i$, $2\,800 \le f_k \times f_s / 2 \le 10\,000$

$M^k_{DMX,A,B} = G^P{}_{iL} \times G^k{}_{EQ,E_i^P}$ for $A = D^P{}_i$, $B = S^P{}_i$, $f_k \times f_s / 2 > 10\,000$

For each i in $\mathbf{S}^S$ do:

If ($E^S{}_i = 0$)

$$M^k_{DMX2,A,B} = G^S{}_i \text{ for } A = D^S{}_i,\ B = S^S{}_i, \qquad \text{for } 1 \le k \le K$$

Else

$$M^k_{DMX2,A,B} = G^S{}_i \times G^k{}_{EQ,E^S_i} \text{ for } A = D^S{}_i,\ B = S^S{}_i,\ 1 \le k \le K$$

where f_k is the normalized centre frequency of frequency band k, specified in Table 183, f_s is the sampling frequency, $M^k_{DMX,A,B}$ and $M^k_{DMX2,A,B}$ denotes the matrix element in the Ath row and Bth column of $\mathbf{M}_{DMX}$ and $\mathbf{M}_{DMX2}$. After the rules-based initialization this matrix of downmix coefficients will contain columns of zeros if unknown channels are present in the input format. Those columns of zeros shall be filled with downmix gains as specified in subclause 11.4.1.6.9.

11.4.1.6.9 VBAP-based downmix coefficients derivation

This subclause defines how to generically derive downmix gains using VBAP in case of unknown output formats or unknown input channels. The following restrictions apply.

— If the target setup contains at least one LFE, then map each LFE channel directly to the LFE of the target setup that minimizes the azimuth angle deviation. No VBAP-based downmix coefficients derivation shall be applied for the LFE channels. The downmix coefficient for the direct mapping shall be set to unity gain, i.e. to 1.0.

— Otherwise apply the VBAP-based downmix coefficients derivation defined in the following also to the LFE channels.

Handling of unknown output formats:

In case the output format is considered unknown, the downmix coefficients for all input channels shall be derived as follows.

Each channel of the input setup is regarded as a static audio object at the position defined by the azimuth and elevation angles associated with the input channel. For each input channel the mixing gains to all output loudspeakers are calculated as VBAP panning gains $\mathbf{g}_{\mathbf{scaled}}$ according to 8.4.4, where the same output format shall be signalled to the VBAP algorithm as to the format converter. The panning gain vectors $\mathbf{g}_{\mathbf{scaled}}$ shall be post-processed according to subclause 11.4.1.6.10.

The downmix matrix $\mathbf{M}^k_{\mathrm{DMX}}$ is finally derived by filling each matrix column with the post-processed panning gain vector elements of the corresponding input channel, independently of the processing band index k.

Handling of unknown input channels:

In case the input format contains unknown input channels, the downmix coefficients for these channels shall be derived as follows:

Each unknown channel of the input setup is regarded as a static audio object at the position defined by the azimuth and elevation angles associated with the input channel. For each unknown input channel the mixing gains to all output loudspeakers are calculated as VBAP panning gains $\mathbf{g}_{\mathbf{scaled}}$ according to subclause 8.4.4, where the same output format shall be signalled to the VBAP algorithm as to the format converter. The panning gain vectors $\mathbf{g}_{\mathbf{scaled}}$ shall be post-processed according to subclause 11.4.1.6.10.

The downmix matrix $\mathbf{M}_{\mathrm{DMX}}^{k}$ is finally derived by filling each matrix column corresponding to an unknown input channel with the post-processed panning gain vector elements of the corresponding unknown input channel, independently of the processing band index k.

11.4.1.6.10 VBAP gains post-processing

The mixing gains obtained from the VBAP rendering algorithm shall be post-processed to avoid excessive use of phantom sources. Therefore, small matrix gains are set to zero, followed by a renormalization of the panning gains to ensure energy-preservation.

For each panning gain vector $\mathbf{g}_{\mathbf{scaled}}$ do:

— If the vector contains at least one panning gain that exceeds the threshold value 0.3, then:

— Set all vector elements smaller or equal to 0.3 to the value 0.0;

— Normalize the gain vector such that the sum of squares of the vector elements remains the same as before the post-processing.

11.4.1.7 Format converter initialization tables

Table 180 lists channel labels, corresponding azimuth and elevation angles, and associated sectors. The sectors are defined as points on the unit sphere, whose azimuth/elevation angles are within or on the borders of the intervals given by the azimuth/elevation start and end values in the table, connecting azimuth start and end values in a counter-clockwise direction and connecting elevation start and end values in the direction of increasing elevation angles.

Table 180 — Channels definitions: Channel labels, corresponding azimuth and elevation angles, and associated sectors

Loudspeaker Geometry (as defined in ISO/IEC 23001-8)	**Channel**	**Azimuth** [deg]	**Elevation** [deg]	**Azimuth start angle of sector** [deg]	**Azimuth end angle of sector** [deg]	**Elevation start angle of sector** [deg]	**Elevation end angle of sector** [deg]	**Ch. is LFE**	**Position is relative**
	CH_EMPTY	n/a	n/a	n/a	n/a	n/a	n/a	0	0
0	CH_M_L030	+30	0	+23	+37	-9	+20	0	0
1	CH_M_R030	-30	0	-37	-23	-9	+20	0	0
2	CH_M_000	0	0	-7	+7	-9	+20	0	0
3	CH_LFE1	0	n/a	n/a	n/a	n/a	n/a	1	0
4	CH_M_L110	+110	0	+101	+124	-45	+20	0	0
5	CH_M_R110	-110	0	-124	-101	-45	+20	0	0
6	CH_M_L022	+22	0	+8	+22	-9	+20	0	0
7	CH_M_R022	-22	0	-22	-8	-9	+20	0	0
8	CH_M_L135	+135	0	125	142	-45	+20	0	0
9	CH_M_R135	-135	0	-142	-125	-45	+20	0	0
10	CH_M_180	180	0	158	-158	-45	+20	0	0
13	CH_M_L090	+90	0	+76	+100	-45	+20	0	0
14	CH_M_R090	-90	0	-100	-76	-45	+20	0	0
15	CH_M_L060	+60	0	+53	+75	-9	+20	0	0
16	CH_M_R060	-60	0	-75	-53	-9	+20	0	0
17	CH_U_L030	+30	+35	+11	+37	+21	+60	0	0
18	CH_U_R030	-30	+35	-37	-11	+21	+60	0	0
19	CH_U_000	0	+35	-10	+10	+21	+60	0	0
20	CH_U_L135	+135	+35	+125	+157	+21	+60	0	0

Loudspeaker Geometry (as defined in ISO/IEC 23001-8)	Channel	Azimuth [deg]	Elevation [deg]	Azimuth start angle of sector [deg]	Azimuth end angle of sector [deg]	Elevation start angle of sector [deg]	Elevation end angle of sector [deg]	Ch. is LFE	Position is relative
21	CH_U_R135	-135	+35	-157	-125	+21	+60	0	0
22	CH_U_180	180	+35	+158	-158	+21	+60	0	0
23	CH_U_L090	+90	+35	+67	+100	+21	+60	0	0
24	CH_U_R090	-90	+35	-100	-67	+21	+60	0	0
25	CH_T_000	0	+90	-180	+180	+61	+90	0	0
26	CH_LFE2	+45	n/a	n/a	n/a	n/a	n/a	1	0
27	CH_L_L045	+45	-15	+11	+75	-45	-10	0	0
28	CH_L_R045	-45	-15	-75	-11	-45	-10	0	0
29	CH_L_000	0	-15	-10	+10	-45	-10	0	0
30	CH_U_L110	+110	+35	+101	+124	+21	+60	0	0
31	CH_U_R110	-110	+35	-124	-101	+21	+60	0	0
32	CH_U_L045	+45	+35	+38	+66	+21	+60	0	0
33	CH_U_R045	-45	+35	-66	-38	+21	+60	0	0
34	CH_M_L045	+45	0	+38	+52	-9	+20	0	0
35	CH_M_R045	-45	0	-52	-38	-9	+20	0	0
36	CH_LFE3	-45	n/a	n/a	n/a	n/a	n/a	1	0
37	CH_M_LSCR	+60	0	n/a	n/a	n/a	n/a	0	1
38	CH_M_RSCR	-60	0	n/a	n/a	n/a	n/a	0	1
39	CH_M_LSCH	+30	0	n/a	n/a	n/a	n/a	0	1
40	CH_M_RSCH	-30	0	n/a	n/a	n/a	n/a	0	1
41	CH_M_L150	+150	0	143	157	-45	+20	0	0
42	CH_M_R150	-150	0	-157	-143	-45	+20	0	0

Table 181 — Formats with corresponding number of channels and channel ordering

Loudspeaker layout index or ChannelConfiguration as defined in ISO/IEC 23001-8	Number of channels	Channels (with ordering)
1	1	CH_M_000
2	2	CH_M_L030, CH_M_R030
3	3	CH_M_L030, CH_M_R030, CH_M_000
4	4	CH_M_L030, CH_M_R030, CH_M_000, CH_M180
5	5	CH_M_L030, CH_M_R030, CH_M_000, CH_M_L110, CH_M_R110
6	6	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110
7	8	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_M_L060, CH_M_R060
8		n.a.
9	3	CH_M_L030, CH_M_R030, CH_M_180
10	4	CH_M_L030, CH_M_R030, CH_M_L110, CH_M_R110
11	7	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_M_180
12	8	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_M_L135, CH_M_R135
13	24	CH_M_L060, CH_M_R060, CH_M_000, CH_LFE2, CH_M_L135, CH_M_R135, CH_M_L030, CH_M_R030, CH_M_180, CH_LFE3, CH_M_L090, CH_M_R090, CH_U_L045, CH_U_R045, CH_U_000, CH_T_000, CH_U_L135, CH_U_R135, CH_U_L090, CH_U_R090, CH_U_180, CH_L_000, CH_L_L045, CH_L_R045

Loudspeaker layout index or ChannelConfiguration as defined in ISO/IEC 23001-8	Number of channels	Channels (with ordering)
14	8	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_U_L030, CH_U_R030
15	12	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE2, CH_M_L135, CH_M_R135, CH_LFE3, CH_M_L090, CH_M_R090, CH_U_L045, CH_U_R045, CH_U_180
16	10	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_U_L030, CH_U_R030, CH_U_L110, CH_U_R110
17	12	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_U_L030, CH_U_R030, CH_U_000, CH_U_L110, CH_U_R110, CH_T_000
18	14	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_M_L150, CH_M_R150, CH_U_L030, CH_U_R030, CH_U_000, CH_U_L110, CH_U_R110, CH_T_000
19	12	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L135, CH_M_R135, CH_M_L090, CH_M_R090, CH_U_L030, CH_U_R030, CH_U_L135, CH_U_R135
20	14	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L135, CH_M_R135, CH_M_L090, CH_M_R090, CH_U_L045, CH_U_R045, CH_U_L135, CH_U_R135, CH_M_LSCR, CH_M_RSCR

Table 182 — Converter rules matrix

Source	Destination	Gain	EQ index
CH_M_000	CH_M_L022, CH_M_R022	1.0	0 (off)
CH_M_000	CH_M_L030, CH_M_R030	1.0	0 (off)
CH_M_L022	CH_M_000, CH_M_L030	1.0	0 (off)
CH_M_L022	CH_M_L030	1.0	0 (off)
CH_M_R022	CH_M_000, CH_M_R030	1.0	0 (off)
CH_M_R022	CH_M_R030	1.0	0 (off)
CH_M_L045	CH_M_L030, CH_M_L060	1.0	0 (off)
CH_M_L045	CH_M_L030	1.0	0 (off)
CH_M_R045	CH_M_R030, CH_M_R060	1.0	0 (off)
CH_M_R045	CH_M_R030	1.0	0 (off)
CH_M_L060	CH_M_L045, CH_M_L110	1.0	0 (off)
CH_M_L060	CH_M_L030, CH_M_L110	1.0	0 (off)
CH_M_L060	CH_M_L030	0.8	0 (off)
CH_M_R060	CH_M_R045, CH_M_R110,	1.0	0 (off)
CH_M_R060	CH_M_R030, CH_M_R110,	1.0	0 (off)
CH_M_R060	CH_M_R030,	0.8	0 (off)
CH_M_L090	CH_M_L045, CH_M_L110	1.0	0 (off)
CH_M_L090	CH_M_L030, CH_M_L110	1.0	0 (off)
CH_M_L090	CH_M_L030	0.8	0 (off)
CH_M_R090	CH_M_R045, CH_M_R110	1.0	0 (off)
CH_M_R090	CH_M_R030, CH_M_R110	1.0	0 (off)
CH_M_R090	CH_M_R030	0.8	0 (off)
CH_M_L110	CH_M_L135	1.0	0 (off)
CH_M_L110	CH_M_L090	0.8	0 (off)
CH_M_L110	CH_M_L045	0.8	0 (off)
CH_M_L110	CH_M_L030	0.8	0 (off)
CH_M_R110	CH_M_R135	1.0	0 (off)
CH_M_R110	CH_M_R090	0.8	0 (off)
CH_M_R110	CH_M_R045	0.8	0 (off)

Source	Destination	Gain	EQ index
CH_M_R110	CH_M_R030	0.8	0 (off)
CH_M_L135	CH_M_L110	1.0	0 (off)
CH_M_L135	CH_M_L150	1.0	0 (off)
CH_M_L135	CH_M_L090	0.8	0 (off)
CH_M_L135	CH_M_L045	0.8	0 (off)
CH_M_L135	CH_M_L030	0.8	0 (off)
CH_M_R135	CH_M_R110	1.0	0 (off)
CH_M_R135	CH_M_R150	1.0	0 (off)
CH_M_R135	CH_M_R090	0.8	0 (off)
CH_M_R135	CH_M_R045	0.8	0 (off)
CH_M_R135	CH_M_R030	0.8	0 (off)
CH_M_L150	CH_M_L135	1.0	0 (off)
CH_M_L150	CH_M_L110	1.0	0 (off)
CH_M_L150	CH_M_L045	0.8	0 (off)
CH_M_L150	CH_M_L030	0.8	0 (off)
CH_M_R150	CH_M_R135	1.0	0 (off)
CH_M_R150	CH_M_R110	1.0	0 (off)
CH_M_R150	CH_M_R045	0.8	0 (off)
CH_M_R150	CH_M_R030	0.8	0 (off)
CH_M_180	CH_M_R150, CH_M_L150	1.0	0 (off)
CH_M_180	CH_M_R135, CH_M_L135	1.0	0 (off)
CH_M_180	CH_M_R110, CH_M_L110	1.0	0 (off)
CH_M_180	CH_M_R090, CH_M_L090	0.8	0 (off)
CH_M_180	CH_M_R045, CH_M_L045	0.6	0 (off)
CH_M_180	CH_M_R030, CH_M_L030	0.6	0 (off)
CH_U_000	CH_U_L030, CH_U_R030	1.0	0 (off)
CH_U_000	VIRTUAL	1.0	9 (EQVFC)
CH_U_000	CH_M_L030, CH_M_R030	0.85	0 (off)
CH_U_L045	CH_U_L030	1.0	0 (off)
CH_U_L045	VIRTUAL	1.0	7 (EQVF)
CH_U_L045	CH_M_L045	0.85	1
CH_U_L045	CH_M_L030	0.85	1
CH_U_R045	CH_U_R030	1.0	0 (off)
CH_U_R045	VIRTUAL	1.0	7 (EQVF)
CH_U_R045	CH_M_R045	0.85	1
CH_U_R045	CH_M_R030	0.85	1
CH_U_L030	CH_U_L045	1.0	0 (off)
CH_U_L030	VIRTUAL	1.0	7 (EQVF)
CH_U_L030	CH_M_L030	0.85	1
CH_U_R030	CH_U_R045	1.0	0 (off)
CH_U_R030	VIRTUAL	1.0	7 (EQVF)
CH_U_R030	CH_M_R030	0.85	1
CH_U_L090	CH_U_L030, CH_U_L110	1.0	0 (off)
CH_U_L090	CH_U_L030, CH_U_L135	1.0	0 (off)
CH_U_L090	CH_U_L045	0.8	0 (off)
CH_U_L090	CH_U_L030	0.8	0 (off)
CH_U_L090	VIRTUAL	1.0	12 (EQVS)
CH_U_L090	CH_M_L045, CH_M_L110	0.85	2
CH_U_L090	CH_M_L030, CH_M_L110	0.85	2
CH_U_L090	CH_M_L030	0.85	2
CH_U_R090	CH_U_R030, CH_U_R110	1.0	0 (off)
CH_U_R090	CH_U_R030, CH_U_R135	1.0	0 (off)
CH_U_R090	CH_U_R045	0.8	0 (off)

Source	Destination	Gain	EQ index
CH_U_R090	CH_U_R030	0.8	0 (off)
CH_U_R090	VIRTUAL	1.0	12 (EQVS)
CH_U_R090	CH_M_R045, CH_M_R110	0.85	2
CH_U_R090	CH_M_R030, CH_M_R110	0.85	2
CH_U_R090	CH_M_R030	0.85	2
CH_U_L110	CH_U_L135	1.0	0 (off)
CH_U_L110	CH_U_L045	0.8	0 (off)
CH_U_L110	CH_U_L030	0.8	0 (off)
CH_U_L110	VIRTUAL	1.0	14 (EQVBA)
CH_U_L110	CH_M_L110	0.85	2
CH_U_L110	CH_M_L045	0.85	2
CH_U_L110	CH_M_L030	0.85	2
CH_U_R110	CH_U_R135	1.0	0 (off)
CH_U_R110	CH_U_R045	0.8	0 (off)
CH_U_R110	CH_U_R030	0.8	0 (off)
CH_U_R110	VIRTUAL	1.0	14 (EQVBA)
CH_U_R110	CH_M_R110	0.85	2
CH_U_R110	CH_M_R045	0.85	2
CH_U_R110	CH_M_R030	0.85	2
CH_U_L135	CH_U_L110	1.0	0 (off)
CH_U_L135	CH_U_L045	0.8	0 (off)
CH_U_L135	CH_U_L030	0.8	0 (off)
CH_U_L135	VIRTUAL	1.0	8 (EQVB)
CH_U_L135	CH_M_L110	0.85	2
CH_U_L135	CH_M_L045	0.85	2
CH_U_L135	CH_M_L030	0.85	2
CH_U_R135	CH_U_R110	1.0	0 (off)
CH_U_R135	CH_U_R045	0.8	0 (off)
CH_U_R135	CH_U_R030	0.8	0 (off)
CH_U_R135	VIRTUAL	1.0	8 (EQVB)
CH_U_R135	CH_M_R110	0.85	2
CH_U_R135	CH_M_R045	0.85	2
CH_U_R135	CH_M_R030	0.85	2
CH_U_180	CH_U_R135, CH_U_L135	1.0	0 (off)
CH_U_180	CH_U_R110, CH_U_L110	1.0	0 (off)
CH_U_180	VIRTUAL	1.0	10 (EQVBC)
CH_U_180	CH_M_180	0.85	2
CH_U_180	CH_M_R110, CH_M_L110	0.85	2
CH_U_180	CH_U_R030, CH_U_L030	0.8	0 (off)
CH_U_180	CH_M_R030, CH_M_L030	0.85	2
CH_T_000	ALL_U	0.8	3
CH_T_000	VIRTUAL	1.0	11 (EQVOG)
CH_T_000	ALL_M	0.8	4
CH_L_000	CH_M_000	1.0	13 (EQBTM)
CH_L_000	CH_M_L030, CH_M_R030	1.0	13 (EQBTM)
CH_L_L045	CH_M_L045	1.0	13 (EQBTM)
CH_L_L045	CH_M_L030	1.0	13 (EQBTM)
CH_L_R045	CH_M_R045	1.0	13 (EQBTM)
CH_L_R045	CH_M_R030	1.0	13 (EQBTM)
CH_LFE1	CH_LFE2	1.0	0 (off)
CH_LFE1	CH_M_L030, CH_M_R030	1.0	0 (off)
CH_LFE2	CH_LFE1	1.0	0 (off)
CH_LFE2	CH_M_L030, CH_M_R030	1.0	0 (off)

Table 183 — Normalized centre frequencies of the 71 filter-bank bands

Normalized frequency [0, 1]
0,004 583 30
0,000 833 33
0,002 083 30
0,005 875 00
0,009 791 70
0,014 292 00
0,019 792 00
0,027 000 00
0,035 417 00
0,042 625 00
0,056 750 00
0,072 375 00
0,088 000 00
0,103 620 00
0,119 250 00
0,134 870 00
0,150 500 00
0,166 120 00
0,181 750 00
0,197 370 00
0,213 000 00
0,228 620 00
0,244 250 00
0,259 880 00
0,275 500 00
0,291 130 00
0,306 750 00
0,322 380 00
0,338 000 00
0,353 630 00
0,369 250 00
0,384 880 00
0,400 500 00
0,416 130 00
0,431 750 00
0,447 380 00
0,463 000 00
0,478 630 00
0,494 250 00
0,509 870 00
0,525 500 00
0,541 120 00
0,556 750 00
0,572 370 00
0,588 000 00
0,603 620 00
0,619 250 00
0,634 870 00
0,650 500 00
0,666 120 00
0,681 750 00
0,697 370 00

Normalized frequency [0, 1]
0,713 000 00
0,728 620 00
0,744 250 00
0,759 870 00
0,775 500 00
0,791 120 00
0,806 750 00
0,822 370 00
0,838 000 00
0,853 620 00
0,869 250 00
0,884 870 00
0,900 500 00
0,916 120 00
0,931 750 00
0,947 370 00
0,963 000 00
0,974 540 00
0,999 040 00

Table 184 — Equalizer parameters

Equalizer	P_f [Hz]	P_Q	P_g [dB]	g [dB]
$G_{EQ,1}$	12 000	0,3	-2	1,0
$G_{EQ,2}$	12 000	0,3	-3,5	1,0
$G_{EQ,3}$	200,1 300, 600	0,3, 0,5, 1,0	-6,5, 1,8, 2,0	0,7
$G_{EQ,4}$	5 000, 1 100	1,0, 0,8	4,5, 1,8	-3,1
$G_{EQ,5}$	35	0,25	-1,3	1,0

Table 185 — Vertically corresponding channels

CH_L_000	CH_M_000	CH_U_000
CH_L_L045	CH_M_L030	CH_U_L030
CH_L_L045	CH_M_L030	CH_U_L045
CH_L_L045	CH_M_L045	CH_U_L030
CH_L_L045	CH_M_L045	CH_U_L045
CH_L_L045	CH_M_L060	CH_U_L030
CH_L_L045	CH_M_L060	CH_U_L045
CH_L_R045	CH_M_R030	CH_U_R030
CH_L_R045	CH_M_R030	CH_U_R045
CH_L_R045	CH_M_R045	CH_U_R030
CH_L_R045	CH_M_R045	CH_U_R045
CH_L_R045	CH_M_R060	CH_U_R030
CH_L_R045	CH_M_R060	CH_U_R045
CH_M_180	CH_U_180	
CH_M_L090	CH_U_L090	
CH_M_L110	CH_U_L110	
CH_M_L135	CH_U_L135	
CH_M_L090	CH_U_L110	
CH_M_L090	CH_U_L135	

CH_M_L110	CH_U_L090	
CH_M_L110	CH_U_L135	
CH_M_L135	CH_U_L090	
CH_M_L135	CH_U_L135	
CH_M_R090	CH_U_R090	
CH_M_R110	CH_U_R110	
CH_M_R135	CH_U_R135	
CH_M_R090	CH_U_R110	
CH_M_R090	CH_U_R135	
CH_M_R110	CH_U_R090	
CH_M_R110	CH_U_R135	
CH_M_R135	CH_U_R090	
CH_M_R135	CH_U_R135	
NOTE Each row lists channels which are considered to be above/below each other.		

11.4.2 Audio signal processing

11.4.2.1 General

The audio processing block of the format converter obtains time domain audio samples for N_{in} channels from the core decoder and generates a downmixed time domain audio output signal consisting of N_{out} channels.

The processing takes as input:

— the audio data and the flag **rendering3DType** decoded by the core decoder;

— the static downmix matrixes $\mathbf{M}_{DMX}$ and $\mathbf{M}_{DMX2}$ returned by the initialization of the format converter.

It returns an N_{out}-channel time domain output signal for the OutConf channel configuration signalled during the initialization of the format converter.

The format converter operates on contiguous, non-overlapping frames of length L = 2048 time domain samples of the input audio signals and outputs one frame of L samples per processed input frame of length L.

The **rendering3DType** is used in the selection of the downmix matrix as shown in Figure 55.

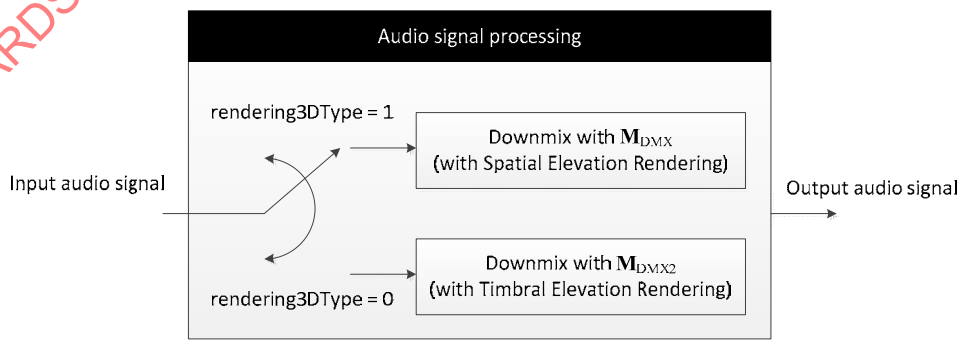


Figure 55 — Audio signal processing with switching scheme by rendering3DType

11.4.2.2 T/F-transform (hybrid QMF analysis)

As the first processing step the converter transforms L = 2048 samples of the N_{in} channel time domain input signal $\left[\tilde{\mathbf{y}}^{v}_{\text{ch},1}\ldots\tilde{\mathbf{y}}^{v}_{\text{ch},N_{\text{in}}}\right]=\tilde{\mathbf{y}}^{v}_{\text{ch}}$ to a hybrid QMF N_{in} channel signal representation consisting of $L_{\text{n}}=32$ QMF time slots (slot index n) and K = 71 frequency bands (band index k). A QMF analysis according to ISO/IEC 14496-3:2009, subclause 4.6.18.4, is performed first:

$$\left[\hat{y}^{n,k}_{\text{ch},1}\ldots\hat{y}^{n,k}_{\text{ch},N_{\text{in}}}\right]=\hat{\mathbf{y}}^{n,k}_{\text{ch}}=\text{QmfAnalysis}\left(\tilde{\mathbf{y}}^{v}_{\text{ch}}\right)\quad\text{with}\quad 0\le v<L\quad\text{and}\quad 0\le n<L_n,$$

followed by a hybrid analysis:

$$\left[y^{n,k}_{\text{ch},1}\ldots y^{n,k}_{\text{ch},N_{\text{in}}}\right]=\mathbf{y}^{n,k}_{\text{ch}}=\text{HybridAnalysis}\left(\hat{\mathbf{y}}^{n,k}_{\text{ch}}\right).$$

The hybrid filtering shall be carried out as specified in ISO/IEC 14496-3:2009, 8.6.4.3 for the 10,20 bands configuration of parametric stereo, resulting in a 71-band hybrid QMF domain representation.

11.4.2.3 Delayed channels and downmix matrix modification

If there exists a frontal height input channel to be rendered by spatial elevation rendering, delayed QMF samples of the input channel are added to the input QMF samples and the downmix matrix is expanded with modified coefficients. This step leads to the spatial elevation rendered signal being more stable by avoiding the front-back confusion. The process is achieved by:

For each input channel i in [1 N_{in}], do

If the i^{th} input channel has attached one of the labels CH_U_L030, CH_U_L045, CH_U_R030, CH_U_R045, and CH_U_000

If the channel i is to be rendered by spatial rendering

(1) Create delayed QMF samples of the input channel by:

$$delay=\text{round}(f_s\cdot 0.003/64)$$

$$\mathbf{y}^{\text{n,k}}_{\text{ch}}=\begin{cases}\left[\mathbf{y}^{\text{n,k}}_{\text{ch}}\quad y^{n-delay,k}_{\text{ch},i}\right] & \text{for n}\ge\text{delay}\\ \left[\mathbf{y}^{\text{n,k}}_{\text{ch}}\quad y^{L_n+(n-delay),k}_{\text{ch,prev},i}\right] & \text{otherwise}\end{cases},$$

$where\ y^{L_n+(n-delay),k}_{\text{ch,prev},i}$ is the $L_n+(n-delay)^{\text{th}}$ QMF subband sample in k$^{\text{th}}$ band of the previous frame

(2) Modify the downmix matrix $\mathbf{M}_{\text{DMX}}$ for the CH_M_L110 and CH_M_R110

$$\mathbf{M}_{\text{DMX}}=[\mathbf{M}_{\text{DMX}}\quad M_{DMX,1\sim N_{out},i}]$$

$$\mathbf{M}_{\text{DMX2}}=[\mathbf{M}_{\text{DMX2}}\quad[0\ 0\ \ldots 0]^{\text{T}}]$$

$$N_{in}=N_{in}+1$$

For each j in [1 N_{out}], do

If the channel j^{th} output channel has attached the label of either CH_M_L110 or CH_M_R110

$$M_{DMX,j,i} = 0$$

Else

$$M_{DMX,j,N_{in}} = 0$$

Explanation: By reproducing the spatial elevation rendered output surround channel for the frontal height input channel with a delay of approximately 3 msec, front-back confusion is avoided through the precedence effect. The delay for 48 kHz sampled input signal is 2 subband samples.

11.4.2.4 Covariance analysis

Note that for clarity the frequency band parameter (superscript k) is omitted in the following equations if it is not required for the presentation.

Let F be a monotonically increasing frame index denoting the current frame of input data, e.g. $\mathbf{y}_{\text{ch}}^{F,n} = \mathbf{y}_{\text{ch}}^{n}$ for frame F, starting at $F = 0$ for the first frame of input data after initialization of the format converter. An analysis frame of length $2L_n$ is constructed from the input hybrid QMF spectra as:

$$\mathbf{y}_{\text{in, ch}}^{F,n} = \begin{cases} 0 & \text{,for} \quad 0 \le n < L_n \quad , \quad F = 0 \\ \mathbf{y}_{\text{in, ch}}^{F-1,n+L_n} & \text{,for} \quad 0 \le n < L_n \quad , \quad F > 0 \\ \mathbf{y}_{\text{ch}}^{F,n-L_n} & \text{,for} \quad L_n \le n < 2L_n \quad , \quad F \ge 0 \end{cases}$$

Note that $\mathbf{y}_{\text{in, ch}}^{F,n}$ is a row vector with N_{in} elements in case of N_{in} input channels. The covariance matrix is analysed from four quarter segments of $\mathbf{y}_{\text{in, ch}}^{F,n}$, so that:

$$\mathbf{C}_{y,q}^{F} = \sum_{n=16q}^{16q+15} \left(\mathbf{y}_{\text{in, ch}}^{F,n}\right)^{\text{T}} \left(\mathbf{y}_{\text{in, ch}}^{F,n}\right)^{*} \quad \text{for} \quad q = 0,1,2,3$$

where

$(\cdot)^{\text{T}}$ denotes the transpose;

$(\cdot)^{*}$ denotes the complex conjugate of a variable;

$\mathbf{C}_{y,q}^{F}$ is an $N_{\text{in}} \times N_{\text{in}}$ matrix for each $q = 0,1,2,3$.

Note that $\mathbf{C}_{y,0}^{F}$ and $\mathbf{C}_{y,1}^{F}$ are the same as $\mathbf{C}_{y,2}^{F-1}$ and $\mathbf{C}_{y,3}^{F-1}$, correspondingly, and do not need to be re-calculated. The covariance matrices of the four quarter segments are added with centre weighting assuming a staircase shape:

$$\mathbf{C}_{y,\text{sum}}^{\text{F}} = \mathbf{C}_{y,0}^{F} + 4\mathbf{C}_{y,1}^{F} + 4\mathbf{C}_{y,2}^{F} + \mathbf{C}_{y,3}^{F}$$

The final estimation for the covariance matrix $\mathbf{C}_{y}^{F}$ is obtained by modifying the entries of $\mathbf{C}_{y,\text{sum}}^{F}$ with a small channel dependent offset:

$$C_{y,a,b} = (1 + 0.0002\sqrt{ab})C_{y,\mathrm{sum},a,b}$$

where the two indices in a notation $C_{y,a,b}$ denote the matrix element in the *a*th row and *b*th column of $\mathbf{C}_y$. From the covariance matrix $\mathbf{C}_y$ inter-channel correlation coefficients between the channels *A* and *B* are derived as follows:

$$ICC_{A,B} = \sqrt{\frac{\left|C_{y,A,B}\right|^2}{\mathrm{eps} + C_{y,A,A} \cdot C_{y,B,B}}}$$

11.4.2.5 Phase-alignment matrix formulation

11.4.2.5.1 General

The $ICC_{A,B}$ values are mapped to an attraction measure matrix **T** with elements:

$$T_{A,B} = \begin{cases} \min\left(PasMax, \max\left(0, PasCurveSlope \cdot ICC_{A,B} + PasCurveShift\right)\right), & \text{for} \quad A \neq B \\ \min\left(1.0, \max\left(0, PasCurveSlope \cdot 4 \cdot ICC_{A,B} + 4 \cdot PasCurveShift\right)\right), & \text{for} \quad A = B \end{cases},$$

where *PasMax, PasCurveSlope, PasCurveShift* are derived from Table 186.

Table 186 — Phase attraction mapping curve parameters

phaseAlignStrength	PasMax	PasCurveSlope	PasCurveShift
0	0	0	0
1	0,071 4	0,170 1	-0,089 1
2	0,154 8	0,377 1	-0,189 6
3	0,25	0,625	-0,3
4	0,357 1	0,918 4	-0,418 4
5	0,476 2	1,262 3	-0,542 7
6	0,607 1	1,662 4	-0,670 7
7	0,75	2,125	-0,8
phaseAlignStrength shall be set to 3 if no other value has been signalled in the bitstream. If passiveDownmixFlag==1, then phaseAlignStrength shall be set to 0.			

An intermediate phase-aligning mixing matrix $\mathbf{M}_{\mathrm{int}}$ is calculated. With

$$P_{A,B} = T_{A,B} \cdot C_{y,A,B}$$

compute $\mathbf{M}_{\mathrm{int}}$ depending on the **rendering3DType** by:

If **rendering3DType** == TRUE

$\mathbf{V} = \mathbf{M}_{\mathrm{DMX}}\mathbf{P}$

If the input channel A is rendered by spatial elevation rendering,

$M^k_{int,B,A} = M^k_{DMX,B,A}$ for $f_k \times f_s / 2 \geq 2\,800$ and $f_k \times f_s / 2 \leq 10\,000$

$M^k_{int,B,A} = \frac{M^k_{DMX,B,A}}{eps + |V_{B,A}|} V_{B,A}$ otherwise

else

$$M^{k}_{int,B,A} = \frac{M^{k}_{DMX,B,A}}{eps + |V_{B,A}|} V_{B,A}$$

else

$$\mathbf{V} = \mathbf{M}_{\mathrm{DMX2}}\mathbf{P}$$

$$M^{k}_{int,B,A} = \frac{M^{k}_{DMX2,B,A}}{eps + |V_{B,A}|} V_{B,A} \qquad \text{for } all\ k$$

where f_k is the normalized centre frequency of frequency band k, specified in Table 183, f_s is the sampling frequency, and $M^{k}_{DMX,A,B}$ and $M^{k}_{DMX2,A,B}$ denotes the matrix element in the Ath row and Bth column of $\mathbf{M}_{\mathrm{DMX}}$ and $\mathbf{M}_{\mathrm{DMX2}}$.

Note that the phase alignment for the frequency components of 2,8 ~10 kHz for the spatial elevation rendered downmix coefficients are avoided for the spatial elevation rendering when the **rendering3DType** is true. Although the immersive format converter has an active downmix with phase correction, the synchronization of the spatial elevation rendered signal from a height input channel without phase correction plays an important role in providing an overhead sound image in 3D rendering.

The intermediate phase-aligning mixing matrix $\mathbf{M}_{\mathrm{int}}$ is modified to avoid abrupt phase shifts, resulting in $\mathbf{M}_{\mathrm{mod}}$. This is a recursive regularization process, running for each frame F, processing the frequency bands k in ascending order.

The regularization against phase shifts takes place in two stages: In the first stage, the regularization performs amplitude-weighted phase comparison against the previous frame, previous band, while also linking the phase-attracted channels. In the second stage, the regularization limits the update rate of the phase coefficients in comparison to the previous frame only.

Both regularization stages make use of a phase update limiting parameter, $\theta^{F,k}_{\mathrm{diff},A}$, which is formulated as a function of an onset measure $o^{F,k}_{A}$ so that a low energy portion of a signal does not affect the phase processing after an onset:

$$o^{F,k}_{A} = \frac{C^{F,k}_{y,A,A}}{eps + C^{F,k}_{y,A,A} + C^{F-1,k}_{y,A,A}}$$

$$\theta^{F,k}_{\mathrm{diff},A} = \max(0.15, 20.3 o^{F,k}_{A} - 19)\pi$$

11.4.2.5.2 Regularization stage 1

Stage 1 recursively takes into account comparison values $\mathbf{M}_{\mathrm{cmp}}$ from the last frame index (F-1) as well as for the last processing band (k-1). $\mathbf{M}_{\mathrm{cmp}}$ is derived from $\mathbf{M}_{\mathrm{mod}}$ at the end of the regularization process. The first step of regularization stage 1 combines the comparison data across frequency and time as follows:

if (F=0)

$$\mathbf{M}_{\text{cmpFk}}^{F,k} = \begin{cases} 0 & \text{for} \quad k = 1 \\ \mathbf{M}_{\text{cmp}}^{\text{F,k-1}} & \text{for} \quad k > 1 \ \wedge \ k \neq 3 \\ \left(\mathbf{M}_{\text{cmp}}^{\text{F,k-1}}\right)^{*} & \text{for} \quad k = 3 \end{cases}$$

else (i.e. for F>0)

$$\mathbf{M}_{\text{cmpFk}}^{F,k} = \begin{cases} \mathbf{M}_{\text{cmp}}^{F-1,k} & \text{for} \quad k = 1 \\ \mathbf{M}_{\text{cmp}}^{F-1,k} + \mathbf{M}_{\text{cmp}}^{\text{F,k-1}} & \text{for} \quad k > 1 \ \wedge \ k \neq 3 \\ \mathbf{M}_{\text{cmp}}^{F-1,k} + \left(\mathbf{M}_{\text{cmp}}^{\text{F,k-1}}\right)^{*} & \text{for} \quad k = 3 \end{cases}$$

where the complex conjugate processing for the third band (k=3) accounts for the complex conjugate properties of the filterbank.

The frequency index k is omitted in the following since the inter-band dependency is now contained in the matrix $\mathbf{M}_{\text{cmpFk}}^{F,k}$. The phase change of the current unregularized phase-aligning matrix $\mathbf{M}_{\text{int}}^{F}$ relative to $\mathbf{M}_{\text{cmpFk}}^{F,k}$ is measured by amplitude weighting with $\sqrt{C_{y,A,A}^{F}}$ and comparison against $\mathbf{M}_{\text{cmpFk}}^{F}$, forming $\mathbf{M}_{\text{timeFreq}}^{F}$ with elements:

$$M_{\text{timeFreq},B,A}^{F} = M_{\text{cmpFk},B,A}^{F}\left(M_{\text{int},B,A}^{F}\right)^{*}\sqrt{C_{y,A,A}^{F}}$$

To also take into account the interdependent channels in the regularization, the relevant entries are intermixed with the attraction matrix $\mathbf{T}^{F}$

$$\mathbf{M}_{\text{timeFreqChan}}^{F} = \mathbf{M}_{\text{timeFreq}}^{F}\mathbf{T}^{F}$$

The phase values of the elements of matrix $\mathbf{M}_{\text{timeFreqChans}}^{F}$ are:

$$\theta_{\text{timeFreqChan},B,A}^{F} = \arg\left(M_{\text{timeFreqChan},B,A}^{F}\right)$$

To avoid constant phase offsets, $\theta_{\text{timeFreqChan},B,A}^{F}$ is adjusted towards zero by $\theta_{\text{diff},A}^{F}$:

$$\theta_{\text{timeFreqChanStage1},B,A}^{F} = \text{sign}\left(\theta_{\text{timeFreqChan},B,A}^{F}\right)\max\left(0, abs(\theta_{\text{timeFreqChan},B,A}^{F}) - \theta_{\text{diff},A}^{F}\right)$$

11.4.2.5.3 Regularization stage 2

In stage 2 of the regularization another phase comparison parameter, only across time, is calculated:

$$\theta_{\text{time},B,A}^{F} = \arg\left(M_{\text{PA},B,A}^{F-1}\left(M_{\text{int},B,A}^{F}\right)^{*}\right)$$

The final regularization parameter is such that is as close as possible to $\theta^{F}_{\text{timeFreqChanStage1},B,A}$, but not further than $\theta^{F}_{\text{diff},A}$ in respect to $\theta^{F}_{\text{time},B,A}$. Let *unwrap()* be a function that maps any angular parameter to the corresponding angle in the interval $-\pi \ldots \pi$. The final phase parameter is calculated as:

$$\theta^{F}_{\text{update},B,A} = unwrap\left(\theta^{F}_{\text{timeFreqChanStage1},B,A} - \theta^{F}_{\text{time},B,A}\right),$$
$$\theta^{F}_{\text{mod},B,A} = \theta^{F}_{\text{time},B,A} + sign\left(\theta^{F,k}_{\text{update},B,A}\right)\max(0, abs(\theta^{F}_{\text{update},B,A}) - \theta^{F}_{\text{diff},A}),$$

and the modified, i.e. phase-regularized, mixing matrix elements are obtained as:

$$M^{F}_{\text{mod},B,A} = M^{F}_{\text{int},B,A} \cdot \exp\left(j \cdot \theta^{F}_{\text{mod},B,A}\right)$$

Finally, $\mathbf{M}_{\text{cmp}}$ is derived by amplitude weighting the regularized downmixing coefficients:

$$M^{F}_{\text{cmp,B,A}} = M^{F}_{\text{mod},B,A}\sqrt{C^{F}_{y,A,A}}\,.$$

Note that $\mathbf{M}_{\text{cmp}}$ is used in the time and frequency recursive formulation of regularization stage 1.

11.4.2.5.4 Energy scaling

An energy scaling is applied to the mixing matrix to obtain the final phase-aligning mixing matrix $\mathbf{M}_{\text{PA}}$. With

$$\mathbf{M}_{Cy} = \mathbf{M}_{\text{mod}}\mathbf{C}_{y}\mathbf{M}^{\text{H}}_{\text{mod}}$$

where $(\cdot)^{\text{H}}$ denotes the conjugate transpose operator, and

if **rendering3DType** == TRUE

$$S_B = \sqrt{\frac{\sum_{A=1}^{N_{in}} M_{DMX,B,A} \cdot M_{DMX,B,A} \cdot C_{y,A,A}}{eps + M_{C_y,B,B}}}$$

Else

$$S_B = \sqrt{\frac{\sum_{A=1}^{N_{in}} M_{DMX2,B,A} \cdot M_{DMX2,B,A} \cdot C_{y,A,A}}{eps + M_{C_{y,B,B}}}},$$

$$S_{\text{lim},B} = \min\left(S_{\max}, \max\left(S_{\min}, S_B\right)\right)$$

where the limits are defined as $S_{\max} = 10^{0.4}$ and $S_{\min} = 10^{-0.5}$, the final phase-aligning mixing matrix elements follow as:

$$M_{\text{PA},B,A} = \left(S_{\text{lim},B} \cdot AES + (1 - AES)\right) \cdot M_{\text{mod},B.A},$$

where *AES* = (1 – **passiveDownmixFlag**).

11.4.2.6 Calculation of output data

The output signals for the current frame F are computed by linearly interpolating the mixing matrices from the previous frame to the current frame:

$$\mathbf{z}_{\mathrm{ch}}^{F,n} = \left(\left(\frac{n+1}{L_n} \mathbf{M}_{\mathrm{PA}}^{F} + \frac{L_n - n - 1}{L_n} \mathbf{M}_{\mathrm{PA}}^{F-1} \right) \left(\mathbf{y}_{\mathrm{in,\,ch}}^{F,n} \right)^{\mathrm{T}} \right)^{\mathrm{T}} \quad \text{for} \quad 0 \le n < L_n$$

Note that the input audio for the above mixing procedure is the first half of the analysis window.

11.4.2.7 F/T-transform (hybrid QMF synthesis)

Note that the processing steps described above have to be carried out for each hybrid QMF band k (recursively, for ascending k). In the following procedure the band index k is reintroduced, i.e. $\mathbf{z}_{\mathrm{ch}}^{F,n,k} = \mathbf{z}_{\mathrm{ch}}^{F,n}$. The hybrid QMF frequency domain output signal $\mathbf{z}_{\mathrm{ch}}^{F,n,k}$ is transformed to an N_{out}-channel time domain signal frame of length L time domain samples per output channel B, yielding the time domain output signal $\tilde{\mathbf{z}}_{\mathrm{ch}}^{F,v}$.

The hybrid synthesis:

$$\hat{\mathbf{z}}_{\mathrm{ch}}^{F,n,k} = \mathrm{HybridSynthesis}\left(\mathbf{z}_{\mathrm{ch}}^{F,n,k} \right)$$

is carried out as defined in Figure 8.21 of ISO/IEC 14496-3:2009, i.e. by summing the sub-subbands of the three lowest QMF subbands to obtain the three lowest QMF subbands of the 64band QMF representation. The subsequent QMF synthesis:

$$\tilde{\mathbf{z}}_{\mathrm{ch}}^{F,n,k} = \mathrm{QMFSynthesis}\left(\hat{\mathbf{z}}_{\mathrm{ch}}^{F,n,k} \right)$$

shall be carried out as defined in ISO/IEC 14496-3:2009, 4.6.18.4.

12 Higher order ambisonics (HOA)

12.1 Technical overview

12.1.1 Block diagram

A block diagram of the HOA decoder architecture is shown in Figure 56. First, the input bitstream is de-multiplexed and decoded by the MPEG-H 3D audio Core decoder into I PCM transport channels plus the HOA bitstream that contains parameters required to recompose the full HOA representation from these PCM signals. In the successive spatial decoding component, first, the actual value range of these signals is reconstructed by the inverse gain control processing. In a next step, the I signals are re-distributed to provide the M predominant signals and $(I - M)$ HOA coefficient signals representing the more ambient HOA components.

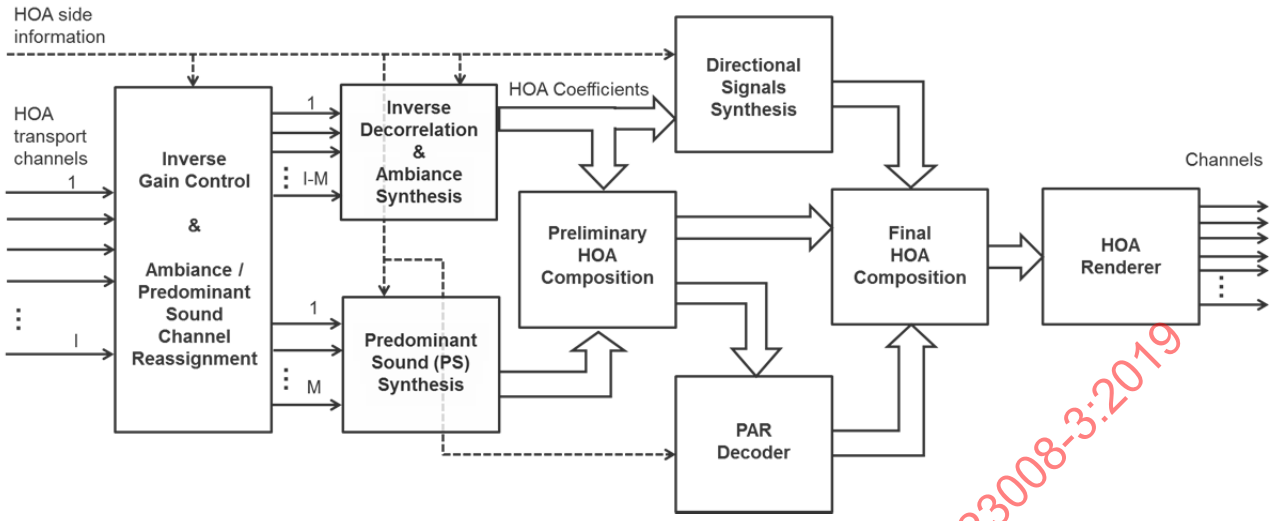


Figure 56 — Simplified decoder

The fixed subset of the $(I - M)$ HOA coefficient signals is re-correlated, this means the decorrelation at the HOA encoding stage is reversed. Next, all of the $(I - M)$ HOA coefficient signals are used to create the ambient HOA component. The ambient HOA component is an input to the directional signals synthesis and the preliminary HOA composition. The directional signals synthesis creates a new HOA representation from the ambient HOA component by prediction. The predominant HOA component is synthesized from the M predominant sound signals and the corresponding parameters. The predominant and the ambient HOA components are combined into the preliminary HOA representation, which is then fed to the parametric ambience replication (PAR). The PAR adds missing ambience components to the preliminary HOA representation, which is parametrically created from its input signals. Finally, the output of the PAR, the preliminary HOA representation and the output of the directional signals synthesis are combined to the decoded HOA representation that is rendered to the loudspeaker setup by the HOA renderer.

12.1.2 Overview of the decoder tools

12.1.2.1 HOA decoding tools

The inputs of the HOA frame converter are the HOA configuration data HOAConfig() and the HOA Frame HOAFrame(), as it is depicted in Figure 57. The variables of Figure 57 are defined in subclauses 12.3 and 12.4.1.

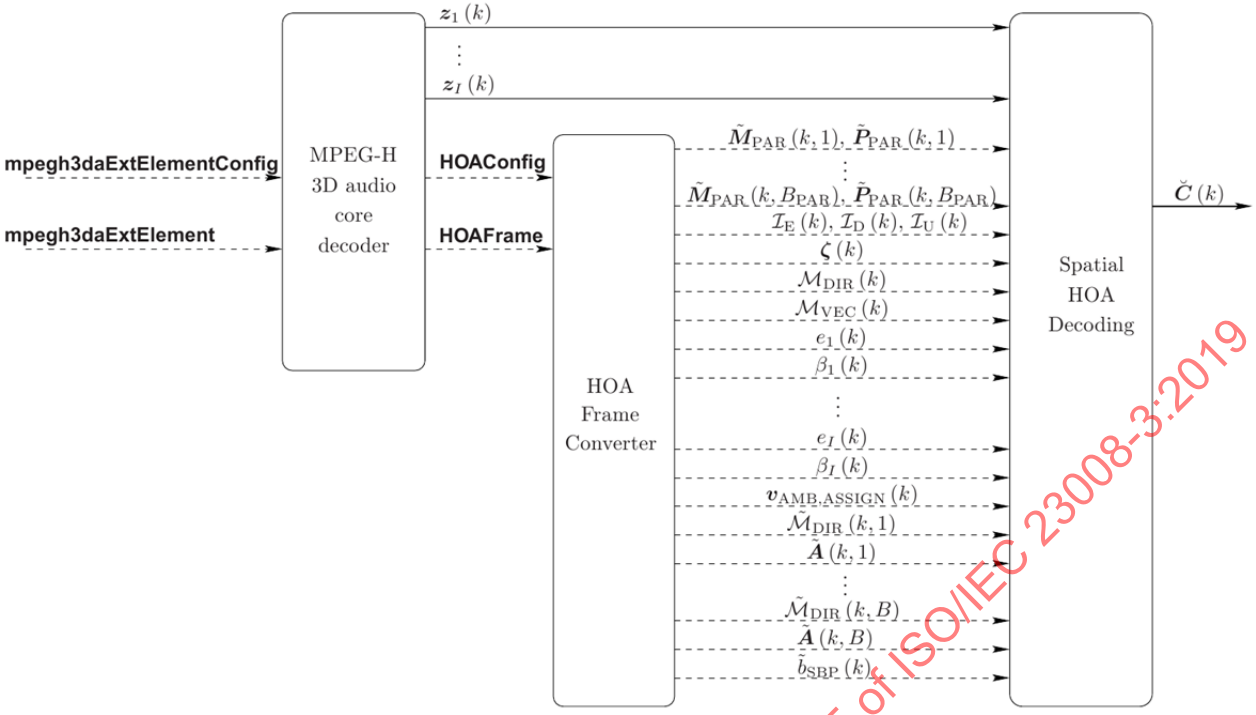


Figure 57 — The architecture of the HOA decoder tools

The HOA frame converter (Convert) and the MPEG-H 3D audio core decoder provide the data for the spatial HOA decoding.

The spatial HOA decoding re-creates the HOA time domain signals of the previous frame from the signals $z_i(k)$ and from the spatial side information provided by the HOA frame converter. The spatial HOA decoding consists of the following coding tools, which are specified in subclause 12.4:

— inverse gain control;

— channel reassignment;

— predominant sound synthesis;

— ambience synthesis;

— preliminary HOA composition;

— sub-band directional signals synthesis;

— parametric ambience replication decoder;

— HOA composition.

12.1.2.2 HOA renderer

The HOA renderer converts the HOA signal matrix $\boldsymbol{C}(k)$ to the loudspeaker signals $\boldsymbol{W}_{\text{SPEAKER}}(k)$ using the loudspeaker position matrix $\boldsymbol{\Omega}_{\text{SPEAKER}}$ and the HOAConfig() (subclause 12.3.1) for its initialization.

12.1.2.3 Layered coding for HOA

For the streaming of the compressed HOA sound field representation over a transmission channel with time-varying conditions layered coding is a means to adapt the quality of the received sound representation to the transmission conditions, and in particular to avoid undesired signal dropouts.

For layered coding the compressed HOA sound field representation is subdivided into a high priority base layer of a relatively small size and additional enhancement layers with decremental priorities and arbitrary sizes. Each enhancement layer is assumed to contain incremental information to complement that of all lower priority layers in order to improve the quality of the compressed HOA sound field representation. The idea is to control the amount of error protection for the transmission of the individual layers according to their priority. In particular, it is intended that the base layer may be provided with a high error protection, which is reasonable and affordable due to its low size.

12.2 Syntax

12.2.1 Configuration of HOA elements

Table 187 — Syntax of HOAConfig()

Syntax	No. of bits	Mnemonic

HOAConfig()
{
  HoaOrder = escapedValue(3,5,0);
  NumOfHoaCoeffs = (HoaOrder + 1)^2;
  IsScreenRelative;
  UsesNfc;
  if (UsesNfc) {
  NfcReferenceDistance;
  }
  HOADecoderConfig(numHOATransportChannels);
}
	
	3,8
	
	1
	1
	
	32
	
		uimsbf
		
		uimsbf
		bslbf
		
		bslbf

NOTE HoaOrder = 30 … 38 are reserved.		

Table 188 — Syntax of HOADecoderConfig()

Syntax	No. of bits	Mnemonic

HOADecoderConfig(numHOATransportChannels)
{
  MinAmbHoaOrder = escapedValue(3,5,0) – 1;
  MinNumOfCoeffsForAmbHOA = (MinAmbHoaOrder + 1)^2;
  NumOfAdditionalCoders = numHOATransportChannels –
  MinNumOfCoeffsForAmbHOA;
  NumLayers = 1;
  NumHOAChannelsLayer[0] = numHOATransportChannels;
  if(**SingleLayer** == 0){
  HOALayerChBits = ceil(log2(NumOfAdditionalCoders));
  NumHOAChannelsLayer[0] = **codedLayerCh** +
  MinNumOfCoeffsForAmbHOA;
  remainingCh = numHOATransportChannels –
  NumHOAChannelsLayer[0];
  while (remainingCh>1) {
  HOALayerChBits = ceil(log2(remainingCh));
  NumHOAChannelsLayer[NumLayers] =
  NumHOAChannelsLayer[NumLayers-1] +
  codedLayerCh + 1;
  remainingCh = remainingCh –
  NumHOAChannelsLayer[NumLayers];
  NumLayers++;
  }
  if (remainingCh) {
	
	3,8
	
	1
	
	HOALayer ChBits
	
	HOALayer ChBits
	
		uimsbf
		
		Bslbf
		
		uimsbf
		
		uimsbf

Syntax	No. of bits	Mnemonic
NumHOAChannelsLayer[NumLayers] = NumOfAdditionalCoders; NumLayers++; } }		
CodedSpatialInterpolationTime;	**3**	**uimsbf**
SpatialInterpolationMethod;	**1**	**bslbf**
CodedVVecLength;	**2**	**uimsbf**
MaxGainCorrAmpExp;	**3**	**uimsbf**
HOAFrameLengthIndicator;	**2**	**uimsbf**

if(MinAmbHoaOrder < HoaOrder) {
DiffOrderBits = ceil(log2(HoaOrder- MinAmbHoaOrder+1))
MaxHoaOrderToBeTransmitted = **DiffOrder** + MinAmbHoaOrder;
}
else {
MaxHoaOrderToBeTransmitted = HoaOrder;
}
MaxNumOfCoeffsToBeTransmitted = (MaxHoaOrderToBeTransmitted + 1)^2;
MaxNumAddActiveAmbCoeffs = MaxNumOfCoeffsToBeTransmitted - MinNumOfCoeffsForAmbHOA;
VqConfBits = ceil (log2(ceil(log2(NumOfHoaCoeffs+1))));		

NumVVecVqElementsBits;	**VqConfBits**	**uimsbf**

if(MinAmbHoaOrder == 1) {
UsePhaseShiftDecorr;
}
	1	**bslbf**

if(SingleLayer==1) {
HOADecoderEnhConfig();
}
AmbAsignmBits = ceil(log2(MaxNumAddActiveAmbCoeffs));
ActivePredIdsBits = ceil(log2(NumOfHoaCoeffs));
i = 1;
while(i * ActivePredIdsBits + ceil(log2(i)) < NumOfHoaCoeffs){
i++;
}
NumActivePredIdsBits = ceil(log2(max(1, i – 1)));
GainCorrPrevAmpExpBits = ceil(log2(ceil(log2(1.5 * NumOfHoaCoeffs)) + MaxGainCorrAmpExp + 1));
}		

NOTE MinAmbHoaOrder = 30 … 37 are reserved. HOAFrameLengthIndicator = 3 is reserved. CodedVVecLength = 3 is reserved.		

Table 189 — Syntax of HOAEnhConfig()

Syntax	No. of bits	Mnemonic

HOAEnhConfig()
{
    HOALayerIdxBits = ceil(log2(NumOfAdditionalCoders+1));
    LayerIdx
 	
    HOADecoderEnhConfig();
}
	
	HOALayerId
	xBits
		
		uimsbf

Table 190 — Syntax of HOADecoderEnhConfig()

Syntax	No. of bits	Mnemonic
HOADecoderEnhConfig()		
{		
MaxNoOfDirSigsForPrediction =		
(**bsMaxNoOfDirSigsForPrediction** + 1);	**2**	**uimsbf**
NoOfBitsPerScalefactor = **bsNoOfBitsPerScalefactor** + 1;	**4**	**uimsbf**
if(**PredSubbandsIdx** < 3) {	**2**	**uimsbf**
NumOfPredSubbands =		
NumOfPredSubbandsTable[PredSubbandsIdx];		
PredSubbandWidths =		
PredSubbandWidthTable[PredSubbandsIdx];		
}		
else {		
NumOfPredSubbands = **bsNumOfPredSubbands**+1;		
PredSubbandWidths =		
getSubbandWidths(NumOfPredSubbands);		
}		
if (NumOfPredSubbands > 0) {		
FirstSBRSubbandIdxBits = ceil(log2 (NumOfPredSubbands+1));		
FirstSBRSubbandIdx;	**FirstSBRS** **ubbandId** **xBits**	**uimsbf**
MaxNumOfPredDirs = 2^(**MaxNumOfPredDirsLog2**);	**3**	**uimsbf**
MaxNumOfPredDirsPerBand = escapedValue(3,2,5) + 1;		
NumOfBitsPerDirIdx =		
NumOfBitsPerDirIdxTable[**DirGridTableIdx**];	**2**	**uimsbf**
}		
if(**ParSubbandTableIdx** < 3) {	**2**	**uimsbf**
NumOfParSubbands =		
NumOfParSubbandsTable[ParSubbandTableIdx];		
ParSubbandWidths =		
ParSubbandWidthTable[ParSubbandTableIdx];		
}		
else {		
NumOfParSubbands = **bsNumOfParSubbands**+1;		
ParSubbandWidths =		
getSubbandWidths(NumOfParSubbands);		
}		
if(NumOfParSubbands > 0) {		
LastFirstOrderSubbandIdxBits =		
ceil(log2(NumOfParSubbands + 1));		

Syntax	No. of bits	Mnemonic
LastFirstOrderSubbandIdx;	**LastFirstOrderSubbandIdxBits**	**uimsbf**
for (idx = 0; idx < NumOfParSubbands; idx++) {		
UseRealCoeffsPerParSubband[idx];	**1**	**bslbf**
}		
for (idx = 0; idx < LastFirstOrderSubBandIdx; idx++) {		
UpmixHoaOrderPerParSubband[idx] = 1;		
MaxNumOfDecoSigs[idx]=		
(UpmixHoaOrderPerParSubband[idx] + 1)^2;		
}		
for (idx = LastFirstOrderSubBandIdx;		
idx < NumOfParSubbands; idx++) {		
UpmixHoaOrderPerParSubband[idx] = 2;		
MaxNumOfDecoSigs[idx] =		
(UpmixHoaOrderPerParSubband[idx] + 1)^2;		
}		
}		
}		

Table 191 — Syntax of getSubbandWidths()

Syntax	No. of bits	Mnemonic
getSubbandWidths(NumberOfSubbands)		
{		
totalBwSum = 0;		
if(NumberOfSubbands > 1) {		
CodedBwFirstBand	**1..**	**uclbf**
bw[0] = CodedBwFirstBand+1;		
totalBwSum = totalBwSum + bw[0];		
if(NumberOfSubbands > 2) {		
for (nb = 1; nb < NumberOfSubbands-2; nb++) {		
bw[nb] = bw[nb-1] + **bw_diff**;	**1..**	**uclbf**
totalBwSum = totalBwSum + bw[nb];		
}		
bw[nb] = bw[nb-1] + **bw_diff**;	**5**	**uimsbf**
totalBwSum = totalBwSum + bw[nb];		
}		
}		
bw[NumberOfSubbands-1] = 64 – totalBwSum;		
return(bw);		
}		

12.2.2 Payloads of HOA elements

Table 192 — Syntax of HOAFrame()

Syntax	No. of bits	Mnemonic
HOAFrame()		
{		
NumOfDirSigs = 0;		
NumOfVecSigs = 0;		
NumOfContAddHoaChans = 0;		
for(lay=0; (lay< NumLayers); ++lay){		

Syntax	No. of bits	Mnemonic
NumOfDirSigsPerLayer[lay] = 0; NumOfAddHoaChansPerLayer[lay] = 0; NumOfContAddHoaChans[lay] = 0; NumOfNewAddHoaChans[lay] = 0; }		
hoaIndependencyFlag;	**1** [a]	**bslbf**

for(i=0; i< NumOfAdditionalCoders; ++i){
ChannelSideInfoData(i);
HOAGainCorrectionData(i);
switch ChannelType[i] {
case 0:
DirSigChannelIds[NumOfDirSigs] = i + 1;
NumOfDirSigs++;
for(lay=0; (lay< NumLayers); ++lay){
if((MinNumOfCoeffsForAmbHOA + i) <
NumHOAChannelsLayer[lay]){
NumOfDirSigsPerLayer[lay]++;
}
}
break;
case 1:
VecSigChannelIds[NumOfVecSigs] = i + 1;
lay = 0;
while((MinNumOfCoeffsForAmbHOA + i)
≥ NumHOAChannelsLayer[lay]) {
lay ++;
}
VecSigLayerIdx[NumOfVecSigs] = lay;
NumOfVecSigs++;
break;
case 2:
for(lay=0; (lay< NumLayers); ++lay){
if((MinNumOfCoeffsForAmbHOA + i) <
NumHOAChannelsLayer[lay]){
if (AmbCoeffTransitionState[i] == 0) {
ContAddHoaCoeff[lay]
[NumOfContAddHoaChans[lay]]
= AmbCoeffIdx[i];
NumOfContAddHoaChans[lay]++;
}
}else{
if(AmbCoeffTransitionState[i] == 1) {
NewAddHoaCoeff[lay]
[NumOfNewAddHoaChans]
= AmbCoeffIdx[i];
NumOfNewAddHoaChans[lay]++;
}
AddHoaCoeffPerLayer[lay]
[NumOfAddHoaChans]
= AmbCoeffIdx[i];
NumOfAddHoaChansPerLayer[lay]++;		

Syntax	No. of bits	Mnemonic

}
}
AddHoaCoeff[NumOfAddHoaChans] = AmbCoeffIdx[i];
NumOfAddHoaChans++;
break;
}
}
for (i= NumOfAdditionalCoders;
i< NumHOATransportChannels; ++i){
HOAGainCorrectionData(i);
}
 	
for(i=0; i< NumOfVecSigs; ++i){
VVectorData (VecSigChannelIds(i));
}
 	
if(SingleLayer==1) {
HOAEnhFrame();
}
}
	b	

[a] The encoder shall set hoaIndependencyFlag to 1 if usacIndependencyFlag (see mpegh3daFrame() in Table 41) is set to 1. [b] If SingleLayer == 1 set NumLayers = 1.		

Table 193 — Syntax of HOAEnhFrame()

Syntax	No. of bits	Mnemonic

HOAEnhFrame()
{
if(((SingleLayer==1) & (NumOfDirSigs > 0)) |
((SingleLayer==0) & (NumOfDirSigsPerLayer[lay]) > 0)){
HOAPredictionInfo(DirSigChannelIds, NumOfDirSigs);
}
if(NumOfPredSubbands > 0){
HOADirectionalPredictionInfo();
}
if(NumOfParSubbands > 0) {
HOAParInfo();
}
}
	a	

[a] lay is the index of the currently active HOA enhancement layer If SingleLayer==0 then use NumOfPredSubbands and NumOfParSubbands from the **HOADecoderEnhConfig** of the corresponding layer with index lay.		

Table 194 — Syntax of ChannelSideInfoData(i)

Syntax	No. of bits	Mnemonic
ChannelSideInfoData(i) { **ChannelType**[i] switch ChannelType[i] { case 0:	**2**	**uimsbf**

Syntax	No. of bits	Mnemonic
ActiveDirsIds[i];	**10**	**uimsbf**
break;		
case 1:		
if(hoaIndependencyFlag){		
if(CodedVVecLength==1){		
NewChannelTypeOne(k)[i];	**1**	**bslbf**
}		
NbitsQ(k)[i]	**4**	**uimsbf**
if (NbitsQ(k)[i] == 4) {		
CodebkIdx(k)[i];	**3**	**uimsbf**
NumVvecIndices(k)[i]++;	**NumVVecVqElementsBits**	**uimsbf**
}		
elseif (NbitsQ(k)[i] >= 6) {		
PFlag(k)[i] = 0;		
CbFlag(k)[i];	**1**	**bslbf**
}		
}		
else{		
if(CodedVVecLength==1){		
NewChannelTypeOne(k)[i] = (1!=ChannelType(k-1)[i]));		
}		
bA;	**1**	**bslbf**
bB;	**1**	**bslbf**
if ((bA + bB) == 0) {		
NbitsQ(k)[i] = NbitsQ(k-1)[i];		
PFlag(k)[i] = PFlag(k-1)[i];		
CbFlag(k)[i] = CbFlag(k-1)[i];		
CodebkIdx(k)[i] = CodebkIdx(k-1)[i];		
NumVvecIndices(k)[i] = NumVvecIndices(k-1)[i];		
}		
else{		
NbitsQ(k)[i] = (8*bA)+(4*bB)+**uintC**;	**2**	**uimsbf**
if (NbitsQ(k)[i] == 4) {		
CodebkIdx(k)[i];	**3**	**uimsbf**
NumVvecIndices(k)[i]++;	**NumVVecVqElementsBits**	**uimsbf**
}		
elseif (NbitsQ(k)[i] >= 6) {		
PFlag(k)[i];	**1**	**bslbf**
CbFlag(k)[i];	**1**	**bslbf**
}		
}		
}		
break;		
case 2:		
AddAmbHoaInfoChannel(i);		
break;		
default:		
}		
}		
NOTE CodebkIdx = 4 … 6 are reserved.		

Table 195 — ChannelType definition

ChannelType: 0 : Direction-based signal 1 : Vector-based signal 2 : Additional ambient HOA coefficient 3: Empty

Table 196 — Syntax of AddAmbHoaInfoChannel(i)

Syntax	No. of bits	Mnemonic

AddAmbHoaInfoChannel(i)
{
if(hoaIndependencyFlag){
AmbCoeffTransitionState[i];
	2	**uimsbf**

AmbCoeffIdx[i] = **CodedAmbCoeffIdx** + 1 + MinNumOfCoeffsForAmbHOA;	**AmbAsignBits** [a]	**uimsbf**

}
else {
if(**AmbCoeffIdxTransition** == 1) {
	1	**bslbf**

if (AmbCoeffTransitionState[i] > 1) { AmbCoeffTransitionState[i] = 1; AmbCoeffIdx[i] = **CodedAmbCoeffIdx** + 1 + MinNumOfCoeffsForAmbHOA;	**AmbAsignBits**	**uimsbf**

}
else {
AmbCoeffTransitionState[i] = 2;
}		

}
else {
AmbCoeffTransitionState[i] = 0;		

} } }		

[a] The AmbCoeffIdx of the preceding frame shall be used under the following conditions
if (AmbCoeffIdxTransitionState == 0 || AmbCoeffIdxTransitionState == 2)

AmbCoeffTransitionState:
0: No transition (continuous additional ambient HOA coefficient)
1: Fade-in of additional ambient HOA coefficient
2: Fade-out of additional ambient HOA coefficient
3: reserved

Table 197 — Syntax of HOAGainCorrectionData()

Syntax	No. of bits	Mnemonic

HOAGainCorrectionData(i)
{
if(hoaIndependencyFlag){		

GainCorrPrevAmpExp[i] = **bsGainCorrPrevAmpExp** - ceil(log2(1.5 * NumHoaCoeffs));	**GainCorrPrevAmpExpBits**	**uimsbf**

Syntax	No. of bits	Mnemonic
}		
n=0;		
while(1) {		
CodedGainCorrectionExp[i][n]	**1**	**bslbf**
if(CodedGainCorrectionExp[i][n])		
break;		
n++;		
}		
GainCorrectionException[i];	**1**	**bslbf**
}		

Table 198 — Syntax of VVectorData()

Syntax	No. of bits	Mnemonic
VVectorData(i)		
{		
if (CodedVVecLength == 1) {		
VVecLengthUsed = VVecLength[i];		
VVecCoeffIdUsed = VVecCoeffId[i];		
} else {		
VVecLengthUsed = VVecLength;		
VVecCoeffIdUsed = VVecCoeffId;		
}		
if (NbitsQ(k)[i] == 4) {		
if (NumVvecIndices(k)[i] == 1) {		
VvecIdx[0] = **bsVvecIdx** + 1;	**nbitsIdx**	**uimsbf**
WeightVal[0] = ((**SgnVal***2)-1);	**1**	**uimsbf**
} else {		
WeightIdx;	**8**	**uimsbf**
for (j=0; j< NumVvecIndices(k)[i]; ++j) {		
VvecIdx[j] = **bsVvecIdx** + 1;	**nbitsIdx**	**uimsbf**
if (j<8) {		
WeightVal[j] = ((**SgnVal***2)-1) * WeightValCdbk[WeightIdx][j];	**1**	**uimsbf**
} else {		
WeightVal[j] = ((**SgnVal***2)-1) * WeightValCdbk[WeightIdx][6+j%2];	**1**	**uimsbf**
}		
}		
}		
}		
else if (NbitsQ(k)[i] == 5) {		
for (m=0; m< VVecLengthUsed; ++m){		
aVal[i][m] = (**VecVal** / 128.0) – 1.0;	**8**	**uimsbf**
}		
else if(NbitsQ(k)[i] >= 6) {		
for (m=0; m< VVecLengthUsed; ++m){		
huffIdx = *huffSelect*(VVecCoeffIdUsed[m], PFlag[i], CbFlag[i]);		
cid = *huffDecode*(NbitsQ[i], huffIdx, **huffVal**);	**1..11**	**vlclbf**
aVal[i][m] = 0.0;		

Syntax	No. of bits	Mnemonic
if (cid > 0) {		
aVal[i][m] = sgn = (**sgnVal** * 2) - 1;	**1**	**bslbf**
if (cid > 1) {		
aVal[i][m] = sgn * (2.0^(cid -1) + **intAddVal**);	**cid-1**	**uimsbf**
}		
}		
}		
}		
}		
NOTE See subclause 12.4.1.11 for computation of VVecLength and nbitsIdx.		

Table 199 — Syntax of HOAPredictionInfo(DirSigChannelIds, NumOfDirSigs)

Syntax	No. of bits	Mnemonic
HOAPredictionInfo(DirSigChannelIds, NumOfDirSigs)		
{		
PredIdsBits = ceil(log2(NumOfDirSigs + 1));		
if(**PSPredictionActive**){	**1**	**bslbf**
NumActivePred = 0;		
if(**KindOfCodedPredIds**){	**1**	**bslbf**
NumActivePred = **NumActivePredIds** + 1;	**NumActivePredIdsBits**	**uimsbf**
i=0;		
while(i < NumActivePred){		
PredIds[i] = **PredIds**[i] + 1;	**ActivePredIdsBits**	**uimsbf**
i++;		
}		
}		
else{		
for (i=0; i<(HoaOrder +1)^2; i++) {		
if(**ActivePred[i]**) {	**1**	**bslbf**
NumActivePred ++;		
}		
}		
}		
NumOfGains=0;		
for (i=0; i<NumActivePred * MaxNoOfDirSigsForPrediction; i++) {		
if (**PredDirSigIds[i]** > 0) {	**PredIdsBits**	**uimsbf**
PredDirSigIds[i] = DirSigChannelIds[PredDirSigIds[i] - 1]**;**		
NumOfGains++;		
}		
}		
n=0;		
for (i=0; i< NumOfGains; i++) {		
if (PredDirSigIds[i]>0) {		
bsPredGains;	**NoOfBitsPerScalefactor**	**uimsbf**
PredGains[i] = bsPredGains – 2^(NoOfBitsPerScalefactor-1)		
}		

Syntax	No. of bits	Mnemonic
} } }		
NOTE lay is the index of the currently active HOA enhancement layer. In case of SingleLayer==1 set lay to zero.		

Table 200 — Syntax of HOADirectionalPredictionInfo()

Syntax	No. of bits	Mnemonic

HOADirectionalPredictionInfo()
{
if(**UseDirectionalPrediction**) {
	1	**bslbf**

if (!hoaIndependencyFlag**)** { **KeepPreviousGlobalPredDirsFlag;**	**1**	**bslbf**

}
else{
KeepPreviousGlobalPredDirsFlag = 0;
}		

if(!KeepPreviousGlobalPredDirsFlag) { NumOfGlobalPredDirs = **bsNumOfGlobalPredDirs** + 1;	**MaxNumOfPredDirsLog2**	**bslbf**

NumBitsForRelDirGridIdx = ceil(
log2(NumOfGlobalPredDirs));
for (idx=0; idx < NumOfGlobalPredDirs; idx++) {
GlobalPredDirsIds[idx];
	NumOfBitsPerDirIdx	**uimsbf**

}
}
else{
/* Keep values from previous HOADirectionalPredictionInfo
payload for NumOfGlobalPredDirs and
GlobalPredDirsIds. */
}		

SortedAddHoaCoeff = sort(AddHoaCoeffPerLayer[lay],
‘ascend’);
for (band = 0; band < NumOfPredSubbands; band++) {
for (dir = 0; dir < MaxNumOfPredDirsPerBand; dir++) {
for (hoaIdx = 0;
hoaIdx < MinNumOfCoeffsForAmbHOA;
hoaIdx++) {
DecodedMagDiff[band][dir][hoaIdx] = 0;
DecodedAngleDiff[band][dir][hoaIdx] = 0;
}
}
}		

for (band = 0; band < NumOfPredSubbands; band++) {
if (!hoaIndependencyFlag**)** {
KeepPreviousDirPredMatrixFlag[band]**;**
	1	**bslbf**

} else{		

Syntax	No. of bits	Mnemonic

KeepPreviousDirPredMatrixFlag[band] = 0;
}
if (!KeepPreviousDirPredMatrixFlag[band]) {
UseHuffmanCodingDiffMag;
	1	**bslbf**

if(band < FirstSBRSubbandIdx) { **UseHuffmanCodingDiffAngle;**	**1**	**bslbf**
for (dir = 0; dir < MaxNumOfPredDirsPerBand; dir++) { if (**DirIsActive**[band][dir]) {	**1**	**bslbf**
RelDirGridIdx;	**NumBitsForRelDirGridIdx**	**uimsbf**

PredDirGridIdx[band][dir] =
GlobalPredDirsIds[RelDirGridIdx];
for (hoaIdx = 0;
hoaIdx < MinNumOfCoeffsForAmbHOA;
hoaIdx++) {
readDirPredDiffValues (band, dir, hoaIdx,
UseHuffmanCodingDiffAbs,
UseHuffmanCodingDiffAngle,
FirstSBRSubbandIdx);
}
for (idx = 0;
idx < NumOfAddHoaChansPerLayer[lay];
idx++) {
readDirPredDiffValues (band, dir,
SortedAddHoaCoeff[idx] -1,
UseHuffmanCodingDiffAbs,
UseHuffmanCodingDiffAngle,
FirstSBRSubbandIdx);
}
}
}
}
}
}
}		

NOTE lay is the index of the currently active HOA enhancement layer. In case of SingleLayer==1 set lay to zero.		

Table 201 — Syntax for readDirPredDiffValues()

Syntax	No. of bits	Mnemonic

readDirPredDiffValues(band, dir, hoaIdx,
UseHuffmanCodingDiffAbs,
UseHuffmanCodingDiffAngle,
FirstSBRSubbandIdx)
{
if(UseHuffmanCodingDiffAbs) {
if(band < FirstSBRSubbandIdx) {
DecodedMagDiff[band][dir][hoaIdx] =
HuffmanMagDiffNoSbr[**HuffmanCodedMagDiff**];
else {
DecodedMagDiff[band][dir][hoaIdx] =
HuffmanMagDiffSbr[**HuffmanCodedRealMagDiff**];
}
if(DecodedMagDiff[band][dir][hoaIdx] ≤ -8){
	1..10
	
	1..12
		vlclbf
		
		vlclbf

Syntax	No. of bits	Mnemonic
DecodedMagDiff[band][dir][hoaIdx] = -8 - **runLengthCodedVal**;	1..	**uclbf**
{		
else if (DecodedMagDiff[band][dir][hoaIdx] ≥ 9**){**		
DecodedMagDiff[band][dir][hoaIdx] = 9 + **runLengthCodedVal**;	1..	**uclbf**
}		
}		
}		
else {		
DecodedMagDiff[band][dir][hoaIdx] = **CodedMagDiff** -7;	4	**uimsbf**
if(DecodedMagDiff[band][dir][hoaIdx] ≤ -7){		
DecodedMagDiff[band][dir][hoaIdx] = -7 - **runLengthCodedVal**;	1..	**uclbf**
{		
else if (DecodedMagDiff[band][dir][hoaIdx] ≥ 8**){**		
DecodedMagDiff[band][dir][hoaIdx] = 8 + **runLengthCodedVal**;	1..	**uclbf**
}		
}		
if(band < FirstSBRSubbandIdx) {		
if(UseHuffmanCodingDiffAngle) {		
DecodedAngleDiff[band][dir][hoaIdx] = DecTableAngleDiff[**HuffCodedAngleDiff**];	1..7	**vlclbf**
}		
else {		
DecodedAngleDiff[band][dir][hoaIdx] = DecTableAngleDiff[**CodedAngleDiff**];	4	**uimsbf**
}		
}		
}		

Table 202 — Syntax of HOAParInfo()

Syntax	No. of bits	Mnemonic
HOAParInfo()		
{		
if (**UsePar**) {	1	**bslbf**
for (band = 0; band < NumOfParSubbands; band++) {		
for(n = 0; n < MaxNumOfDecoSigs[band]; n++) {		
for(m = 0; m < MaxNumOfDecoSigs[band]; m++) {		
DecodedParMagDiff [band][n][m] = 0;		
DecodedParAngleDiff [band][n][m] = 0;		
}		
}		
if (!hoaIndependencyFlag**)** {		
KeepPreviousParMatrixFlag[band]**;**	1	**bslbf**
}		
else{		
KeepPreviousParMatrixFlag[band] = 0;		

Syntax	No. of bits	Mnemonic
}		
if (!KeepPreviousParMatrixFlag[band]) {		
ParDecorrSigsSelectionTableIdx[band]	**2**	**uimsbf**
if (UpmixHoaOrderPerParSubband[band] == 2) {		
NumOfDecorrSigsPerParSubband =		
NumOfDecorrSigsPerParSubbandTable[		
ParDecorrSigsSelectionTableIdx[band]]		
ParSelectedDecorrSigsIdxMatrix =		
ParSelectedDecorrSigsIdxMatrixTable[		
ParDecorrSigsSelectionTableIdx[band]]		
}		
else{		
NumOfDecorrSigsPerParSubband =		
NumOfDecorrSigsPerFirstOrderPar		
SubbandTable[		
ParDecorrSigsSelectionTableIdx[band]]		
ParSelectedDecorrSigsIdxMatrix =		
ParFirstOrderSelectedDecorr		
SigsIdxMatrixTable [		
ParDecorrSigsSelectionTableIdx[band]]		
}		
if (**UseReducedNoOfUpmixSigs**){	**1**	**bslbf**
for(n = 0;		
n < MaxNumOfDecoSigs[band]; n++) {		
UseParUpmixSig[band][n];	**1**	**bslbf**
}		
}		
else {		
for(n = 0;		
n < MaxNumOfDecoSigs[band]; n++) {		
UseParUpmixSig[band][n] = 1;		
}		
}		
UseParHuffmanCodingDiffAbs;	**1**	**bslbf**
if(!UseRealCoeffsPerParSubband[band]) {		
UseParHuffmanCodingDiffAngle;	**1**	**bslbf**
}		
for(n = 0;		
n < MaxNumOfDecoSigs[band];		
n++) {		
if(UseParUpmixSig[band][n]) {		
for(m = 0;		
m < NumOfDecorrSigsPerParSubband ;m++) {		
c = ParSelectedDecorrSigsIdxMatrix[n][m];		
readParDiffValues(band, n, c,		
UseParHuffmanCodingDiffAbs,		
UseParHuffmanCodingDiffAngle);		
}		
}		
}		
}		
}		
}		
}		

Table 203 — Syntax for readParDiffValues()

Syntax	No. of bits	Mnemonic
readParDiffValues (band, idx, decoIdx, UseParHuffmanCodingDiffAbs, UseParHuffmanCodingDiffAngle)		
{		
if(UseParHuffmanCodingDiffAbs) {		
if(UseRealCoeffsPerParSubband[band]) {		
DecodedParMagDiff[band][idx][decoIdx] = HuffmanMagDiffSbr[**HuffmanCodedParMagDiff**];	**1..10**	**vlclbf**
else {		
DecodedParMagDiff[band][idx][decoIdx] = HuffmanMagDiffNoSbr[**HuffmanCodedRealParMagDiff**];	**1..12**	**vlclbf**
}		
if (DecodedParMagDiff [band][dir][hoaIdx] ≤ -8**)**		
DecodedParMagDiff[band][idx][decoIdx] = -8 - **runLengthCodedVal**;	**1..**	**uclbf**
}		
else if (DecodedParMagDiff [band][dir][hoaIdx] ≥ 9)		
DecodedParMagDiff[band][idx][decoIdx] = 9 + **runLengthCodedVal**;	**1..**	**uclbf**
}		
else {		
DecodedParMagDiff[band][idx][decoIdx] = **CodedParMagDiff** - 7;	**4**	**uimsbf**
if (DecodedParMagDiff [band][dir][hoaIdx] ≤ -7**)**		
DecodedParMagDiff[band][idx][decoIdx] = -7 - **runLengthCodedVal**;	**1..**	**uclbf**
}		
else if (DecodedParMagDiff [band][dir][hoaIdx] ≥ 8)		
DecodedParMagDiff[band][idx][decoIdx] = 8 + **runLengthCodedVal**;	**1..**	**uclbf**
}		
}		
if(!UseRealCoeffsPerParSubband[band]){		
if(UseParHuffmanCodingDiffAngle) {		
DecodedParAngleDiff[band][idx][decoIdx] = DecTableAngleDiff[**HuffCodedParAngleDiff**];	**1..7**	**vlclbf**
}		
else {		
DecodedParAngleDiff[band][idx][decoIdx] = DecTableAngleDiff[**CodedParAngleDiff**];	**4**	**uimsbf**
}		
}		
}		

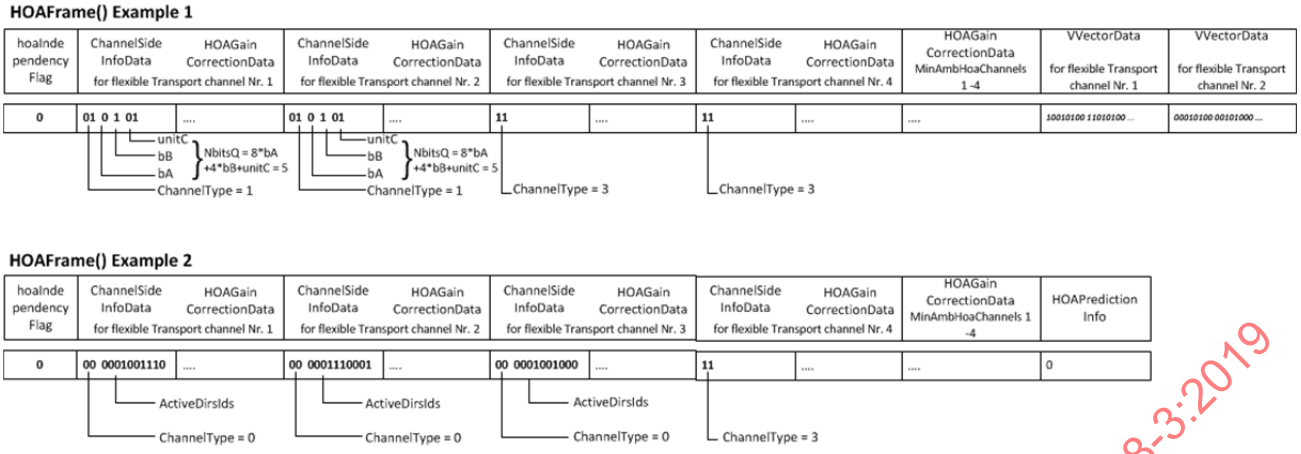


Figure 58 — Two examples for HOAFrame()

12.3 Data structure

12.3.1 Definitions of HOA Config

HOAConfig()	This element contains information about the audio content.
HoaOrder	This element determines the HOA order of the coded HOA signal.
NumOfHoaCoeffs	This element determines the number of HOA coefficients of the coded HOA representation, which is equal to the number of HOA coefficients to be reconstructed.
IsScreenRelative	This element indicates if the HOA representation shall be rendered with respect to the reproduction screen size as described in subclause 18.4.
UsesNfc	This element determines whether or not the HOA Near Field Compensation (NFC) has been applied to the coded signal.
NfcReferenceDistance	This element determines the radius in meter that has been used for the HOA NFC (interpreted as float in IEEE 754 format in little-endian).
HOADecoderConfig()	This element contains information to initialize the HOA spatial decoder.
MinAmbHoaOrder	This element determines the minimum HOA order used for the coding of the ambient HOA representation by MinAmbHoaOrder = **MinAmbHoaOrder** - 1. The value -1 indicates that the number of decorrelated ambiance coefficients is equal to zero. Thus the HOA representation is transmitted without a residual ambiance HOA representation of lower order and all transport channels have a flexible channel type. The value 6 is used to extend the HOA order signaling.
MinNumOfCoeffsForAmbHOA	This element determines the minimum number of ambient HOA coefficients.
NumOfAdditionalCoders	This element determines the number of additional transport channels used for coding the directional and/or additional HOA coefficients of the ambient component. These transport channels have a flexible ChannelType that is defined in Table 195.

SingleLayer This element indicates that the HOA signal is provided in a single layer according to Table 204.

Table 204 — SingleLayer definition

Value	Meaning
0	HOA signal is provided in multiple layers; enables the signaling of the distribution of the HOA transport channels into the different layers
1	HOA signal is provided in a single layer

codedLayerCh This element indicates for the first (i.e. base) layer the number of included transport signals, which is given by **codedLayerCh +** MinNumOfCoeffsForAmbHOA. For the higher (i.e. enhancement) layers, this element indicates the number of additional signals included into an enhancement layer compared to the next lower layer, which is given by **codedLayerCh** + 1.

HOALayerChBits This element indicates the number of bits for reading **codedLayerCh**.

NumLayers This element indicates (after the reading of the HOADecoderConfig()) the total number of layers within the bitstream.

NumHOAChannelsLayer This element is an array consisting of NumLayers elements, of which the i-th element indicates the number of transport signals included in all layers up to the i-th layer.

HOAEnhConfig() This payload contains the HOA enhancement configuration data of an HOA enhancement layer in the HOA layered coding mode.

HOALayerIdxBits This element indicates the number of bits used for signaling the element LayerIdx.

LayerIdx This element indicates the index of the HOA enhancement layer payload, where the index is defined in the range from zero to NumLayers-1.

HOADecoderEnhConfig() This payload contains all configuration elements for the HOA enhancement layer payloads.

CodedSpatialInterpolationTime This element determines the time of the spatio-temporal interpolation of the Vector-based directional signals as defined in Table 209.

SpatialInterpolationMethod This element determines the interpolation method applied during the spatio-temporal interpolation of the vector-based signals.

CodedVVecLength	This element indicates the length of the transmitted data vector used to synthesize the vector-based signals.
MaxGainCorrAmpExp	Gives the exponent of basis two of the maximum, accumulated amplification that has been used in the gain correction tool of the HOA spatial encoder for amplification of signals of the transport channel. This value is required to compute the number of bits for reading MaxGainCorrPrevAmpExp when the IndependencyFlag is set to true.
HOAFrameLengthIndicator	Indicates the frame length *L* (number of samples) of the HOA spatial decoding relative to the core coder frame length as defined in Table 208.
MaxHoaOrderToBeTransmitted	This element indicates the maximum HOA order of the additional ambient HOA coefficients to be transmitted.
MaxNumOfCoeffsToBeTransmitted	This element indicates the maximum number of HOA coefficients to be transmitted, computed depending on MaxHoaOrderToBeTransmitted.
MaxNumAddActiveAmbCoeffs	This element signals the maximum index for the signaling of additional ambient HOA coefficients.
VqConfBits	This element indicates the number of bits necessary to signal the element **NumVVecVqElementsBits**
NumVVecVqElementsBits	This element indicates the number of bits used to signal the element **NumVvecIndices** in ChannelSideInfoData()
UsePhaseShiftDecorr	This element signals the usage of the inverse phase based transform to recorrelate the ambience signals. A value of 1 means that it shall be used.
MaxNoOfDirSigsForPrediction	This element determines the number of directional signals used for the prediction of the predominant sound components.
NoOfBitsPerScalefactor	This element determines the number of bits for reading **PredGains[**n**]**.
PredSubbandsIdx	This element signals the table index for the sub-band configuration of the sub-band directional signals synthesis.
NumOfPredSubbands	This element contains the number of sub-band groups used for the sub-band directional signals synthesis.
NumOfPredSubbandsTable	This table contains the number of sub-band groups for each PredSubbandsIdx for the sub-band directional signals synthesis.

PredSubbandWidths[idx]	This array of NumOfPredSubbands elements contains the number of QMF sub-bands per sub-band group of the sub-band directional signals synthesis.
PredSubbandWidthTable	This table contains the bandwidths of the sub-band groups for each table index PredSubbandsIdx for the sub-band directional signals synthesis.

Table 205 — Directional prediction subbands table

PredSubbandsIdx	NumOfPredSubbandsTable	PredSubbandWidthTable
0	0	[]
1	10	[1, 1, 1, 2, 2, 2, 3, 6, 11, 35]
2	20	[1,1,1,1,1,1,1,1,1,1,2,2,2,2,4,5,7,11, 18]

getSubbandWidths()	This function reads a flexible sub-band group configuration.
FirstSBRSubbandIdxBits	This element determines the number of bits for reading **FirstSBRSubbandIdx**.
FirstSBRSubbandIdx	Indicates the first sub-band group of the directional signals synthesis where the samples are reconstructed by the SBR tool. To deactivate the special SBR processing in the sub-band directional signals synthesis set the value of this element to NumOfPredSubbands + 1.
MaxNumOfPredDirsLog2	This element signals the logarithm to the base of two from the maximum number of signals that are predicted in the sub-band directional signals synthesis.
MaxNumOfPredDirs	This element signals the maximum number of signals that are predicted in the sub-band directional signals synthesis.
MaxNumOfPredDirsPerBand	This element contains the maximum number of predicted signals per sub-band group of the sub-band directional signals synthesis.
DirGridTableIdx	This index determines the grid of potential directions of directional sub-band signals created at the sub-band directional signals synthesis.
NumOfBitsPerDirIdx	The element determines the number of bits for signaling the index of a virtual loudspeaker position index.
NumOfGridPointsTable	This table determines the number of potential directions of directional sub-band signals created at the sub-band directional signals synthesis.
NumOfBitsPerDirIdxTable	This table determines the number of bits used to code each single potential direction of a directional sub-band signal created at the sub-band directional signals synthesis.

Table 206— Quantized direction index table

DirGridTableIdx	NumOfGridPointsTable	NumOfBitsPerDirIdxTable
0	256	8
1	484	9
2	900	10
3	reserved	reserved

ParSubbandTableIdx This element signals the table index for the sub-band configuration of the parametric ambience replication decoder.

NumOfParSubbands This element signals the number of sub-band groups used by the parametric ambience replication decoder.

NumOfParSubbandsTable This table contains the number of sub-band groups for each PAR sub-band group table index **ParSubbandTableIdx**.

ParSubbandWidths This array of NumOfParSubbands elements contains the number of QMF sub-bands per sub-band group of the Parametric Ambience Replication decoder.

ParSubbandWidthTable This table contains the bandwidths of the sub-band groups for each table index ParSubbandTableIdx for the PAR decoder.

Table 207— PAR subbands table

ParSubbandTableIdx	NumOfParSubbandsTable	ParSubbandWidthTable
0	0	[]
1	4	[1, 1, 22, 40]
2	8	[1, 1, 1, 2, 2, 5, 10, 42]

LastFirstOrderSubbandIdxBits This element determines the number of bits required for reading LastFirstOrderSubbandIdx.

LastFirstOrderSubbandIdx This element indicates the index of the last PAR sub-band group that uses an HOA order of one.

UseRealCoeffsPerParSubband[idx] This Boolean array indicates for each PAR sub-band group if the mixing matrix consists of real-valued non-negative (true) or complex valued elements matrix (false).

UpmixHoaOrderPerParSubband[idx] This array contains the HOA order used in each PAR sub-band group.

MaxNumOfDecoSigs[idx] This array contains the maximum number of de-correlated signals per PAR sub-band group.

AmbAsignmBits This element determines the number of bits that are required for reading **CodedAmbCoeffIdx[i]**.

ActivePredIdsBits This element determines the number of bits for reading **ActivePredIds**.

NumActivePredIdsBits This element determines the number of bits for reading **NumActivePredIds**.

GainCorrPrevAmpExtBits This element determines the number of bits for reading MaxGainCorrPrevAmpExp.

12.3.2 Syntax of getSubbandBandwidths()

This function reads the bandwidth for NumberOfSubbands sub-band groups.

CodedBwFirstBand	This element signals the bandwidth of the first sub-band group by the number of QMF sub-bands.
NumberOfSubbands	This element contains the total number of sub-band groups.
bw[idx]	This array holds the bandwidth of each sub-band group in the number of QMF sub-bands.
bw_diff	This element contains the differentially coded bandwidth of a sub-band group.

12.3.3 Definitions of HOA payload

12.3.3.1 HOAFrame()

The HOAFrame() holds the information that is required to decode the L samples of an HOA frame of order HoaOrder.

HOAFrame()	This block of data contains the data required to decode one frame of L samples of the coded HOA representation.
NumOfDirSigs	This element determines the number of active directional signals in the current HOAFrame().
NumOfVecSigs	This element determines the number of active vector-based signals in the current HOAFrame().
NumOfAddHoaChans	This element determines the total number of additional ambient HOA channels in the current HOAFrame().
NumOfDirSigsPerLayer[lay]	This element determines the number of active directional signals in the current HOAFrame() that are actually used in the HOA enhancement layer *lay*.
NumOfAddHoaChansPerLayer[lay]	This element signals the total number of additional ambient HOA coefficients actually used in the HOA enhancement layer *lay*.
NumOfContAddHoaChans[lay]	This element determines the number of continuous additional ambient HOA coefficients in the current HOAFrame() that are actually used in the HOA enhancement layer *lay*.
NumOfNewAddHoaChans[lay]	This element determines the number of newly introduced additional ambient HOA channels in the current HOAFrame() that are actually used in the HOA enhancement layer *lay*.

hoaIndependencyFlag	This flag signals that the current frame is an independent frame that can be decoded without having knowledge about the previous frame. Otherwise this flag is equal to the usacIndependencyFlag.
ChannelSideInfoData(i)	This payload holds the side information for the i-th of the NumOfAdditionalCoders channels of flexible ChannelType.
HOAGainCorrectionData(i)	This payload contains data for the inverse gain correction of channel i.
DirSigChannelIds[NumOfDirSigs]	This element stores the channel index of each active directional signal of the current frame.
VecSigChannelIds[NumOfVecSigs]	This element stores the channel index of each active vector-based signal of the current frame.
VecSigLayerIdx[i]	This element indicates for the i-th vector-based channel element of the current frame the index of the enhancement layer that transmits the corresponding vector-based signal.
ContAddHoaCoeff [lay][NumOfContAddHoaChans]	This 2D array contains the HOA coefficient indices for each continuous additional ambient HOA coefficient actually used in the HOA enhancement layer *lay*. For these HOA coefficients the AmbCoeffTransitionState word is of value 0 in the current frame.
NewAddHoaCoeff [lay][NumOfContAddHoaChans]	This 2D array contains the HOA coefficient indices for each additional ambient HOA coefficient, which was not present in the preceding frame used in the HOA enhancement layer *lay*.
AddHoaCoeffPerLayer[lay]	This array contains the HOA coefficient indices for each additional ambient HOA coefficient actually used in the HOA enhancement layer *lay*.
AddHoaCoeff[lay]	This element stores the HOA coefficient number of each additional ambient HOA channel in the current frame.
HOAEnhFrame()	This payload contains the frame data for the HOA enhancement payloads HOAPredictionInfo(), HOADirectionalPredictionInfo() and HOAParInfo().

12.3.3.2 HOAEnhFrame()

HOAPredictionInfo()	This payload contains data for the prediction of dominant sound sources from the active directional signals of the current frame.
HOADirectionalPredictionInfo	This payload contains data for the sub-band directional

	signals synthesis.
HOAParInfo	This payload contains data for the parametric ambience replication.

12.3.3.3 ChannelSideInfoData(i)

This payload holds the side information for the i-th channel. The size and the data of the payload depend on the type of the channel.

ChannelType[i**]**	This element stores the type of the i-th channel which is defined in Table 195.
ActiveDirsIds[i**]**	This element indicates the direction of the active directional signal using an index of the 900 predefined, uniformly distributed points from Annex F.9. The code word 0 is used for signalling the end of a directional signal.
NewChannelTypeOne[i]	This flag indicates if in the previous frame (k-1) the transport channel was not initialized as a vector-based signal.
PFlag[i]	The prediction flag used for the Huffman decoding of the scalar quantized V-vector associated with the vector-based signal of the i-th channel.
CbFlag[i]	The codebook flag used for the Huffman decoding of the scalar quantized V-vector associated with the Vector-based signal of the i-th channel.
CodebkIdx[i]	Signals the specific codebook used to dequantize the vector-quantized V-vector associated with the vector-based signal of the i-th channel.
NumVvecIndices(k)[i]	The number of vectors used to dequantize a vector-quantized V-vector.
NbitsQ[i]	The NbitsQ[i] value determines the decoding method of the V-Vector associated with the Vector-based signal of the i-th channel. An NbitsQ[i] value of 4 determines the decoding of a vector-quantized V-vector. The value 5 determines the decoding of a uniform 8bit scalar quantized V-vector. If the value is greater than 5, Huffman decoding of the V-vector is determined by using the Huffman table index as signalled with the NbitsQ[i] value (the Huffman tables are provided in Annex F.15 to F.24). The two MSBs 00 signal the reuse of the values NbitsQ[i], PFlag[i], CbFlag[i], and CodebkIdx[i] from the previous frame (k-1).
bA, bB	The msb (bA) and second msb (bB) of the NbitsQ[i] field.
uintC	The code word of the remaining two bits of the NbitsQ[i] field.
AddAmbHoaInfoChannel(i)	This payload holds the information for additional ambient HOA coefficients.

12.3.3.4 AddAmbHoaInfoChannel(i)

This payload contains the information required to add an additional ambient HOA coefficient to the reconstructed HOA representation.

AmbCoeffTransitionState	This decoder-internal variable tracks the state of the life-cycle of an additional ambient HOA coefficient. Those states are fade-in, continous state, and fade-out. The AmbCoeffIdxTransition signals a change of the state in the bitstream. When an additional ambient HOA coefficient is faded in, the CodedAmbCoeffIdx word is sent to signal the new AmbCoeffIdx. In all other states, the AmbCoeffIdx of the previous frame is used.
AmbCoeffIdxTransition	This element indicates that in this frame an additional ambient HOA coefficient is either being faded in or faded out. This flag will update the decoder-internal AmbCoeffTransitionState variable for this transport channel accordingly (see Table 196).
CodedAmbCoeffIdx	This element reads the coded index of the additional ambient HOA coefficient.
AmbCoeffIdx[i]	This element determines the index of the HOA signal where channel i contributes to the reconstructed HOA representation.

12.3.3.5 HOAGainCorrectionData(i)

This structure comprises the parameters for the inverse gain correction of the spatial HOA decoding tool for the i-th transport channel.

HOAGainCorrectionData()	This payload comprises the parameters for the inverse gain correction.
GainCorrPrevAmpExp[i]	This element gives the amplification as an exponent to the basis of two that has been applied to the signal of the transport channel i in the previous frame. It is only required to send if the IndependencyFlag is set to true for starting decoding at the current frame. The value range of the element is from –ceil(log2(1.5*NumOfHoaCoeffs)) to MaxGainCorrAmpExp, where the lower bound is the maximal amplitude of any transport channel signal. This bound is computed from the assumption that all spatial domain signals of the encoded HOA representation have absolute amplitudes less than one. The spatial domain signals are computed by the multiplication of the HOA representation with the inverse of the modematrix created from the uniformly distributed positions from F.2 to F.11 corresponding to the **HoaOrder**.
CodedGainCorrectionExp[i**][**n**]**	The index n addresses the bits of the run length code to determine the exponent used for the inverse gain correction of the i-th channels.
GainCorrectionException[i**]**	indicates the gain correction exception state for each of the i-th channels.

12.3.3.6 VVectorData(VecSigChannelIds(i))

This structure contains the coded V-vector data used for the vector-based signal synthesis. The subfunctions used for the decoding are specified in subclause 12.4.1.11.

VVec(k)[i]	This is the V-vector for the k-th HOAframe() for the i-th channel.
VVecLength	This variable indicates the number of vector elements to read out.
VVecCoeffId	This vector contains the indices of the transmitted V-vector coefficients.
VecVal	An integer value between 0 and 255.
aVal	A temporary variable used during decoding of the VVectorData.
huffVal	A Huffman code word, to be Huffman-decoded.
sgnVal	This is the coded sign value used during decoding.
intAddVal	This is additional integer value used during decoding.
WeightIdx	The index in WeightValCdbk used to dequantize a vector-quantized V-vector.
nBitsW	Field size for reading **WeightIdx** to decode a vector-quantized V-vector.
WeightValCdbk	Codebook which contains a vector of positive real-valued weighting coefficients. Only necessary if NumVvecIndices is > 1. The WeightValCdbk with 256 entries is provided in Annex F.14.
VvecIdx	An index for VecDict, used to dequantize a vector-quantized V-vector.
nbitsIdx	Field size for reading **VvecIdx** to decode a vector-quantized V-vector.
WeightVal	A real-valued weighting coefficient to decode a vector-quantized V-vector.

12.3.3.7 HOAPredictionInfo(DirSigChannelIds, NumOfDirSigs)

The data of this payload provides information for the predominant sound synthesis in the HOA spatial decoding tool. It includes parameters for the prediction of the predominant sound component from the active directional signals.

PredIdsBits	This helper variable holds the number of bits for reading **PredDirSigIds**[]. The number of bits is adapted to the currently active number of directional signals NumOfDirSigs, which is determined in HOAFrame().
PSPredictionActive	This element determines whether or not the spatial prediction tool contributes to the decoded HOA representation.
KindOfCodedPredIds	This element indicated the method for reading **ActivePred**[].

NumActivePredIds	This element indicates the number of active prediction indices from the NumOfHoaCoeffs uniformly distributed position indices from F.2 to F.11. The table is selected according to HoaOrder.
NumActivePred	This element determines the number of active prediction indices.
PredIds[idx]	This array indicates the active prediction indices from F.2 to F.11. The table is selected according to HoaOrder.
ActivePred[idx]	This element indicates whether or not ('1' or '0') the prediction is computed for each of the NumOfHoaCoeffs uniformly distributed positions from F.2 to F.11. The table is selected according to HoaOrder.
PredDirSigIds[n]	This array indicates the index of the transport channel of the active directional signals for each active prediction.
NumOfGains	The variable indicates the number of prediction gains provided by the bitstream. Only prediction gains that are greater than zero are provided. The maximal number of prediction gains is equal to NumActivePred times MaxNoOfDirSigsForPrediction.
PredGains[n]	This array holds the NumOfGains prediction gains.

12.3.3.8 HOADirectionalPredictionInfo

This payload contains data for the directional signals synthesis.

UseDirectionalPrediction	This flag indicates if the directional signals synthesis is performed in the current frame.
KeepPreviousGlobalPredDirsFlag	This flag indicates that the elements NumOfGlobalPredDirs, NumBitsForRelDirGridIdx and GlobalPredDirsIds read from the previous HOADirectionalPredictionInfo payload are used.
NumOfGlobalPredDirs	This element determines the available number of global, broadband directions.
NumBitsForRelDirGridIdx	This element indicates the number of bit required for reading **GlobalPredDirsIds.**
GlobalPredDirsIds	This array contains the indices for the NumOfGlobalPredDirs directions, where each index belongs to a direction from the grid of positions selected by **DirGridTableIdx.**
SortedAddHoaCoeff	This array contains the HOA coefficients indices from AddHoaCoeff sorted ascendingly.
KeepPreviousDirPredMatrixFlag[band]	This Boolean is true if the signal prediction matrix is kept from the previous frame and false if the mixing matrix is update by new magnitude and angle differences.

UseHuffmanCodingDiffMag	This Boolean element is true if the magnitude differences are encoded by Huffman coding.
UseHuffmanCodingDiffAngle	This Boolean element is true if the angle differences are encoded by Huffman coding.
DirIsActive	This Boolean element is true if the current direction of the current sub-band group has been transmitted.
RelDirGridIdx	This element holds the relative index of the current direction index stored in GlobalPredDirsIds.
PredDirGridIdx[band][dir]	This 2D array contains the direction indices belonging to the direction grid selected by **DirGridTableIdx** for all active directions of each sub-band group.
readDirPredDiffValues()	This function reads the coded magnitude and angle differences.

12.3.3.9 Syntax for readDirPredDiffValues()

This function reads the magnitude and phase differences for the directional signal synthesis.

HuffCodedMagDiff	This element signals the Huffman coded magnitude difference using Table F.34.
DecodedMagDiff[band][dir][idx]	This 3D array holds the decoded magnitude difference from Table F.35 for the prediction of each directional signal with the index dir of a sub-band group with the index band from the HOA coefficient with the index idx.
runLengthCodedVal	This run length code determines magnitude difference if a corresponding escape word has been signalled.
HuffCodedAngleDiff	This element signals the Huffman coded angle difference using Table F.36.
DecodedAngleDiff[band][dir][hoaIdx]	This 3D array holds the decoded angle difference from Table F.36 for the prediction of each directional signal with the index dir of a sub-band group with the index band from the HOA coefficient with the index idx.
HuffCodedSbrMagDiff	This element signals the Huffman coded magnitude difference using Table F.35.
CodedMagDiff	This element holds the magnitude difference index quantized with 4 plain bits.
CodedAngleDiff	This element holds the angle difference index quantized with 4 plain bits.

12.3.3.10 HOAParInfo

This payload contains data for the PAR decoder.

UsePar	This Boolean is true if the PAR decoding is performed in the current frame.
KeepPreviousParMatrixFlag[idx]	This Boolean is true if the mixing matrix is kept from the previous frame and false if the mixing matrix is update by new magnitude and angle differences.
ParDecorrSigsSelectionTableIdx[idx]	This table index determines the number and the corresponding indices of the de-correlated signals that are used to create one upmix signal.
NumOfDecorrSigsPerParSubband	This element indicates the number of de-correlated signals that are used to create one upmix signal, which is obtained for UpmixHoaOrderPerParSubband[idx] of two from F.40 and for UpmixHoaOrderPerParSubband[idx] of one from Table F.40 exploiting the index ParDecorrSigsSelectionTableIdx[idx].
ParSelectedDecorrSigsIdxMatrix[sig][idx]	This matrix contains the NumOfDecorrSigsPerParSubband indices of the de-correlated signals that are used to create the upmix signal with the index sig, where the matrix is obtained for UpmixHoaOrderPerParSubband[idx] of two from Table F.40 and for UpmixHoaOrderPerParSubband[idx] of one from Table F.43 exploiting the index ParDecorrSigsSelectionTableIdx[idx].
UseReducedNoOfUpmixSigs	This element signals that not all upmix signals are created.
UseParUpmixSig[idx][n]	For each PAR sub-band group of index idx the elements of the Boolean matrix are true if the upmix signal with an index n is created or false if it is not created.
UseParHuffmanCodingDiffAbs	This Boolean indicates by true that the magnitude differences are Huffman coded.
UseParHuffmanCodingDiffAngle	This Boolean indicates by true that the angle differences are Huffman coded. The index is equal to the index of the de-correlated signal used to create the current upmix signal. It is obtained from F.40 for the current value of **ParDecorrSigsSelectionTableIdx**.

12.3.3.11readParDiffValues()

This function reads magnitude and phase differences of the mixing matrices that are used by the PAR decoder.

HuffCodedParMagDiff	This element signals the Huffman coded magnitude difference using Table F.37.

runLengthCodedVal	This run length code determines magnitude difference if a corresponding escape word has been signalled.
DecodedParMagDiff[band][idx][decoIdx]	This 3D array holds the decoded magnitude differences for the prediction of each virtual loudspeaker signal with the index idx of a sub-band group with the index band from the de-correlated signal with the index decoIdx.
HuffCodedParAngleDiff	This element signals the Huffman coded angle difference using Table F.39.
DecodedParAngleDiff[band][idx][decoIdx]	This 3D array holds the decoded angle differences for the prediction of each virtual loudspeaker signal with the index idx of a sub-band group with the index band from the de-correlated signal with the index idx.
HuffmanCodedRealParMagDiff	This element signals the Huffman coded magnitude difference using Table F.38.
CodedParMagDiff	This element holds the coded magnitude difference index quantized with 4 plain bits.
CodedParAngleDiff	This element holds the coded angle difference index quantized with 4 plain bits.

12.4 HOA tool description

12.4.1 HOA frame converter

12.4.1.1 General

The HOA frame converter, shown in Figure 57, converts the parameters from the HOAFrame() payload and from the HOAConfig() payload to the parameters required for the HOA decoding tools.

12.4.1.2 Global parameter

The HOA frame converter defines the following variables.

I =NumHOATransportChannels	Number of transport signals used as input for the spatial decoding tool
$i \in \{1, \ldots, I\}$	Index for the transport signals
N = HoaOrder	Ambisonics order of the coded HOA signal
$O = (N+1)^2$	Number of HOA coefficients
N_{MIN} = **MinAmbHoaOrder**	Minimum order of the transmitted ambient HOA representation
$O_{\mathrm{MIN}} = (N_{\mathrm{MIN}}+1)^2$	Minimum number of ambient HOA coefficients

d	Direction index
D_{PRED} = MaxNoOfDirSigsForPrediction	Maximum number of directional signals used for the prediction of dominant sound sources
D_{SB} = MaxNumOfPredDirsPerBand	Maximum number of directions per sub-band for sub-band directional signals synthesis
$\mathrm{D}_{\mathrm{MAX}}$ = MaxNumOfPredDirs	Maximum number of different directions over all sub-bands for sub-band directional signals synthesis
D = NumOfAdditionalCoders	Number of channels with flexible channel types
B_{SC} =NoOfBitsPerScaleFactor	Number of bits per scale factor
$Q = 900$	Number of direction indices
$q \in \{1, \ldots, Q\}$	Quantization index of a direction
L	Frame length in samples for the HOA spatial decoding. The frame length L depends on outputFrameLength, which is the number of output samples of the core decoder for decoding one frame (it can be 768, 1024, 2048 or 4096 output samples long). The Output frame length can be found in ISO/IEC 23003-3:2012, Table 70. Furthermore, L depends on the value of **HOAFrameLengthIndicator**. The definition of L can be found in Table 208.

Table 208 — Value of frame length L depending on HOAFrameLengthIndicator and outputFrameLength

outputFrameLength	**HOAFrameLengthIndicator**			
	00_2	**01_2**	**10_2**	**11_2**
768	768	768	768	reserved
1 024	1 024	1 024	1 024	
2 048	2 048	1 024	1 024	
4 096	4 096	2 048	1 024	

l	Sample index
k	Frame index
$r_{\max}$	Determines the maximum loudspeaker distance of the listening setup. In case the actual loudspeaker distances are unknown (hasLoudspeakerDistance == 0) $r_{\max}$ is set to infinity ($r_{\max}=\infty$).
E = **CodedVVecLength**	Coded V-vector length
VecDict[cdbLen]	Codebook with *cdbLen* codebook entries containing vectors of HOA expansion coefficients, used to decode a vector-quantized V-vector.

12.4.1.2.1 Upper and lower bounds of sub-band groups for sub-band directional signals synthesis

$B = \text{NumOfPredSubbands};$

$\mathcal{L}(1) = 1;$

```
for (b=0; b < NumOfPredSubbands − 1; b++)
{
        U(b + 1) = L(b) + PredSubbandWidths[b] − 1;
        L(b + 1) = U(b + 1) + 1;
}
```

$\mathcal{U}(B) = \mathcal{L}(B-1) + \text{PredSubbandWidths}[B-1] - 1;$

12.4.1.2.2 Upper and lower bounds of sub-band groups for PAR

$G = \text{NumOfParSubbands};$

$\mathcal{L}_{\text{PAR}}(1) = 1;$

```
for (g=0; g < NumOfParSubbands − 1; g++)
{
        U_PAR(g + 1) = L_PAR(g) + ParSubbandWidths [g] − 1;
        L_PAR(g + 1) = U_PAR(g + 1) + 1;
}
```

$\mathcal{U}_{\text{PAR}}(G) = \mathcal{L}_{\text{PAR}}(G-1) + \text{ParSubbandWidths}[G-1] - 1;$

12.4.1.2.3 Orders $N_{\text{PAR}}(g)$ for each sub-band group $g = 1, \ldots, G$ for PAR

```
for (g=0; g < NumOfParSubbands; g++)
{
    N_PAR(g + 1) = UpmixHoaOrderPerParSubband[g];
}
```

12.4.1.3 Frame and user dependent parameters

$M_{\text{LAY}}(k)$ Number of all actually used layers for the k-th frame (to be specified) at the decoder side. Note that in the case of layered coding (indicated by SingleLayer==0) this number shall be less or equal to the total number of layers present in the bitstream, i.e. $M_{\text{LAY}} \leq$ NumLayers. In the case of single-layered coding (indicated by SingleLayer==1) M_{LAY} is set to one.

Dependent on the choice of $M_{\text{LAY}}(k)$ the number $I_{\text{ADD,LAY}}(k)$ of additional transport channels actually used for spatial HOA decoding (i.e. additional to the O_{MIN} channels that are implicitly always used) is computed as follows:

if(**SingleLayer** | (!**SingleLayer** & $M_{\mathrm{LAY}}(k)$ == NumLayers))
{
    $I_{\mathrm{ADD,LAY}}(k)$ = NumOfAdditionalCoders;
}
else
{
    $I_{\mathrm{ADD,LAY}}(k)$ = NumHOAChannelsLayer[0] - MinNumOfCoeffsForAmbHOA;
    for (m=1; m < M_{LAY} ; m++){
        $\mathrm{I}_{\mathrm{ADD,LAY}}(\mathrm{k})$ = NumHOAChannelsLayer[$\mathrm{M}_{\mathrm{LAY}}(\mathrm{k}) - 1$] - MinNumOfCoeffsForAmbHOA;
    }
}

The number is required to extract from the total side information the part that is relevant for the actually used transport signals. For this reason, in the following, it is used for the conversion of the bitstream parameters to the parameters used in the description of the actual spatial HOA decoding in subclause 12.4.2.

12.4.1.4 Helper functions

```
bool isMemberOf(val, array, arraySize)
{
    idx = 0;
    for( idx = 0; idx < arraySize; idx++ )
    {
        if(array[idx] == val)
            return true;
    }
    return false;
}
```

12.4.1.5 Prediction parameters $\boldsymbol{\zeta}(\boldsymbol{k})$

The set of predictions parameters consists of three arrays $\boldsymbol{\zeta}(\boldsymbol{k}) = \{\boldsymbol{p}_{\mathrm{TYPE}}(k), \boldsymbol{P}_{\mathrm{IND}}(k), \boldsymbol{P}_{\mathrm{Q,F}}(k)\}$. The array $\boldsymbol{p}_{\mathrm{TYPE}}(k)$ consists of O entries, $\boldsymbol{P}_{\mathrm{IND}}(k)$ and $\boldsymbol{P}_{\mathrm{Q,F}}(k)$ are D_{PRED}xO matrices.

for (q=1; q ≤ O; q++) {
    $\boldsymbol{p}_{\mathrm{TYPE}}(k)[q] = 0$;
    for (d=1; d ≤ D_{PRED}; d++) {
        $\boldsymbol{P}_{\mathrm{IND}}(k)$[d][q] = 0;
        $\boldsymbol{P}_{\mathrm{Q,F}}(k)$[d][q] = 0;
    }
}
if (**PSPredictionActive**) {
    if (!**KindOfCodedPredIds**) {
        i=0;
        for (q=0; q < O; q++) {
            if (**ActivePred**[q]) {
                $\boldsymbol{p}_{\mathrm{TYPE}}(k)$[q+1] = 1;
                i++;
            }
        }
    }

```
        else {
            for (i=0; i < NumActivePred; i++) {
                p_TYPE(k)[PredIds[i]] = 1;
            }
        }
}

i=0;
j=0;
for (q=1; q ≤ O; q++) {
        for (d=1; d ≤ D_PRED; d++) {
            if (p_TYPE(k)[i]>0) {
                P_IND(k)[d][q] = PredDirSigIds[i];
                i++;
                if (PredDirSigIds[i] != 0) {
                    P_Q,F(k)[d][q] = PredGains[j];
                    j++;
                }
            }
        }
}
```

12.4.1.6 Assigment vector $v_{AMB,ASSIGN}(k)$

```
for (i=0; i <I_ADD,LAY (k); i++){
        if(ChannelType[i]==2){
            v_AMB,ASSIGN(k)[i+1] = AmbCoeffIdx[i];
        }
        else{
            v_AMB,ASSIGN(k)[i+1]=0;
        }
}
nIdx = 1;
for (i= I_ADD,LAY (k); i < NumHOATransportChannels; i++){
        v_AMB,ASSIGN(k)[i+1]=nIdx;
        nIdx++;
}
```

12.4.1.7 Tuple set $\mathcal{M}_{DIR}(k)$

```
M_DIR(k)=∅;
for (i=0; i < I_ADD,LAY (k); i++){
        if(ChannelType[i]==0){
            M_DIR(k)=M_DIR(k) ∪ (i + 1, ActiveDirsIds[i]);
        }
}
```

12.4.1.8 The sets $\mathcal{I}_E(k), \mathcal{I}_D(k)$ and $\mathcal{I}_U(k)$

$\mathcal{I}_E(k) = \emptyset;$
$\mathcal{I}_D(k) = \emptyset;$
$\mathcal{I}_U(k) = \{1, \ldots O_{MIN}\};$

```
for (i=0; i < I_ADD,LAY (k); i++) {
    switch(AmbCoeffTransitionState[i]) {
        case 0:
        {
                    I_U(k) = I_U(k) ∪ {AmbCoeffIdx[i]};
                    break;
        }
        case 1:
        {
                    I_E(k) = I_E(k) ∪ {AmbCoeffIdx[i]} ;
                    break;
        }
        case 2:
        {
                    I_D(k) = I_D(k) ∪ {AmbCoeffIdx[i]} ;
        }

    }
}
```

12.4.1.9 Gain correction exponents $e_i(k)$

```
RunIdx = 0;
for (i = 0; i < I; i++) {
    if (IndependencyFlag) {
        g_IGC,i(k − 1) = pow( 2, GainCorrPrevAmpExp[i] );
       }
    TmpCodedExp = 0;
    FoundOne = false;

    while (!FoundOne) {
        FoundOne = CodedGainCorrectionExp[i][RunIdx];
        TmpCodedExp++;
        RunIdx++;
    }

    switch (TmpCodedExp) {
        case 1:
        {
            e_i+1(k) = 0;
            break;
        }
        case 2:
        {
            e_i+1(k) = -1;
            break;
        }
        default:
        {
            e_i+1(k) = TmpCodedExp-2;
        }
    }
}
```

12.4.1.10 Gain correction exception flag $\beta_i(k)$

```
for (i = 0; i < I; i++) {
    β_{i+1}(k) = GainCorrectionException[i];
}
```

12.4.1.11 Decoding of V-vector

12.4.1.11.1 General

A V-vector represents the spatial distribution of the soundfield for a particular vector-based predominant sound and is decomposed from the HOA frame using a vector-based synthesis approach. Various examples of the V-vectors are illustrated in the following Figure 59.

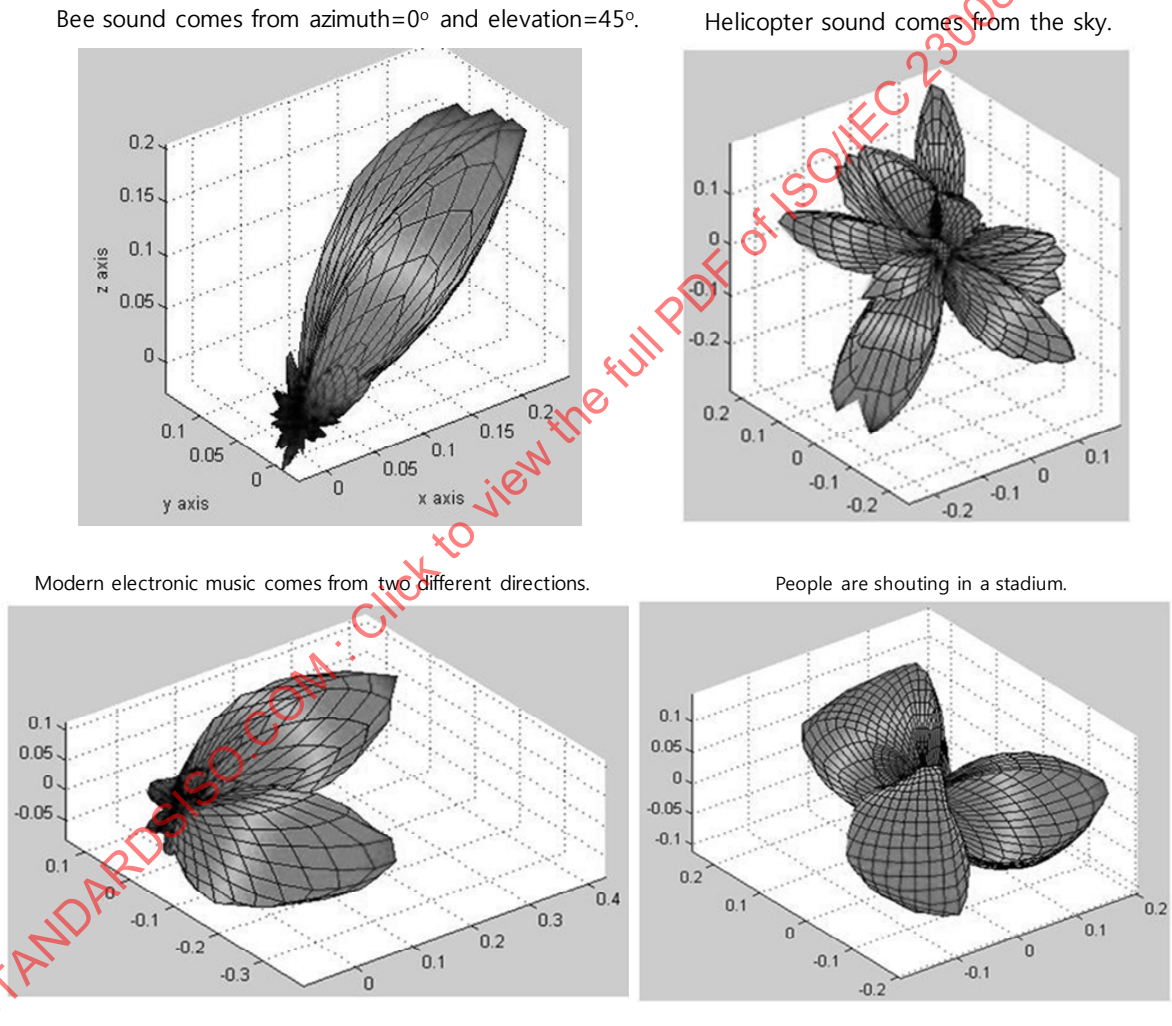


Figure 59 — Illustrations of V-vectors

These vectors were either scalar-quantized or vector-quantized. If the vectors are scalar quantized, an optional prediction from the V-vector of frame k-1 as well as efficient Huffman coding can be employed as shown in the following Figure 60.

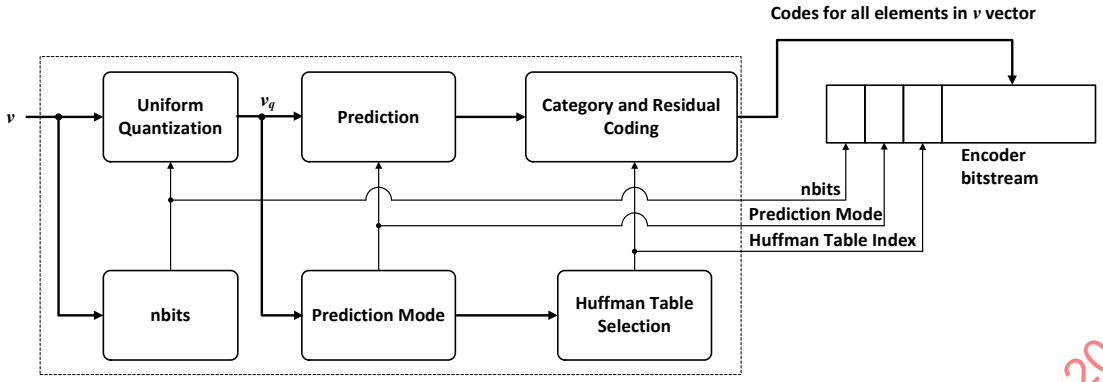


Figure 60 — Coding of V-vector

To decode a scalar-quantized V-vector, the nbits, prediction mode and Huffman Table index are obtained from the bitstream as specified in subclauses 12.4.1.11.2, 12.4.1.11.3 and 12.4.1.11.4. The bitstream is formulated to specify the compressed version of this spatial component (V-vector) along with the *nbits*, *prediction mode* and *Huffman table index*. The compressed V-vector is Huffman decoded, predicted and dequantized to obtain a reconstructed V-vector, where the V-vectors for a k^{th} frame are denoted as as $\boldsymbol{v}(k)$as part of the tuple set $\mathcal{M}_{\mathrm{VEC}}(k)$.. In the text below, *nbits* denotes the quantization step size. Portions of *nbits* are denoted as *unitC*, *bB* and *bA* below. The prediction mode is denoted as *PFlag* and the Huffman table index is denoted as *huffIdx*. The Huffman table index is not explicitly signalled in the bitstream but is derived as specified in subclause 12.4.1.11.4 based on a *CbFlag* specified in the bitstream.

When decoding vector-quantized V-vectors, a V-vector is represented by a weighted summation of pre-defined vectors as given by:

$$\hat{V} \approx \sum_{i=1}^{I} \omega_i \Omega_i$$

where ω_i and Ω_i are an i-th weighting value and the corresponding vectors, respectively. Once the V-vector is reconstructed, a rescaling of the V-vector is performed, so that the Euclidian norm is of size N+1.

An example of the vector-quantized V-vector decoding is illustrated in Figure 61. As shown in Figure 61 (a), an original V-vector is represented by a mixture of multiple vectors. Thus, the original V-vector is estimated by a weighted sum as shown in Figure 61 (b) where a weighting vector is shown in Figure 61 (e). Figure 61 (c) and (f) illustrate the cases that only I_S ($I_S \leq I$) highest weighting values are selected. Vector quantization is performed for the selected weighting values (Figure 61 (g)). The codebook indices and weights are signalled to the decoder, so that the V-vector as illustrated in Figure 61 (d) can be constructed.

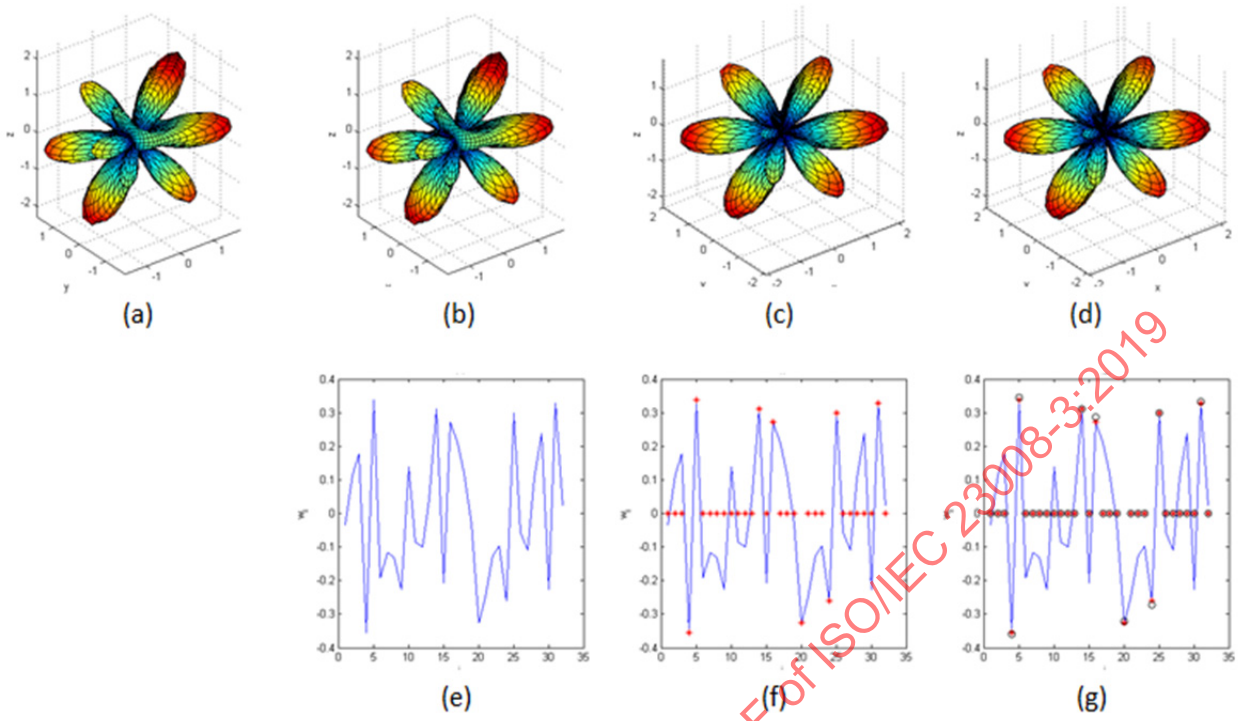


Figure 61 — Example for vector-quantization of V-vector

The size of the vector codebook depends on the value CodebkIdx(k)[i], on the value NumVvecIndices(k)[i] and on the HOA order. If NumVvecIndices is larger than 1, the 256x8 weighting values (Table F.14) are used. If NumVvecIndices is larger than 8, the last 2 columns of the 256x8 weighting values (Table F.14) are used repeatedly with a modular operator.

If the CodebkIdx(k)[i] is set to 0, a codebook containing the HOA expansion coefficients derived from Annex F.9 is used.

If the CodebkIdx(k)[i] is set to 1 the V-vector codebook is generated based on the loudspeaker directions in Table F.12 and used with scaling. If the CodebkIdx(k)[i] is set to 2, the V-vector codebook based on the loudspeaker directions in Table F.12 is generated and used without further scaling. If the CodebkIdx(k)[i] is set to 3, the V-vector codebook with 64 entries as shown in the Table F.13 is used. If the CodebkIdx(k)[i] is set to 7, a vector with O vectors is used. For the HOA order 4, the Vector codebook with 32 entries as derived from the Table F.6 is used.

12.4.1.11.2 VVecLength and VVecCoeffId

The codedVVecLength word indicates.

Value 0: Complete vector length (`NumOfHoaCoeffs` elements). Indicates that all of the coefficients for the predominant vectors `(NumOfHoaCoeffs)` are specified.

Value 1: Vector elements 1 to `MinNumOfCoeffsForAmbHOA` and all elements defined in `ContAddHoaCoeff[lay]` are not transmitted, where lay is the index of layer containing the vector-based signal corresponding to the vector. Indicates that only those coefficients of the predominant vector corresponding to the number greater than a `MinNumOfCoeffsForAmbHOA` are specified. Further those `NumOfContAddAmbHoaChan[lay]` coefficients identified in

ContAddAmbHoaChan[lay] are subtracted. The list ContAddAmbHoaChan[lay] specifies additional channels corresponding to an order that exceeds the order MinAmbHoaOrder.

Value 2: Vector elements 1 to MinNumOfCoeffsForAmbHOA are not transmitted. Indicates that those coefficients of the predominant vectors corresponding to the number greater than a MinNumOfCoeffsForAmbHOA are specified.

In case of codedVVecLength==1 both the VVecLength[i] array as well as the VVecCoeffId[i][m] 2D array are valid for the VVector of index i, in the other cases both the VVecLength element as well as the VVecCoeffId[m] array are valid for all VVector within the HOAFrame. For the assignment algorithm below a helper function is defined as follows.

```
switch CodedVVecLength{
    case 0:
        VVecLength = NumOfHoaCoeffs;
        for (m=0; m<VVecLength; ++m) {
            VVecCoeffId[m] = m;
        }
        break;
    case 1:
        for (i=0; i < NumOfVecSigs; ++i) {
            lay = VecSigLayerIdx[i];
        VVecLength[i] = NumOfHoaCoeffs - MinNumOfCoeffsForAmbHOA -
NumOfContAddHoaChans[lay] - (NewChannelTypeOne[i] * NumOfNewAddHoaChans[lay]);
            CoeffIdx = MinNumOfCoeffsForAmbHOA;
            for (m=0; m<VVecLength[i]; ++m) {
                CoeffIdx++;
                if(NewChannelTypeOne[i]) {
                    bIsInArray = ( isMemberOf(CoeffIdx, ContAddHoaCoeff[lay],
NumOfContAddHoaChans[lay]) || isMemberOf(CoeffIdx, NewAddHoaChans[lay],
NumOfNewAddHoaChans[lay]) );
                    while (bIsInArray) {
                        CoeffIdx++;
                        bIsInArray = ( (isMemberOf(CoeffIdx, ContAddHoaCoeff[lay],
NumOfContAddHoaChans[lay]) || isMemberOf(CoeffIdx, NewAddHoaChans[lay],
NumOfNewAddHoaChans[lay]) );
                    }
                }
                else{
                bIsInArray = isMemberOf(CoeffIdx, ContAddHoaCoeff[lay],
                    NumOfContAddHoaChans[lay]);
                while (bIsInArray) {
                    CoeffIdx++;
                    bIsInArray = isMemberOf(CoeffIdx, ContAddHoaCoeff[lay],
                        NumOfContAddHoaChans[lay]);
                    }
                }
            VVecCoeffId[i][m] = CoeffIdx-1;
            }
        }
    case 2:
        VVecLength = NumOfHoaCoeffs - MinNumOfCoeffsForAmbHOA;
        for (m=0; m< VVecLength; ++m) {
            VVecCoeffId[m] = m + MinNumOfCoeffsForAmbHOA;
        }
}
```

The first switch statement with the three cases (cases 0-2) thus provides a way by which to determine the predominant vector length in terms of the number (VVecLength) and indices of coefficients (VVecCoeffId).

12.4.1.11.3 huffSelect `huffIdx`

```
huffIdx = 5;
if (CbFlag(k)[i] == 1) {
    huffIdx = min(3, max(1, ceil(sqrt(VVecCoeffId[m]+1) - 1)));
}
else if (PFlag(k)[i] == 1) {
    huffIdx = 4;
}
```

12.4.1.11.4 huffDecode `cid`

```
thisTable = huffmanTable[NbitsQ[i]].codebook[huffIdx];
huffWordFound = 0;
word = 0;
while (huffWordFound == 0) {
   word = concatenate( word, hufVal);
   for ( l = 0 ; l < thisTable.length ; l++ ) {
      if ( word == thisTable [l] ) {
         huffWordFound = 1;
         cid = word->symbol;
      }
    }
}
```

12.4.1.11.5 Conversion to VVec element

The type of dequantization of the V-vector is signalled by the word NbitsQ. The NbitsQ value of 4 indicates vector-quantization. When NbitsQ equals 5, a uniform 8 bit scalar dequantization is performed. In contrast, an NbitsQ value of greater or equal to 6 indicates the application of Huffman decoding of a scalar-quantized V-vector. The prediction mode is denoted as the PFlag, while the CbFlag represents a Huffman Table information bit.

```
if (CodedVVecLength == 1) {
   VVecLengthUsed = VVecLength[i];
   VVecCoeffIdUsed = VVecCoeffId[i];
} else {
   VVecLengthUsed = VVecLength;
   VVecCoeffIdUsed = VVecCoeffId;
}
for (m=0; m<O; ++m) {
   v^(i)_m(k) = 0;
}

if (NbitsQ(k)[i] == 4) {
      FNorm = 1.0;
for (m=0; m< O; ++m) {
      TmpVVec[m] = 0;
         for (j=0; j< NumVvecIndices(k)[i]; ++j) {
         TmpVVec[m]+=WeightVal[j]*VecDict[CodebkIdx].[VvecIdx[j]][m];
         }
```

```
      }
if (doScaling) {
      FNorm = 0.0;
         for (m=0; m<O; ++m) {
            FNorm += TmpVVec[m] * TmpVVec[m];
         }
      }
      for (m=0; m< VVecLengthUsed; ++m) {
         idx = VVecCoeffIDUsed[m];
         v^(i)_idx(k) = TmpVVec[idx] * (N+1)/sqrt(FNorm);
      }
}
elseif (NbitsQ(k)[i] == 5) {
   for (m=0; m< VVecLengthUsed; ++m) {
        v^(i)_VVecCoeffIdUsed[m](k) =  (N+1)*aVal[i][m];
   }
}
elseif (NbitsQ(k)[i] >= 6) {
   for (m=0; m< VVecLengthUsed; ++m) {
      v^(i)_VVecCoeffIdUsed[m](k) = (N+1) * (2^(16 - NbitsQ(k)[i])*aVal[i][m])/2^15;
      if (PFlag(k)[i] == 1) {
         v^(i)_VVecCoeffIdUsed[m](k) += floor(0.5 + v^(i)_VVecCoeffId[m](k-1) * 2^14) * 2^-14;
      }
   }
}
if ( (CodedVVecLength == 1) & ((M_LAY(k) - 1) > VecSigLayerIdx[i]) ) {
   for (m=0; m< VVecLengthUsed; ++m) {
      CoeffIdx = VVecCoeffIdUsed[m];
      if (isMemberOf(CoeffIdx, ContAddHoaCoeff[M_LAY(k) - 1],
               NumOfContAddHoaChans[M_LAY(k) - 1]) ) {
         v^(i)_CoeffIdx(k)= 0;
      }
   }
}
```

12.4.1.11.6 selectCodebk

```
switch CodebkIdx {
   case 0:
      cdbLen = 900;
      nbitsIdx = 10;
      doScaling = 1;
      break;
   case 1:
      cdbLen = 34;
      nbitsIdx = 6;
      doScaling = 1;
      break;
   case 2:
      cdbLen = 34;
      nbitsIdx = 6;
      doScaling = 0;
      break;
   case 3:
```

```
        cdbLen = 64;
        nbitsIdx = 6;
        doScaling = 1;
        break;
    case 7:
        cdbLen = (N+1)*(N+1);
        if (N==4){
            cdbLen = 32;
        }
        nbitsIdx = ceil(log2(NumOfHoaCoeffs));
        doScaling = 1;
}
```

12.4.1.11.7 Tuple set $\mathcal{M}_{\mathrm{VEC}}(k)$

$\mathcal{M}_{\mathrm{VEC}}(k) = \emptyset;$

```
for (i=0; i < I_ADD,LAY (k); i++) {
    if (ChannelType[i]==1) {
        M_VEC(k) = M_VEC(k) ∪ (i + 1, v^(i)(k));
    }
}
```

12.4.1.12 Tuple sets $\widetilde{\mathcal{M}}_{\mathrm{DIR}}(k, b)$ for sub-band directional signals synthesis

Q_{VAR} = NumOfGridPointsTable;

for (b=0; b < NumOfPredSubbands; b++)
{
    $\tilde{J}_{\mathrm{DIR}}(k, b) = \emptyset;$
    if(KeepPreviousDirPredMatrixFlag[b])
    {
        $\widetilde{\mathcal{M}}_{\mathrm{DIR}}(k, b) = \widetilde{\mathcal{M}}_{\mathrm{DIR}}(k-1, b);$
    }
    else
    {
        $\widetilde{\mathcal{M}}_{\mathrm{DIR}}(k, b) = \emptyset;$
        for (d=0; d < MaxNumOfPredDirsPerBand; d++)
        {
            if(DirIsActive[b][d])
            {
                $\boldsymbol{\Omega}_{\mathrm{SB,d}}(k-1, b) = \boldsymbol{\Omega}^{(\sqrt{Q_{VAR}}-1)}_{\mathrm{PredDirGridIdx[b][d]}}$; // according to tables F.9 – F.11
                $\widetilde{\mathcal{M}}_{\mathrm{DIR}}(k, b) = \widetilde{\mathcal{M}}_{\mathrm{DIR}}(k, b) \cup \{(d+1, \boldsymbol{\Omega}_{\mathrm{SB,d}}(k-1, b))\};$
            }
        }
    }
}

12.4.1.13Sub-band prediction indicator $\widetilde{b}_{\mathrm{SBP}}(k)$ for sub-band directional signals synthesis

$\widetilde{b}_{\mathrm{SBP}}(k) = \mathrm{UseDirectionalPrediction}$;

12.4.1.14Prediction coefficient matrices $\widetilde{\boldsymbol{A}}(k, b)$ for sub-band directional signals synthesis

Since the prediction coefficient matrix elements are coded differentially, before starting to decode/convert the elements for a k-th independent frame it is necessary to initialize the quantized values to zero for the previous frame as follows.

```
if (hoaIndependencyFlag(k) )
{
    for (b=0; b < NumOfPredSubbands; b++)
    {
        for (d=0; d < MaxNumOfPredDirsPerBand; d++)
        {
            for (n=0; n < MaxNumOfCoeffsToBeTransmitted; n++)
            {
                IntQuantMag(k − 1)[b][d][n] = 0;
                IntQuantAngle(k − 1)[b][d][n] = 0;
            }
        }
    }
}
```

The actual decoding/conversion is assumed to be performed as follows.

```
for (b=0; b < NumOfPredSubbands; b++)
{
    if((KeepPreviousDirPredMatrixFlag[b] ! = 0)  && !hoaIndependencyFlag)
    {
        Ã(k, b) = Ã(k − 1, b);
        for (d=0; d < MaxNumOfPredDirsPerBand ; d++){
            for (n=0; n < MaxNumOfCoeffsToBeTransmitted; n++){
                IntQuantMag(k)[b][d][n ] = IntQuantMag(k − 1)[b][d][n ];
                IntQuantAngle(k)[b][d][n ]  = IntQuantAngle(k − 1)[b][d][n ];
            }
        }
    }
    else
    {
        for (d=0; d < MaxNumOfPredDirsPerBand; d++)
        {
            if (DirIsActive[b][d])
            {
                for (n=0; n < MaxNumOfCoeffsToBeTransmitted; n++)
                {
                    if( (n + 1) ∈ I_E(k) ∪ I_D(k) ∪ I_U(k) ){
                        IntQuantMag(k)[b][d][n] =  IntQuantMag(k − 1)[b][d][n]
                                                 + DecodedMagDiff[b][d][n];
                        if(IntQuantMag(k)[b][d][n]  == 0){
                            IntQuantAngle(k)[b][d][n]  = 0;
                        }
```

```
                        else{
                            IntQuantAngle(k)[b][d][n] = IntQuantAngle(k − 1)[b][d][n]
                                                        + DecodedAngleDiff[b][d][n];
                            // constrain the quantized angle to lie in interval [-7,...,8]
                            IntQuantAngle(k)[b][d][n] = ((IntQuantAngle(k)[b][d][n] + 7)mod16) −
7;
                        }
                        if (IntQuantMag(k)[b][d][n] > 8){
                            FloatMag = (8/7)^(IntQuantMag(k)[b][d][n]−8) ;
                        }
                        else{
                            FloatMag = IntQuantMag(k)[b][d][n] · 1/8
                        }
                        FloatAngle = IntQuantAngle(k)[b][d][n] · π/8 ;
                        Ã(k, b)[d + 1][n + 1] = FloatMag · exp( i · FloatAngle);
                    }
                    else{
                        Ã(k, b)[d + 1][n + 1] = 0;
                        IntQuantMag(k)[b][d][n] = 0;
                        IntQuantAngle(k)[b][d][n] = 0;
                    }
                }
            }
            else
            {
                for (n=0; n < MaxNumOfCoeffsToBeTransmitted; n++)
                {
                    IntQuantMag(k)[b][d][n] = 0;
                    IntQuantAngle(k)[b][d][n] = 0;
                    Ã(k, b)[d + 1][n + 1] = 0;
                }
            }
        }
    }
}
```

12.4.1.15 Mixing matrices $\mathbf{M}_{\mathbf{PAR}}(k, g)$ for parametric ambience replication

Since the mixing matrix elements are coded differentially, before starting to decode/convert the elements for a k-th independent frame it is necessary to initialize the quantized values to zero for the previous frame as follows.

```
if (hoaIndependencyFlag(k) )
{
    for (g=0; g < NumOfParSubbands; g++)
    {
        for (d=0; d < MaxNumOfDecoSigs(g); d++)
        {
            for (n=0; n < MaxNumOfDecoSigs(g); n++)
            {
                IntQuantMagPAR(k − 1)[g][d][n] = 0;
                IntQuantAnglePAR(k − 1)[g][d][n] = 0;
            }
        }
    }
}
```

The actual decoding/conversion is assumed to be performed as follows:

```
for (g=0; g < NumOfParSubbands; g++)
{
    if((KeepPreviousParMatrixFlag[g] ! = 0) && !hoaIndependencyFlag)
    {
        M̃_PAR(k, g) = M̃_PAR(k − 1, g);
        for (d=0; d < MaxNumOfDecoSigs(g); d++){
            for (n=0; n < MaxNumOfDecoSigs(g); n++){
                IntQuantParMag(k)[g][d][n ] = IntQuantParMag(k − 1)[g][d][n ];
                IntQuantParAngle(k)[g][d][n ] = IntQuantParAngle(k − 1)[g][d][n ];
            }
        }
    }
    else
    {
        N_SIG(k, g) = NumOfDecorrSigsPerParSubbandTable
                                [ParDecorrSigsSelectionTableIdx[g]];
      CurrParSelectedDecorrSigsIdxMatrix = ParSelectedDecorrSigsIdxMatrixTable
    [ParDecorrSigsSelectionTableIdx[g]];
        for (d=0; d < MaxNumOfDecoSigs(g); d++)
        {
            for (n=0; n < MaxNumOfDecoSigs(g); n++){
                IntQuantParMag(k)[g][d][n ] = 0;
                IntQuantParAngle(k)[g][d][n ] = 0;
                M_PAR(k, g)[d + 1][n + 1] = 0;
            }
            if(UseParUpmixSig[g][d] ! = 0)
            {
                for (n=0; n < N_SIG(k, g); n++)
                {
                    CurrColIdx = CurrParSelectedDecorrSigsIdxMatrix [d][n];
                    IntQuantParMag(k)[g][d][CurrColIdx ] = IntQuantParMag(k − 1)[g][d][CurrColIdx ]
                                + DecodedParMagDiff[g][d][CurrColIdx ];
                    if(IntQuantParMag(k)[g][d][CurrColIdx ] == 0)
                    {
```

```
                    IntQuantParAngle(k)[g][d][CurrColIdx ] = 0;
                }
                else
                {
                    IntQuantParAngle(k)[g][d][CurrColIdx ] =
                        IntQuantParAngle(k − 1)[b][g][d][CurrColIdx ]
                        + DecodedParAngleDiff[b][d][CurrColIdx ];
                    // constrain the quantized angle to lie in interval [-7,...,8]
                    IntQuantParAngle(k)[g][d][CurrColIdx ] =
                            ((IntQuantParAngle(k)[g][d][CurrColIdx ] + 7)mod16 ) − 7;
                }
                if (IntQuantParMag(k)[b][d][n] > 8)
                {
                    FloatParMag = (8/7)^(IntQuantParMag(k)[b][d][n]−8) ;
                }
                else
                {
                    FloatParMag = IntQuantParMag(k)[b][d][n] · 1/8
                }
                FloatParAngle = IntQuantParAngle(k)[b][d][CurrColIdx ] · π/8 ;
                M_PAR(k, g)[d + 1][CurrColIdx + 1] = FloatParMag · exp( i · FloatParAngle);
            }
        }
    }
}
```

12.4.1.16 Permutation matrices $\boldsymbol{P}_{\mathrm{PAR}}(k, g)$ for PAR

```
for (g=0; g < LastFirstOrderSubBandIdx; g++)
{
    N_SIG(k, g) = NumOfDecorrSigsPerFirstOrderParSubbandTable
                                [ParDecorrSigsSelectionTableIdx[g]];
    CurrParPermIdxVector =

ParFirstOrderPermIdxVectorTable[ParDecorrSigsSelectionTableIdx[g]];
    for (d=0; d < MaxNumOfDecoSigs(g); d++)
    {
        for (n=0; n < MaxNumOfDecoSigs(g); n++){
            P_PAR(k, g)[n + 1][d + 1] = 0;
        }
        P_PAR (k, g)[CurrParPermIdxVector[d]+1][d + 1]= 1;
    }
}
for (g= LastFirstOrderSubBandIdx; g < NumOfParSubbands; g++)
{
    N_SIG(k, g) = NumOfDecorrSigsPerParSubbandTable
                                [ParDecorrSigsSelectionTableIdx[g]];
    CurrParPermIdxVector =
                            ParPermIdxVectorTable[ParDecorrSigsSelectionTableIdx[g]];
    for (d=0; d < MaxNumOfDecoSigs(g); d++)
    {
```

```
            for (n=0; n < MaxNumOfDecoSigs(g); n++){
                P_PAR(k, g)[n + 1][d + 1] = 0;
            }
            P_PAR (k, g)[CurrParPermIdxVector[d]+1][d + 1]= 1;
        }
}
```

12.4.1.17 Indicator $\tilde{b}_{\mathrm{PAR}}(k)$ for use of PAR

$\tilde{b}_{\mathrm{PAR}}(k) = \mathrm{UsePAR}$;

12.4.2 Spatial HOA decoding

12.4.2.1 General architecture

The Spatial HOA decoding process describes how to reconstruct the HOA coefficient sequences from the HOA transport channels and the HOA side information (HOA extension payload). Subsequently the HOA rendering matrix is applied to the HOA coefficient sequences to get the final loudspeaker signals. The HOA renderer is described in subclause 12.4.3. Both processing steps can be very efficiently combined resulting in an implementation with much lower computational complexity. Since the HOA synthesis in the decoding process can be expressed as a synthesizing matrix operation, the rendering matrix can be applied to the synthesizing matrix for such combination. This realizes "decoding and rendering" in one-step rendering without having to reconstruct full HOA coefficient sequences when they are not necessary. A detailed description how to integrate the spatial HOA decoding and rendering can be found in Annex G. However, for easier explanation the processing steps are described separately in the following subclauses.

The architecture of the spatial HOA decoder is depicted in Figure 62.

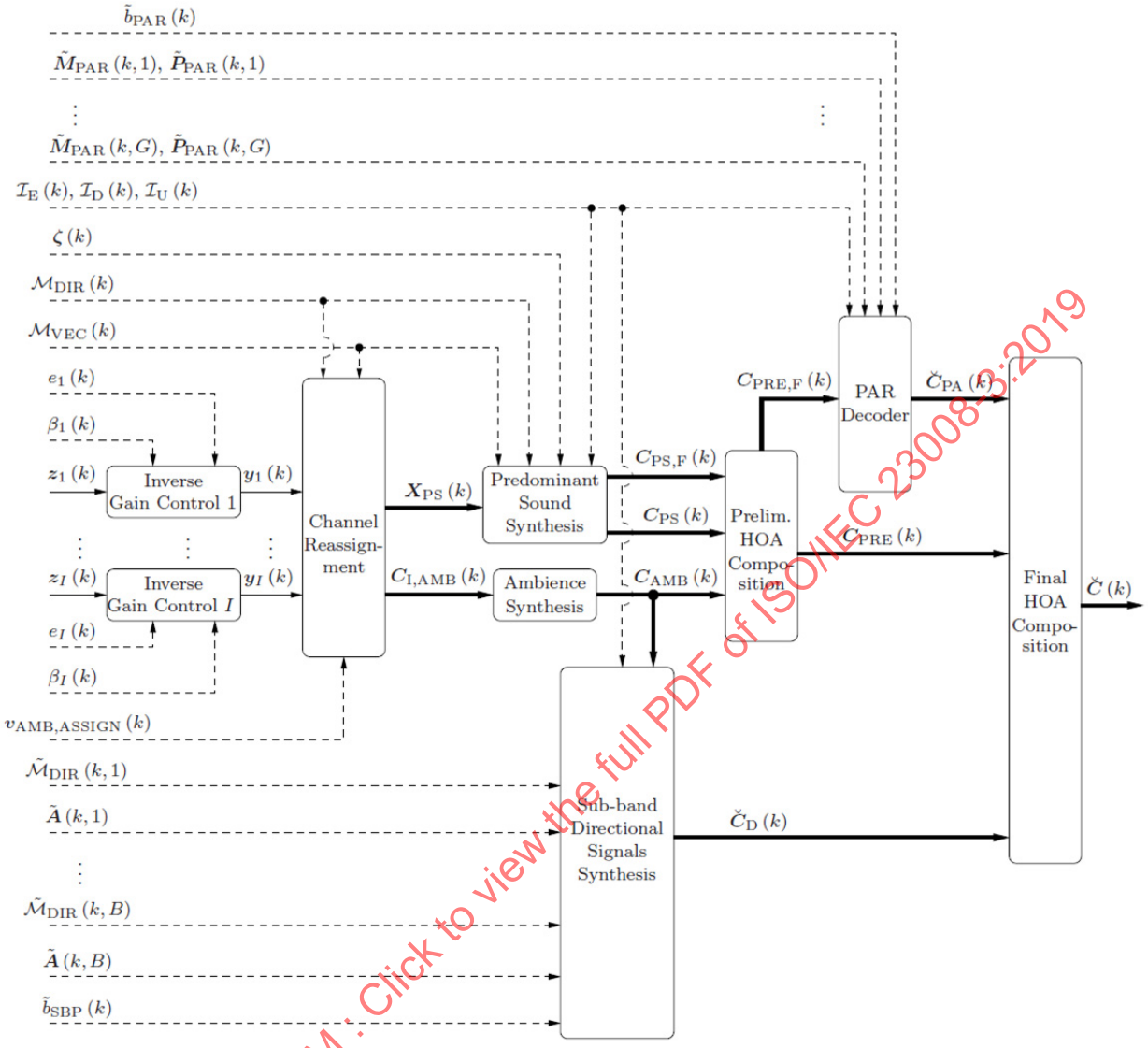


Figure 62 — Architecture of spatial HOA decoder

For each $i \in \{1, \ldots, I\}$, the k-th frame of the i-th perceptually decoded signal, i.e. $\boldsymbol{z}_i(k)$, is input to an inverse gain control processing block together with the associated gain correction exponent $e_i(k)$ and gain correction exception flag $\beta_i(k)$. The i-th inverse gain control processing provides a gain corrected signal frame $\boldsymbol{y}_i(k)$.

All of the I gain corrected signal frames $\boldsymbol{y}_i(k)$, $i \in \{1, \ldots, I\}$, are passed together with the assignment vector $\boldsymbol{v}_{\mathrm{AMB,ASSIGN}}(k)$ and the tuple sets $\mathcal{M}_{\mathrm{DIR}}(k)$ and $\mathcal{M}_{\mathrm{VEC}}(k)$ to the channel reassignment processing block, where they are redistributed to create the frame $\boldsymbol{X}_{\mathrm{PS}}(k)$ of all predominant sound signals (i.e., all directional and vector-based signals) and the frame $\boldsymbol{C}_{\mathrm{I,AMB}}(k)$ of an intermediate representation of the ambient HOA component.

The meaning of the input parameters to the channel reassignment is as follows. The assignment vector $\boldsymbol{v}_{\mathrm{AMB,ASSIGN}}(k)$ indicates for each transport channel the index of a possibly contained coefficient sequence of the ambient HOA component. The tuple set

$$\mathcal{M}_{\mathrm{DIR}}(k) := \left\{ \left(i, \boldsymbol{\Omega}_{\mathrm{TEST,QUANT},i}(k) \right) \middle| \, i \text{ is index of an active direction for } (k+1)^{th} \text{ and } (k)^{th} \text{ frame} \right\}$$

consists of tuples of which the first element i denotes the index of an active direction and of which the second element $\boldsymbol{\Omega}_{\mathrm{TEST,QUANT},i}(k)$ denotes the respective quantized direction defined in F.9. In other words, the first element of the tuple indicates the index i of the gain corrected signal frame $\boldsymbol{y}_i(k)$ that represents the directional signal related to the quantized direction $\boldsymbol{\Omega}_{\mathrm{TEST,QUANT},i}(k)$ given by the second element of the tuple. The quantized directions $\boldsymbol{\Omega}_{\mathrm{TEST,QUANT},i}(k)$ are assumed to have been determined at the spatial HOA encoding stage by an analysis of the input HOA representation. Such an analysis is likely to have been based on a search for the most energetically dominant components of the sound scene using the directional power distribution (see Annex C.5.3 for details). Since directions are in fact computed with respect to two successive frames due to overlap add processing, it is often the case that for the last frame of the activity period for a directional signal there is actually no direction, which is signalled by setting the respective quantized direction to zero.

The set $\mathcal{M}_{\mathrm{VEC}}(k)$

$$\mathcal{M}_{\mathrm{VEC}}(k) := \left\{ \left(i, \boldsymbol{v}^{(i)}(k) \right) \middle| \quad i \text{ is the index of a vector found for } (k+1)^{th} \text{ and } (k)^{th} \text{ frame} \right\}$$

consists of tuples of which the first element indicates the index i of the gain corrected signal frame $\boldsymbol{y}_i(k)$ that represents the signal to be reconstructed by a vector $\boldsymbol{v}^{(i)}(k)$, which is given by the second element of the tuple. The vector $\boldsymbol{v}^{(i)}(k)$ is scalar-dequantized or vector-dequantized as specified in subclause 12.4.1.11 and represents information about the spatial distributions (directions, widths, shapes) of the active signal $\boldsymbol{y}_i(k)$ in the reconstructed HOA frame $\boldsymbol{C}(k)$. It is assumed that $\boldsymbol{v}^{(i)}(k)$ has an Euclidean norm of $N+1$.

In the predominant sound synthesis processing block the HOA representation of the predominant sound component $\boldsymbol{C}_{\mathrm{PS}}(k)$ is computed from the frame $\boldsymbol{X}_{\mathrm{PS}}(k)$ of all predominant sound signals. It uses the tuple sets $\mathcal{M}_{\mathrm{DIR}}(k)$ and $\mathcal{M}_{\mathrm{VEC}}(k)$, the set $\boldsymbol{\zeta}(k)$ of prediction parameters, and the sets $\mathcal{I}_{\mathrm{E}}(k)$, $\mathcal{I}_{\mathrm{D}}(k)$, $\mathcal{I}_{\mathrm{U}}(k)$, which contain indices of coefficient sequences of the ambient HOA component, which are to be enabled, to be disabled and active but not be enabled or disabled, respectively. Additionally, a modified version $\boldsymbol{C}_{\mathrm{PS,F}}(k)$ of $\boldsymbol{C}_{\mathrm{PS}}(k)$ is computed by fading in coefficient sequences with indices contained in the index set $\mathcal{I}_{\mathrm{E}}(k)$ and fading out coefficient sequences with indices contained in the index set $\mathcal{I}_{\mathrm{D}}(k)$. The modified version is only needed for the later computation of the modified version $\boldsymbol{C}_{\mathrm{PRE,F}}(k)$ of the preliminary decoded HOA representation (see subclause 12.4.2.6) to be input to the PAR decoder.

In the ambience synthesis processing block, the ambient HOA component frame $\boldsymbol{C}_{\mathrm{AMB}}(k)$ is created from the frame $\boldsymbol{C}_{\mathrm{I,AMB}}(k)$ of the intermediate representation of the ambient HOA component. This processing also comprises an inverse spatial transform to invert the spatial transform applied in the encoder (see Annex C.5.3) to decorrelate the first O_{MIN} coefficients of the ambient HOA component.

The ambient HOA component frame $\boldsymbol{C}_{\mathrm{AMB}}(k)$ and the frame $\boldsymbol{C}_{\mathrm{PS}}(k)$ of the predominant sound HOA component are superposed in the preliminary HOA composition processing block to provide the frame $\boldsymbol{C}_{\mathrm{PRE}}(k)$ of the preliminary decoded HOA representation. Additionally, the frame $\boldsymbol{C}_{\mathrm{PRE,F}}(k)$ of a modified version of the preliminary decoded HOA representation is computed by replacing the frame $\boldsymbol{C}_{\mathrm{PS}}(k)$ by its modified version $\boldsymbol{C}_{\mathrm{PS,F}}(k)$ for the superposition. The resulting modified HOA representation $\boldsymbol{C}_{\mathrm{PRE,F}}(k)$ is then successively processed by the PAR decoder in place of the original version $\boldsymbol{C}_{\mathrm{PRE}}(k)$. This PAR decoding of $\boldsymbol{C}_{\mathrm{PRE,F}}(k)$ avoids any signal discontinuities that might occur after the truncation and coefficient selection have been performed (see subclause 12.4.2.8.3).

Finally, in the HOA composition processing block the ambient HOA component frame $\boldsymbol{C}_{\mathrm{AMB}}(k)$ and the frame $\boldsymbol{C}_{\mathrm{PS}}(k)$ of the predominant sound HOA component are superimposed to provide the decoded HOA frame $\boldsymbol{C}(k)$.

In the sub-band directional signals synthesis processing block the frame $\breve{\boldsymbol{C}}_{\mathrm{D}}(k)$ of the HOA representation of the composition of all predicted sub-band directional signals is computed. Each directional sub-band signal is assumed to be predicted by a complex valued weighted sum of the transmitted coefficient sequences of the ambient HOA component $\boldsymbol{C}_{\mathrm{AMB}}(k)$, where the indices of the transmitted coefficient sequences are assumed to be among the first O_{MAX}. The prediction of each directional signal related to the j-th sub-band, $j = 1, \ldots, F$, belonging to the b-th sub-band group is carried out using the prediction coefficients matrix $\mathbf{A}(k,b) \in \mathbb{C}^{D_{\mathrm{SB}} \times O_{\mathrm{MAX}}}$ and the tuple set $\widetilde{\mathcal{M}}_{\mathrm{DIR}}(k,b)$. The F assumed sub-bands are uniquely assigned to B sub-band groups, which are determined by the sub-band group configuration specified in the HOAConfig(). It defines for each b-th sub-band group a lower index bound $\mathcal{L}(b)$ and an upper index bound $\mathcal{U}(b)$ such that sub-bands with indices between these bounds, i.e. with $\mathcal{L}(b) \leq j \leq \mathcal{U}(b)$, are assumed to belong to this sub-band group.

Per sub-band group there are at most D_{SB} potential active direction trajectories, where the indices identifying the active direction trajectories for the b-th sub-band group are assumed to be contained in the set $\tilde{\mathcal{J}}_{\mathrm{DIR}}(k,b) \subseteq \{1, \ldots, D_{\mathrm{SB}}\}$. For each index $d \in \tilde{\mathcal{J}}_{\mathrm{DIR}}(k,b)$ of an active direction trajectory the respective direction is denoted by $\boldsymbol{\Omega}_{\mathrm{SB},d}(k,b)$. Both the active direction trajectories and their respective directions are assumed to be contained as tuples in the set $\widetilde{\mathcal{M}}_{\mathrm{DIR}}(k,b)$, i.e.

$$\widetilde{\mathcal{M}}_{\mathrm{DIR}}(k,b) = \left\{ \left(d, \boldsymbol{\Omega}_{\mathrm{SB},d}(k,b) \right) \middle| \, d \in \tilde{\mathcal{J}}_{\mathrm{DIR}}(k,b) \right\}$$

Note that the set $\tilde{\mathcal{J}}_{\mathrm{DIR}}(k,b)$ can be computed from $\widetilde{\mathcal{M}}_{\mathrm{DIR}}(k,b)$, since it contains the first elements of all tuples of $\widetilde{\mathcal{M}}_{\mathrm{DIR}}(k,b)$.

In order to avoid artifacts in the predicted directional sub-band signals due to changes of the estimated directions and prediction coefficients between successive frames, the prediction is performed on concatenated long frames consisting of two temporally successive frames. More specifically, this means that each quantity $\widetilde{\mathcal{M}}_{\mathrm{DIR}}(k,b)$ and $\mathbf{A}(k,b)$ is related to the $(k+1)$-th and k-th frame. The Boolean quantity $\tilde{b}_{\mathrm{SBP}}(k)$ (which can be equal to zero or one) indicates whether a prediction of sub-band directional signals is to be performed relative to the frames k and $k+1$.

The frame $\breve{\boldsymbol{C}}_{\mathrm{D}}(k)$ is assumed to have only non-zero contributions for those coefficient sequences of the ambient HOA component that are not already transmitted within the transport channels. Further, if coefficient sequences of the ambient HOA component are faded in (or faded out respectively), the corresponding coefficient sequences of the HOA representation $\breve{\boldsymbol{C}}_{\mathrm{D}}(k)$ are faded out (or faded in respectively).

Note further that the time domain coefficient sequences of the HOA representation $\breve{\boldsymbol{C}}_{\mathrm{D}}(k)$ of the composition of all predicted sub-band directional signals are delayed by $D_{\mathrm{QMF}} = 577$ samples due to the successive application of the QMF based analysis and synthesis filter banks, which is expressed by the breve symbol (˘) above the variables.

Within the parametric ambience replication (PAR) decoder processing block ambient components, which are potentially still missing within the preliminary decoded HOA representation $\boldsymbol{C}_{\mathrm{PRE}}(k)$, are parametrically replicated from the modified version of it, $\boldsymbol{C}_{\mathrm{PRE,F}}(k)$. The replication shall be carried out in the frequency domain using quadrature mirror filters (QMF) with $F = 64$ sub-bands as defined in ISO/IEC 23003-1. Each individual sub-band j, $j = 1, \ldots, F$, is processed using the corresponding parameters of the g-th sub-band group, $g = 1, \ldots, G$, to which it is uniquely assigned. The assignment is determined by the PAR related sub-band group configuration specified in the HOAConfig(). It defines for each g-th sub-band group a lower index bound $\mathcal{L}_{\mathrm{PAR}}(g)$ and an upper index bound $\mathcal{U}_{\mathrm{PAR}}(g)$ such that sub-bands with indices between these bounds, i.e. with $\mathcal{L}_{\mathrm{PAR}}(b) \leq j \leq \mathcal{U}_{\mathrm{PAR}}(b)$, are assumed to belong to this sub-band group. The PAR related side information for the k-th frame consists of the

mixing matrices $\boldsymbol{M}_{\mathrm{PAR}}(k,g)$ and the permutation matrices $\boldsymbol{P}_{\mathrm{PAR}}(k,g)$ for the individual G sub-band groups $g = 1, \ldots, G$, as well as the boolean quantity $\tilde{b}_{\mathrm{PAR}}(k)$, which indicates (by a zero or one) whether PAR is to be performed for the frames k and $k+1$.

In the final HOA composition processing block the frame $\boldsymbol{C}_{\mathrm{PRE}}(k)$ of the preliminary decoded HOA representation, the frame $\breve{\boldsymbol{C}}_{\mathrm{D}}(k)$ of the HOA representation of the composition of all predicted sub-band directional signals and the frame $\breve{\boldsymbol{C}}_{\mathrm{PA}}(k)$ of the replicated ambient HOA component are superposed to provide the frame $\breve{\boldsymbol{C}}(k)$ of the decoded HOA representation. As part of this superposition of the individual HOA representations, the various delays are to be taken into account appropriately.

In the following, the individual processing blocks are described in more detail.

12.4.2.2 Inverse gain control

The goal of the inverse gain control (IGC) processing block is to invert the gain modifications performed to the signals before perceptual encoding at the HOA encoding stage – recreating their initial value range.

For that purpose, a fixed template transition window function

$$\mathbf{f}_{\mathrm{IGC}} := [f_{\mathrm{IGC}}(1) \quad f_{\mathrm{IGC}}(2) \quad \ldots \quad f_{\mathrm{IGC}}(L)]$$

is employed, whose elements are defined by

$$f_{\mathrm{IGC}}(l) := \frac{1}{4}\cos\left(\frac{\pi(l-1)}{L-1}\right) + \frac{3}{4} \qquad \text{for} \quad l = 1, \ldots, L.$$

Assuming the input frame $\boldsymbol{z}_i(k)$ and output frame $\boldsymbol{y}_i(k)$ are expressed through their samples as

$$\boldsymbol{z}_i(k) := [z_i(k,1) \quad z_i(k,2) \quad \ldots \quad z_i(k,L)]$$

$$\boldsymbol{y}_i(k) := [y_i(k,1) \quad y_i(k,2) \quad \ldots \quad y_i(k,L)],$$

the computation of the samples of the output frame is given by

$$y_i(k,l) = \begin{cases} \frac{z_i(k,l)}{g_{\mathrm{IGC},i}(k-1)} \cdot [f_{\mathrm{IGC}}(l)]^{-e_i(k)} & \text{if} \quad \beta_i(k) = 0 \\ \frac{z_i(k,l)}{g_{\mathrm{IGC},i}(k-1)} \cdot 2^{e_i(k)} & \text{if} \quad \beta_i(k) = 1 \end{cases} \qquad \text{for} \quad i = 1, \ldots, I, l = 1, \ldots, L.$$

The factor $g_{\mathrm{IGC},i}(k-1)$ is initialized by

$$g_{\mathrm{IGC},i}(0) := 1 \qquad \text{for} \quad i = 1, \ldots, I,$$

and is recursively updated in the k-th frame by

$$g_{\mathrm{IGC},i}(k) = g_{\mathrm{IGC},i}(k-1) \cdot 2^{-e_i(k)} \qquad \text{for} \quad i = 1, \ldots, I.$$

For random access, the factor $g_{\mathrm{IGC},i}(k-1)$ is provided explicitely in HOAGainCorrectionData() if the hoaIndependencyFlag has been signalled as true.

12.4.2.3 Channel reassignment

The channel reassignment processing block has the purpose of creating the frame $\boldsymbol{X}_{\mathrm{PS}}(k)$ from all of the predominant sound signals and from the frame $\boldsymbol{C}_{\mathrm{I,AMB}}(k)$ of an intermediate representation of the ambient HOA component from the gain corrected signal frames $\boldsymbol{y}_i(k)$, $i \in \{1, \ldots, I\}$, and the assignment vector $\boldsymbol{v}_{\mathrm{AMB,ASSIGN}}(k)$. The assignement vector indicates, for each transport channel, the index of a possibly contained coefficient sequence of the ambient HOA component. Additionally, the sets $\mathcal{I}_{\mathrm{DIR,ACT}}(k)$ and $\mathcal{I}_{\mathrm{VEC,ACT}}(k)$ are used, which contain the first elements of all tuples of $\mathcal{M}_{\mathrm{DIR}}(k)$ and $\mathcal{M}_{\mathrm{VEC}}(k)$, respectively. It is important to note that these two sets are assumed to be disjoint.

For the actual assignment the following steps are performed.

1) The frame $\boldsymbol{X}_{\mathrm{PS}}(k)$ of all predominant sound signals is assumed to be composed of the individual predominant sound signal frames according to

$$\boldsymbol{X}_{\mathrm{PS}}(k) = \begin{bmatrix} x_{\mathrm{PS},1}(k) \\ x_{\mathrm{PS},2}(k) \\ \vdots \\ x_{\mathrm{PS},D}(k) \end{bmatrix},$$

where each frame consists of its samples according to

$$x_{\mathrm{PS,i}}(k) = \begin{bmatrix} x_{\mathrm{PS},i}(k,1) & x_{\mathrm{PS},i}(k,2) \ldots & x_{\mathrm{PS},i}(k,L) \end{bmatrix} \quad \text{for} \quad i = 1, \ldots, J,$$

where $J = I - O_{\mathrm{MIN}}$.

The sample values of the directional signal frames are computed as follows:

$$x_{\mathrm{PS},i}(k,l) = \begin{cases} \mathrm{y}_i(k,l) & \text{if} \quad i \in \mathcal{I}_{\mathrm{DIR}}(k) \cup \mathcal{I}_{\mathrm{VEC}}(\mathrm{k}) \\ 0 & \text{else} \end{cases} \quad \text{for} \quad i = 1, \ldots, J, \qquad l = 1, \ldots, L.$$

2) The frame $\boldsymbol{C}_{\mathrm{I,AMB}}(k)$ of an intermediate representation of the ambient HOA omponent is assumed to be composed according to

$$\boldsymbol{C}_{\mathrm{I,AMB}}(k) = \begin{bmatrix} \boldsymbol{c}_{\mathrm{I,AMB},1}(k) \\ \boldsymbol{c}_{\mathrm{I,AMB},2}(k) \\ \vdots \\ \boldsymbol{c}_{\mathrm{I,AMB},O}(k) \end{bmatrix}$$

with

$$\boldsymbol{c}_{\mathrm{I,AMB},n}(k) = [c_{\mathrm{I,AMB},n}(k,1) \quad c_{\mathrm{I,AMB},n}(k,2) \quad \ldots \quad c_{\mathrm{I,AMB},n}(k,L)] \quad \text{for} \quad n = 1, \ldots, O.$$

The sample values of the intermediate representation of the ambient HOA component are obtained as follows:

$$c_{\mathrm{I,AMB},n}(k,l) = \begin{cases} y_i(k,l) & \text{if} \quad \exists i \in \{1, \ldots, I\} \quad \text{such that} \quad v_{\mathrm{AMB,ASSIGN},i}(k) = n \\ 0 & \text{else} \end{cases}.$$

In the case that the flag UsePhaseShiftDecorr has a value of 1, it is assumed that the frame $\boldsymbol{C}_{\mathrm{I,AMB}}(k+1)$ instead of $\boldsymbol{C}_{\mathrm{I,AMB}}(k)$ is obtained from the transport channels, which has to be ensured by a respective modification at the spatial HOA encoding stage.

12.4.2.4 Predominant sound synthesis

12.4.2.4.1 General

The purpose of the predominant sound synthesis is to create the frame $\boldsymbol{C}_{\mathrm{PS}}(k)$ of the HOA representation of the predominant sound component from the frame $\boldsymbol{X}_{\mathrm{PS}}(k)$ of all predominant sound signals using the tuple set $\mathcal{M}_{\mathrm{DIR}}(k)$ and $\mathcal{M}_{\mathrm{VEC}}(k)$, the set $\boldsymbol{\zeta}(k)$ of prediction parameters, and the sets $\mathcal{I}_{\mathrm{E}}(k)$, $\mathcal{I}_{\mathrm{D}}(k)$, and $\mathcal{I}_{\mathrm{U}}(k)$.

Additionally, the frame $\boldsymbol{C}_{\mathrm{PS,F}}(k)$ of a modified version of $\boldsymbol{C}_{\mathrm{PS}}(k)$ is computed, where the modification only consist of fading in coefficient sequences with indices contained in the index set $\mathcal{I}_{\mathrm{E}}(k)$ and fading out coefficient sequences with indices contained in the index set $\mathcal{I}_{\mathrm{D}}(\mathrm{k})$. The modified version is only needed for the later computation of the modified version $\mathrm{C}_{\mathrm{PRE,F}}(\mathrm{k})$ of the preliminary decoded HOA representation (see subclause 12.4.2.6) to be input to the PAR decoder. As illustrated in Figure 63, the processing can be subdivided into four processing steps, which are described in the following.

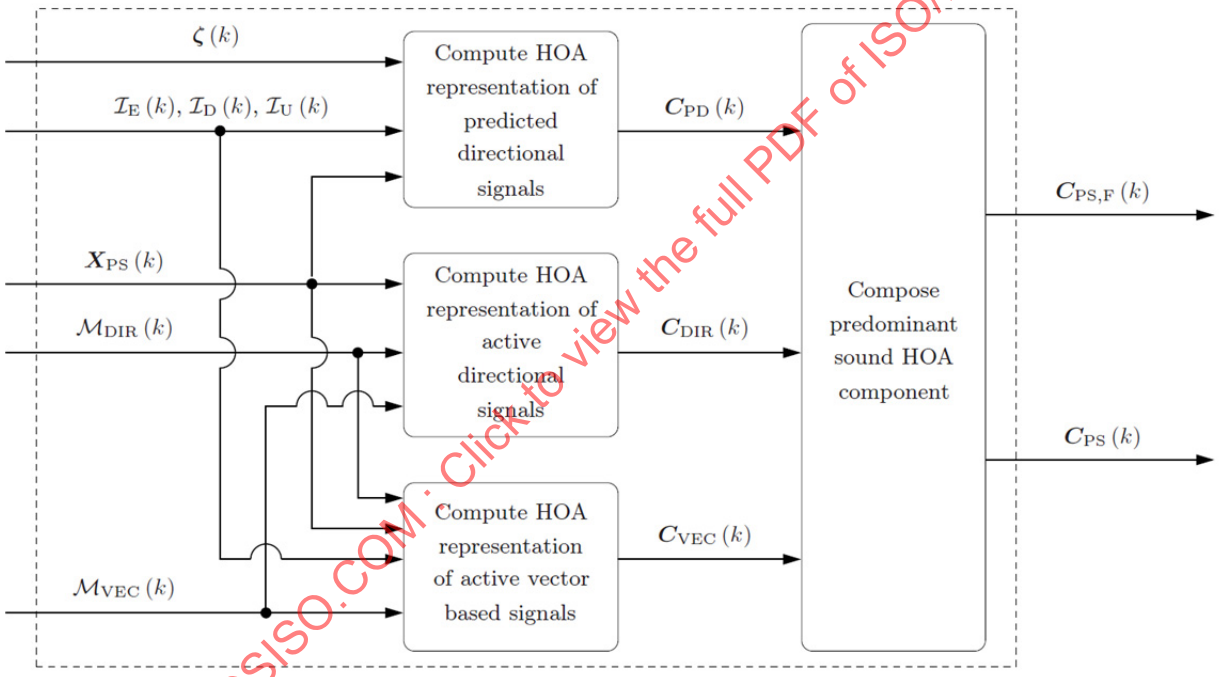


Figure 63 — Predominant sound synthesis

12.4.2.4.2 Compute HOA representation of active directional signals

In order to avoid artifacts due to changes in direction between successive frames, the computation of the HOA representation from the directional signals is based on the concept of overlap add. Hence, the HOA representation $\boldsymbol{C}_{\mathrm{DIR}}(k)$ of active directional signals is computed as the sum of a faded out component and a faded in component:

$$\boldsymbol{C}_{\mathrm{DIR}}(k) = \boldsymbol{C}_{\mathrm{DIR,OUT}}(k) + \boldsymbol{C}_{\mathrm{DIR,IN}}(k)$$

To compute the two individual components, the instantaneous signal frames for directional signal indices $d \in \mathcal{I}_{\mathrm{DIR}}(k_1)$ and directional signal frame index k_2 are defined by:

$$\boldsymbol{C}_{\mathrm{DIR,I}}^{(d)}(k_1;k_2) := \boldsymbol{\Psi}^{(N,29)}\big|_{\boldsymbol{\Omega}_{\mathrm{TEST,QUANT},d}(k_1)}\boldsymbol{x}_{\mathrm{DIR},d}(k_2)$$

where $\boldsymbol{\Psi}^{(N,29)}$ denotes the mode matrix of order N with respect to the predefined directions $\boldsymbol{\Omega}_n^{(29)}, n = 1, \ldots, Q = (29+1)^2 = 900$, is defined in Annex F.1.5 and $\boldsymbol{\Psi}^{(N,29)}|_q$ denotes the q-th column vector of $\boldsymbol{\Psi}^{(N,29)}$.

In the case of single-layered coding (indicated by SingleLayer==1 (see Table 188)), the sample values of the faded out and faded in directional HOA components are then determined by

$$c_{\mathrm{DIR,OUT},i}(k,l) = \sum_{d\in\mathcal{I}_{\mathrm{DIR,NZ}}(k-1)} c_{\mathrm{DIR,I},i}^{(d)}(k-1;k,l)\cdot\begin{cases} w_{\mathrm{DIR}}(L+l) & \text{if } d\in\mathcal{I}_{\mathrm{DIR,NZ}}(k)\\ w_{\mathrm{VEC}}(L+l) & \text{if } d\in\mathcal{I}_{\mathrm{VEC}}(k)\\ 1 & \text{else}\end{cases}$$

$$c_{\mathrm{DIR,IN},i}(k,l) = \sum_{d\in\mathcal{I}_{\mathrm{DIR,NZ}}(k)} c_{\mathrm{DIR,I},i}^{(d)}(k;k,l)\cdot\begin{cases} w_{\mathrm{DIR}}(l) & \text{if } d\in\mathcal{I}_{\mathrm{DIR,NZ}}(k-1)\cup\mathcal{I}_{\mathrm{VEC}}(k-1)\\ 1 & \text{else}\end{cases}$$

where $\mathcal{I}_{\mathrm{DIR,NZ}}(k)$ denotes the set of those first elements of $\mathcal{M}_{\mathrm{DIR}}(k)$ where the corresponding second element is non-zero.

In the case of multiple layered coding (indicated by SingleLayer==0 (see Table 188)), the sample values of the faded out and faded in directional HOA components are then determined by

$$c_{\mathrm{DIR,OUT},i}(k,l) = \sum_{d\in\mathcal{I}_{\mathrm{DIR,NZ}}(k-1)} c_{\mathrm{DIR,I},i}^{(d)}(k-1;k,l)\cdot\begin{cases} w_{\mathrm{DIR}}(L+l)\cdot 1 & \text{if } d\in\mathcal{I}_{\mathrm{DIR,NZ}}(k)\wedge(i\notin\mathcal{I}_{\mathrm{E}}(k)\cup\mathcal{I}_{\mathrm{D}}(k))\\ w_{\mathrm{DIR}}(L+l)\cdot w_{\mathrm{DIR}}(L+l) & \text{if } d\in\mathcal{I}_{\mathrm{DIR,NZ}}(k)\wedge i\in\mathcal{I}_{\mathrm{E}}(k)\\ w_{\mathrm{DIR}}(L+l)\cdot w_{\mathrm{DIR}}(l) & \text{if } d\in\mathcal{I}_{\mathrm{DIR,NZ}}(k)\wedge i\in\mathcal{I}_{\mathrm{D}}(k)\\ w_{\mathrm{VEC}}(L+l) & \text{if } d\in\mathcal{I}_{\mathrm{VEC}}(k)\\ 1\cdot w_{\mathrm{DIR}}(l) & \text{if } d\in\mathcal{I}_{\mathrm{DIR,Z}}(k)\wedge i\in\mathcal{I}_{\mathrm{D}}(k)\\ 1 & \text{else}\end{cases}$$

$$c_{\mathrm{DIR,IN},i}(k,l) = \sum_{d\in\mathcal{I}_{\mathrm{DIR,NZ}}(k)} c_{\mathrm{DIR,I},i}^{(d)}(k;k,l)\cdot\begin{cases} w_{\mathrm{DIR}}(l) & \text{if } \big(d\in\mathcal{I}_{\mathrm{DIR}}(k-1)\cup\mathcal{I}_{\mathrm{VEC}}(k-1)\big)\wedge\big(i\notin\mathcal{I}_{\mathrm{E}}(k)\cup\mathcal{I}_{\mathrm{D}}(k)\big)\\ w_{\mathrm{DIR}}(l)\cdot w_{\mathrm{DIR}}(l) & \text{if } \big(d\in\mathcal{I}_{\mathrm{DIR}}(k-1)\cup\mathcal{I}_{\mathrm{VEC}}(k-1)\big)\wedge i\in\mathcal{I}_{\mathrm{D}}(k)\\ w_{\mathrm{DIR}}(l)\cdot w_{\mathrm{DIR}}(L+l) & \text{if } \big(d\in\mathcal{I}_{\mathrm{DIR}}(k-1)\cup\mathcal{I}_{\mathrm{VEC}}(k-1)\big)\wedge i\in\mathcal{I}_{\mathrm{E}}(k)\\ 1\cdot w_{\mathrm{DIR}}(L+l) & \text{if } \big(d\notin\mathcal{I}_{\mathrm{DIR}}(k-1)\cup\mathcal{I}_{\mathrm{VEC}}(k-1)\big)\wedge i\in\mathcal{I}_{\mathrm{E}}(k)\\ 1 & \text{else}\end{cases}$$

where $\mathcal{I}_{\mathrm{DIR,Z}}(k)$ denotes the set of those first elements of $\mathcal{M}_{\mathrm{DIR}}(k)$ where the corresponding second element is zero.

The fading of the instantaneous HOA representations for the overlap add operation is accomplished with two different fading windows

$$\boldsymbol{w}_{\mathrm{DIR}} := [w_{\mathrm{DIR}}(1) \quad w_{\mathrm{DIR}}(2) \quad \ldots \quad w_{\mathrm{DIR}}(2L)]$$

$$\boldsymbol{w}_{\mathrm{VEC}} := [w_{\mathrm{VEC}}(1) \quad w_{\mathrm{VEC}}(2) \quad \ldots \quad w_{\mathrm{VEC}}(2L)]$$

whose elements are defined by

$$w_{\mathrm{DIR}}(l) := \frac{1}{2}\left[1-\cos\left(2\pi\frac{l-1}{2L}\right)\right]$$

$w_{\mathrm{VEC}}(l)$:

$$= \begin{cases} \dfrac{l-1}{L_{\mathrm{IP}}-1} & \text{if } 1 \le l \le L_{\mathrm{IP}} \wedge \text{SpatialInterpolationMethod} = 0 \\ \dfrac{1}{2}\left[1-\cos\left(2\pi\dfrac{l-1}{2L_{\mathrm{IP}}}\right)\right] & \text{if } 1 \le l \le L_{\mathrm{IP}} \wedge \text{SpatialInterpolationMethod} = 1 \\ 1 & \text{if } L_{\mathrm{IP}}+1 \le l \le L \\ 1-\dfrac{l-L-1}{L_{\mathrm{IP}}-1} & \text{if } L+1 \le l \le L+L_{\mathrm{IP}} \wedge \text{SpatialInterpolationMethod} = 0 \\ 1-\dfrac{1}{2}\left[1-\cos\left(2\pi\dfrac{l-L-1}{2L_{\mathrm{IP}}}\right)\right] & \text{if } L+1 \le l \le L+L_{\mathrm{IP}} \wedge \text{SpatialInterpolationMethod} = 1 \\ 0 & \text{if } L+L_{\mathrm{IP}}+1 \le l \le 2L \end{cases}$$

where L_{IP} is defined in Table 209.

Table 209 — Decoding of codedSpatialInterpolationTime into L_{IP}

L	CodedSpatialInterpolationTime							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
768	0	32	64	128	256	384	512	768
1024	0	64	128	256	384	512	768	1024
2048	0	128	256	512	768	1024	1536	2048
4096	0	256	512	1024	1536	2048	3072	4096

12.4.2.4.3 Compute HOA representation of predicted directional signals

The parameter set $\boldsymbol{\zeta}(k) = \{\boldsymbol{p}_{\mathrm{TYPE}}(k), \boldsymbol{P}_{\mathrm{IND}}(k), \boldsymbol{P}_{\mathrm{Q,F}}(k)\}$ related to the spatial prediction of the directional signals consists of the following components.

— The vector $\boldsymbol{p}_{\mathrm{TYPE}}(k)$, whose elements $p_{\mathrm{TYPE},n}(k)$, $n = 1, \ldots, O$ indicate if for the n-th direction $\boldsymbol{\Omega}_n^{(N)}$, defined in F.2 to F.11 and the k-th and $(k+1)$-th frame a prediction is performed or not, and if so, then they also indicate the kind of prediction. In particular, the meaning of the elements is as follows:

$$p_{\mathrm{TYPE},n}(k) = \begin{cases} 0 & \text{for no prediction for the direction } \boldsymbol{\Omega}_n^{(N)} \text{ and } k-\text{th and } (k+1)-\text{th frame} \\ 1 & \text{for a full band prediction for the direction } \boldsymbol{\Omega}_n^{(N)} \text{ and } k-\text{th and } (k+1)-\text{th frame} \end{cases}.$$

— The matrix $\boldsymbol{P}_{\mathrm{IND}}(k)$, whose elements $p_{\mathrm{IND},d,n}(k)$, $d = 1, \ldots, D_{\mathrm{PRED}}$, $n = 1, \ldots, O$ denote the indices from which directional signals the prediction for the direction $\boldsymbol{\Omega}_n^{(N)}$ and the k-th and $(k+1)$-th frame has to be performed. If no prediction is to be performed for a direction $\boldsymbol{\Omega}_n^{(N)}$, the corresponding column of the matrix $\boldsymbol{P}_{\mathrm{IND}}(k)$ consists of zeros. Further, if less than D_{PRED} directional signals are used for the prediction for a direction $\boldsymbol{\Omega}_n^{(N)}$, the non-required elements in the n-th column of $\boldsymbol{P}_{\mathrm{IND}}(k)$ are also zero.

— The matrix $\boldsymbol{P}_{\mathrm{Q,F}}(k)$, which contains the corresponding quantized prediction factors $p_{\mathrm{Q,F},d,n}(k)$, $d = 1, \ldots, D_{\mathrm{PRED}}$, $n = 1, \ldots, O$.

Note that the prediction parameters $\boldsymbol{\zeta}(k)$ are related to the frames k and $(k+1)$. Additionally, the following dependent quantity:

$$b_{\mathrm{ACT}}(k) = \begin{cases} 1 & \text{if } \exists n \text{ such that } p_{\mathrm{TYPE},n}(k) = 0 \\ 0 & \text{else} \end{cases}$$

is introduced, which indicates whether a prediction is to be performed relative to frames k and $k+1$. Further, the quantized prediction factors $p_{\mathrm{Q,F},d,n}(k), d = 1, \ldots, D_{\mathrm{PRED}}, n = 1, \ldots, O$, are dequantized to provide the actual prediction factors:

$$p_{\mathrm{F},d,n}(k) = \left(p_{\mathrm{Q,F},d,n}(k) + \frac{1}{2}\right) \cdot 2^{-B_{\mathrm{SC}}+1}.$$

The idea behind the computation of the predicted HOA component is to represent it by directional signals (i.e. general plane wave functions) impinging from the predefined directions $\boldsymbol{\Omega}_n^{(N)}, n = 1, \ldots, O$, and then transform this representation to an HOA representation. These directional signals impinging from the predefined directions $\boldsymbol{\Omega}_n^{(N)}, n = 1, \ldots, O$, are predicted from the predominant sound signals $\boldsymbol{X}_{\mathrm{PS}}(k)$ using the parameter set $\boldsymbol{\zeta}(k)$ The computation of the predicted directional signals is based on the concept of overlap add in order to avoid artifacts due to changes of prediction parameters between successive frames. Hence, the k-th frame of the predicted directional signals, denoted by $\mathbf{X}_{\mathrm{PD}}(k)$, is computed as the sum of a faded out component and a faded in component:

$$\mathbf{X}_{\mathrm{PD}}(k) = \mathbf{X}_{\mathrm{PD,OUT}}(k) + \mathbf{X}_{\mathrm{PD,IN}}(k).$$

The sample values $x_{\mathrm{PD,OUT},n}(k,l)$ and $x_{\mathrm{PD,IN},n}(k,l), n = 1, \ldots, O, l = 1, \ldots, L$, of the faded out and faded in predicted directional signals are then respectively computed by:

$$x_{\mathrm{PD,OUT},n}(k,l) = w_{\mathrm{DIR}}(L+l) \cdot \begin{cases} 0 & \text{if } p_{\mathrm{TYPE},n}(k-1) = 0 \\ \sum_{d=1}^{D_{\mathrm{PRED}}} p_{\mathrm{F},d,n}(k-1) \cdot x_{\mathrm{PS},p_{\mathrm{IND},d,n}(k-1)}(k,l) & \text{if } p_{\mathrm{TYPE},n}(k-1) = 1 \end{cases}$$

$$x_{\mathrm{PD,IN},n}(k,l) = w_{\mathrm{DIR}}(l) \cdot \begin{cases} 0 & \text{if } p_{\mathrm{TYPE},n}(k) = 0 \\ \sum_{d=1}^{D_{\mathrm{PRED}}} p_{\mathrm{F},d,n}(k) \cdot x_{\mathrm{PS},p_{\mathrm{IND},d,n}(k)}(k,l) & \text{if } p_{\mathrm{TYPE},n}(k) = 1 \end{cases}$$

In a next step, the predicted directional signals are transformed to the HOA domain by:

$$\boldsymbol{C}_{\mathrm{PD,I}}(k) = \boldsymbol{\Psi}^{(N,N)} \cdot \boldsymbol{X}_{\mathrm{PD}}(k),$$

where $\boldsymbol{\Psi}^{(N,N)}$ denotes the mode matrix of order N with respect to the predefined directions $\boldsymbol{\Omega}_n^{(N)}, n = 1, \ldots, O$, defined in Annex F.1.5, Assuming the preliminary HOA representation $\boldsymbol{C}_{\mathrm{PD,I}}(k)$ of the predicted directional signals to be expressed by means of its samples by:

$$\boldsymbol{C}_{\mathrm{PD,I}}(k) = \begin{bmatrix} c_{\mathrm{PD,I},1}(k,1) & \ldots & c_{\mathrm{PD,I},1}(k,L) \\ \vdots & \ddots & \vdots \\ c_{\mathrm{PD,I},O}(k,1) & \ldots & c_{\mathrm{PD,I},O}(k,L) \end{bmatrix}$$

the samples of the final output HOA representation:

$$\boldsymbol{C}_{\mathrm{PD}}(k) = \begin{bmatrix} c_{\mathrm{PD},1}(k,1) & \ldots & c_{\mathrm{PD},1}(k,L) \\ \vdots & \ddots & \vdots \\ c_{\mathrm{PD},O}(k,1) & \ldots & c_{\mathrm{PD},O}(k,L) \end{bmatrix}$$

of the predicted directional signals are computed:

$$c_{\mathrm{PD},n}(k,l) = \begin{cases} 0 & \text{if } n \in \mathcal{I}_{\mathrm{U}}(k) \\ c_{\mathrm{PD,I},n}(k,l) \cdot w_{\mathrm{DIR}}(l) & \text{if } n \in \mathcal{I}_{\mathrm{D}}(k) \wedge b_{\mathrm{ACT}}(k-1) = 1 \\ c_{\mathrm{PD,I},n}(k,l) \cdot w_{\mathrm{DIR}}(L+l) & \text{if } n \in \mathcal{I}_{\mathrm{E}}(k) \wedge b_{\mathrm{ACT}}(k) = 1 \\ c_{\mathrm{PD,I},n}(k,l) & \text{else} \end{cases} \quad \text{for } n = 1, \ldots, O, l = 1, \ldots, L.$$

12.4.2.4.4 Compute HOA representation of active vector-based signals

12.4.2.4.4.1 General

The computation of the HOA representation of the vector-based signals may be considered as the interpolation of V-vectors which describe the distribution of the predominant sound components in space (e.g., directions, shapes, and widths). Because a V-vector already contains all relevant spatial information about the predominant sound components, a spatial prediction, as specified in subclause 12.4.2.4.3, is not used for vector-based signals. The interpolation of the V-vectors is carried out over time. This spatio-temporal interpolation ensures a continuous and smooth evolution of the soundfield across frame boundaries, that lends itself to perceptually transparent quality. In this manner, the HOA signal is re-composed from the original decomposition comprised of the interpolated vectors and the corresponding predominant signals.

Figure 64 illustrates how the HOA coefficients are reformulated from the vector-based signals. Scalar dequantization is first performed with respect to each V-vector to generate $\mathcal{M}_{\mathrm{VEC}}(k)$, where the i^{th} individual vectors of the current frame may be denoted as $\boldsymbol{v}_{\mathrm{I}}^{(i)}(k)$. The V-vectors are decomposed from the HOA coefficients typically employing a linear invertible transform (see Annex C.5.3.2.1). Spatio-temporal interpolation is performed with respect to the $\mathcal{M}_{\mathrm{VEC}}(k)$ and $\mathcal{M}_{\mathrm{VEC}}(k-1)$ (which denotes V-vectors from a previous frame with individual vectors of $\mathcal{M}_{\mathrm{VEC}}(k-1)$ denoted as $\boldsymbol{v}_{\mathrm{O}}^{(i)}(k)$). The interpolation method is controlled by $w_{\mathrm{VEC}}(l)$, as described in more detail below. Following interpolation, the i^{th} interpolated V-vector ($\overline{\boldsymbol{v}^{(i)}(k,l)}$) are then mutliplied by the i^{th} predominant signal, $X_{PS}^{(i)}(k,l)$, to produce the i^{th} column of the HOA representation ($\boldsymbol{c}_{\mathrm{VEC}}^{(i)}(k,l)$). The column vectors are then summed to derive the HOA representation of the vector-based signals. In this way, the HOA ceofficients are reconstructed for a frame.

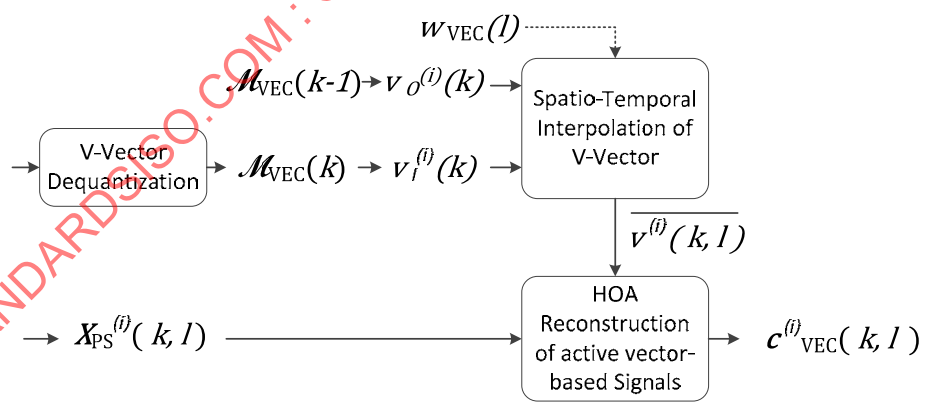


Figure 64 — Computation of vector-based signals

12.4.2.4.4.2 Spatio-temporal interpolation of V-vectors

An interpolation matrix $\overline{\boldsymbol{V}^{(i)}(k)} = [\overline{\boldsymbol{v}^{(i)}(k,1)}, \quad \overline{\boldsymbol{v}^{(i)}(k,2)}, \quad \ldots \quad \overline{\boldsymbol{v}^{(i)}(k,L)},]$ is first computed for each index $i \in \mathcal{I}_{\mathrm{VEC}}(k) \cup \mathcal{I}_{\mathrm{VEC}}(k-1)$ of a vector-based signal that is active in the k-th or $(k-1)$-th frame. Its columns $\overline{\boldsymbol{v}^{(i)}(k,l)}$ represent for each sample $1 \leq l \leq L$ of a frame an interpolated vector defined by:

$$\overline{\boldsymbol{v}^{(i)}(k,l)} = w_{\mathrm{I}}^{(i)}(l)\boldsymbol{v}_{\mathrm{I}}^{(i)}(k) + \left(1 - w_{I}^{(i)}(l)\right)\boldsymbol{v}_{\mathrm{O}}^{(i)}(k)$$

In the equation above, $\boldsymbol{v}_{\mathrm{I}}^{(i)}(k)$ and $\boldsymbol{v}_{\mathrm{O}}^{(i)}(k)$ denote the vectors between which the interpolation takes place, defined by:

$$\boldsymbol{v}_{\mathrm{I}}^{(i)}(k) = \begin{cases} \boldsymbol{v}^{(i)}(k) & \text{if } i \in \mathcal{I}_{\mathrm{VEC}}(k) \\ \boldsymbol{0} & \text{else} \end{cases}$$

$$\boldsymbol{v}_{\mathrm{O}}^{(i)}(k) = \begin{cases} \boldsymbol{v}^{(i)}(k-1) & \text{if } i \in \mathcal{I}_{\mathrm{VEC}}(k-1) \\ \boldsymbol{0} & \text{else} \end{cases}$$

where 0 denotes the zero vector. This is shown in Figure 65.

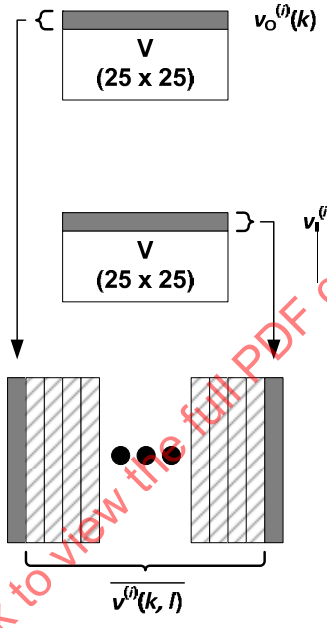


Figure 65 — Interpolation of V-vectors

Further, the interpolation function $w_{\mathrm{I}}^{(i)}(l)$ is defined by:

$$w_{\mathrm{I}}^{(i)}(l) = \begin{cases} w_{\mathrm{VEC}}(l) & \text{if } i \in \mathcal{I}_{\mathrm{VEC}}(k) \ \&\ \text{if } i \in \{\mathcal{I}_{\mathrm{VEC}}(k-1) \cup \mathcal{I}_{\mathrm{DIR}}(k-1)\} \\ 1 & \text{if } i \in \mathcal{I}_{\mathrm{VEC}}(k) \ \&\ \text{if } i \notin \{\mathcal{I}_{\mathrm{VEC}}(k-1) \cup \mathcal{I}_{\mathrm{DIR}}(k-1)\} \\ w_{\mathrm{DIR}}(l) & \text{if } i \in \mathcal{I}_{\mathrm{VEC}}(k-1) \ \&\ \text{if } i \in \mathcal{I}_{\mathrm{DIR}}(k) \\ 0 & \text{else} \end{cases}$$

where

$$w_{\mathrm{VEC}}(l) = \begin{cases} \dfrac{l-1}{L_{\mathrm{IP}}-1} & \text{for } 1 \le l \le L_{\mathrm{IP}} \text{ if } \textbf{SpatialInterpolationMethod} = 0 \\ \dfrac{1}{2}\left[1 - \cos\left(2\pi\dfrac{l-1}{2L_{\mathrm{IP}}}\right)\right] & \text{for } 1 \le l \le L_{\mathrm{IP}} \text{ if } \textbf{SpatialInterpolationMethod} = 1 \\ 1 & \text{for } L_{\mathrm{IP}}+1 \le l \le L \end{cases}$$

and where, L_{IP} is indicated by the variable CodedSpatialInterpolationTime as given by Table 209.

The HOA representation $\boldsymbol{C}_{\mathrm{VEC}}^{(i)}(k) = \left[\boldsymbol{c}_{\mathrm{VEC}}^{(i)}(k,1) \quad \boldsymbol{c}_{\mathrm{VEC}}^{(i)}(k,2) \quad \ldots \quad \boldsymbol{c}_{\mathrm{VEC}}^{(i)}(k,L)\right]$ for each i^{th} vector-based signal $\boldsymbol{x}_{\mathrm{PS},i}(k), i \in \mathcal{I}_{\mathrm{VEC}}(k) \cup \mathcal{I}_{\mathrm{VEC}}(k-1)$, is a matrix of dimension $(N+1)^2 \cdot L$, whose columns are given by:

$$\boldsymbol{c}_{\mathrm{VEC}}^{(i)}(k,l) = \overline{\boldsymbol{v}^{(i)}(k,l)}\, x_{\mathrm{PS},i}(k,l)$$

The complete HOA representation of the vector-based signals can be computed by summing the HOA contribution from each individual vector-based signal as follows:

$$\tilde{\boldsymbol{C}}_{\mathrm{VEC}}(k) = \sum_{i \in \mathcal{I}_{\mathrm{VEC}}(k) \cup \mathcal{I}_{\mathrm{VEC}}(k-1)} \boldsymbol{C}_{\mathrm{VEC}}^{(i)}(k)$$

In the case, that E, the CodedVVecLength, has a value of 1, the following operations have to be performed.

— If there are coefficient sequences of the ambient HOA component that are explicitly additionally transmitted and faded in during the k-th frame (of which the indices are contained in the set $\mathcal{I}_{\mathrm{E}}(k)$), and the parameter NewChannelTypeOne[i] equals zero, then the respective coefficient sequences of the HOA representation $\tilde{\boldsymbol{C}}_{\mathrm{VEC}}(k)$ have to be faded out using the fade-out part of the window w_{DIR}. The respective V-vector elements in $\boldsymbol{v}_{\mathrm{I}}^{(i)}(k)$ are discarded from the spatio-temporal interpolation in the following frame *k+1* by setting them to zero.

— If there are coefficient sequences of the ambient HOA component that are explicitly additionally transmitted and faded out during the k-th frame (of which the indices are contained in the set $\mathcal{I}_{\mathrm{D}}(k)$), then the respective coefficient sequences of the HOA representation $\tilde{\boldsymbol{C}}_{\mathrm{VEC}}(k)$ have to be faded in using the fade-in part of the window w_{DIR}.

— Hence, the final HOA representation of the vector-based signals is obtained by

$$c_{\mathrm{VEC,n}}(k,L) = \begin{cases} \tilde{c}_{\mathrm{VEC,n}}(k,L) \cdot w_{\mathrm{DIR}}(l) & \text{if } n \in \mathcal{I}_{\mathrm{D}}(k) \wedge E = 1 \\ \tilde{c}_{\mathrm{VEC,n}}(k,L) \cdot w_{\mathrm{DIR}}(L+l) & \text{if } n \in \mathcal{I}_{\mathrm{E}}(k) \wedge E = 1 \\ \tilde{c}_{\mathrm{VEC,n}}(k,L) & \text{else} \end{cases} \qquad \text{for } 1 \le l \le L\,.$$

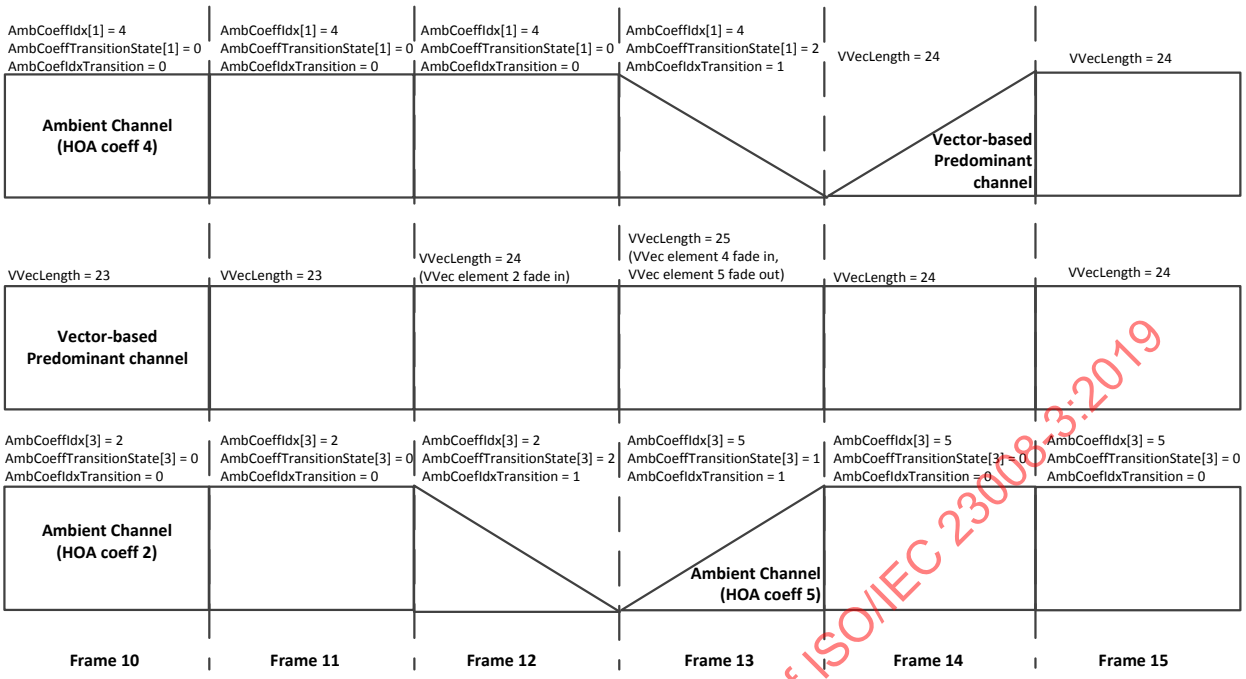


Figure 66 — Example fade-in and fade-out of components

The above example in Figure 66 shows how the ambient HOA coefficient channels 4, 2, and 5 and a vector-based predominant-signal channel are faded-in and faded-out as described above. The ambient HOA coefficient channel 4 undergoes a period of transition (fades out) during frame 13 while the elements of a vector in the vector-based predominant-signal channel fade in during frame 14 to replace the ambient HOA coefficient 4 in the ambient channel during decoding of the bitstream. Figure 66 also shows that the elements of the vector change in frames 12, 13 and 14, and the V-vector length changes during the frames. The ambient HOA coefficient 2 undergoes a transition during frame 12 (fade out). The ambient HOA coefficient 5 undergoes a transition (fades in) during frame 13 to replace the ambient HOA coefficient 2 in the ambient channel during decoding of the bitstream. The annotated parameter exemplifiy the decoder state for a 4th order HOA content (25 HOA coefficients).

During the above described periods of transition, the AmbCoeffIdxTransition flag specified in Table 196 indicates a transition of the respective channel. Given the previous state of the AmbCoeffTransitionState, the AmbCoeffIdxTransition flag indicates which ambient HOA coefficient should be either faded-in or faded-out.

The V-vector length is specified in accordance with three different modes. If the parameter CodedVVecLength is set to 1, the V-vector is specified using the reduced number of V-vector elements (VVecLength) when energy from that element has been fully incorporated into the underlying ambient HOA coefficient. Therefore, for reducing the number of transmitted V-vector elements, only the elements of the HOA soundfield that are not encoded as ambient HOA coefficients may be transmitted. The overall number or the actual HOA coefficients of the ambient components can be dynamic to account for changes in the sound field. For the HOA frames in which an ambient HOA coefficient channel is faded-in or faded-out some additional fade processing is required.

The foregoing is represented by the following pseudo-code which describes how to process the reconstructed vector-based sound components when ambient HOA coefficients are in transition for the case when the CodedVVecLength = 1.

1) *Reconstructing newly introduced distinct components (if any)*

```
if exist(newTransportChannelPredSound) {
    for (l= 0; l<L; ++l) {
        c_VEC^(i)(k,l) = X_PS^(i)(k,l) * v_I^(i)(k)
    }
}
```

2) *Reconstructing continuous distinct components (if any) and apply spatio-temporal interpolation*

```
if exist(contTransportChannelPredSound) {
    for (l= 0; l<L; ++l) {
        c_VEC^(i)(k,l) += conj(v^(i)(k,l)) * X_PS^(i)(k,l);
    }
}
```

3) *If transitional ambient HOA coefficients are present in the frame, apply fade in/fade out to* $\boldsymbol{C}_{VEC}$

```
foreach n ∈ J_E(k) {
    for (l =0; l<L; ++l) {
        C̃_VEC,n(k,l) *= w_DIR(L + l);
    }
}

foreach n ∈ J_D(k) {
    for (l =0; l<L; ++l) {
        C̃_VEC,n(k,l) *= w_DIR(l);
    }
}
}
```

The pseudo-code above describes how to process the reconstructed vector-based sound components $\tilde{\boldsymbol{C}}_{\mathrm{VEC}}$ when an ambient HOA coefficient is in transition in case E has a value of 1.

The first section provides pseudo-code for reconstructing newly introduced distinct components when present. The second section provides pseudo-code for reconstructing continuous distinct components when present and applying spatio-temporal interpolation. In section three of the pseudo-code, there are crossfade-in and crossfade-out operations performed on the V-vector interpolation buffer to fade-in new HOA coefficients and fade-out old HOA coefficients.

Referring to Figure 66, in frame 10 and 11 there are two ambient channels and one vector-based predominant sound channel. In frames 10 and 11, the V-vector does not include the V-vector element for the ambient HOA coefficients specified in the ambient channels because the ambient HOA coefficients specified in the ambient channels were directly coded. In frame 12, the ambient HOA coefficient transmitted in the third transport channel (HOA coefficient 2) is being faded-out. In frame 13, the ambient HOA coefficient specified in the top transport channel (HOA coefficient 4) is being faded-out and the ambient HOA coefficient specified in the third transport channel (HOA coefficient 5) is being faded-in. The bitstream signals the events when an ambient HOA coefficient specified in an ambient channel is faded-out or faded-in. When decoding the bitstream, state information is maintained via the AmbCoeffTransitionMode parameter for each ambient HOA coefficient specified in one of the transport channels. Referring back to the first transport channel, the state information, at frame 10, indicates that the AmbCoeffTransitionState[i] element is set to zero, where i denotes the index of the HOA Transport

Channel in the bitstream. The coefficient index of the ambient HOA coefficient signalled in the HOA transport channel i is defined via AmbCoeffIdx[i]. When the same HOA coefficient 4 is specified in the previous frame 9 (not shown in Figure 66), the AmbCoeffIdxTransition syntax element signalled with a value of zero. As a result, the V-vector may have a total of 23 elements (for a 4th order HOA representation). Vector elements [1, 3, 5 to 25] are specified, omitting the elements that correspond to the ambient HOA coefficients having an index of 2 and 4. Given that no transitions occur before frame 12, the same state information is maintained for the ambient HOA channels during frame 11.

At frame 12, the ambient HOA coefficient having an index of 2 is faded-out. The AmbCoeffIdxTransition is signalled for the corresponding transport channel with a value of one (indicating the transition). The internal state element AmbCoeffTransitionState[3] is updated to a value two. As a result of the state change from no transition to fade-out, the V-vector element is added to the V-vector of the predominant sound corresponding to the ambient HOA coefficient having an index of 2.

In frame 13, two transitions occur, one for fading-out ambient HOA coefficient 4 and another for fading in ambient HOA coefficient 5. The state information may be updated to indicate that the ambient HOA coefficients having an index of 5 is faded-in (e.g. AmbCoeffTransitionState[3] = 1). The state information is also updated to indicate that the ambient HOA coefficient is fadein-out (e.g. AmbCoeffTransitionState[1] = 2).

Given that there are three transport channels as depicted in Figure 66, two of which are ambient HOA channels undergoing a transition, the V-vector includes all 25 of the V-vector elements of a 4th order HOA representation.

At frame 14, a vector-based predominant channel is faded-in to replace the ambient channel in the first transport channel. Given that there are no transitions of ambient HOA coefficients, the V-vectors includes 24 elements, and that the vector element corresponding to the ambient HOA coefficient having an index of 5 needs not be coded. During frame 14, the flag AmbCoeffIdxTransition will signal 0, indicating that the ambient HOA coefficient having an index of 5 (AmbCoeffIdx[3] = 5) is not in transition (AmbCoeffTransitionState[3] = 0).

12.4.2.4.5 Compose complete predominant sound component

The complete predominant sound HOA component, $\boldsymbol{C}_{\mathrm{PS}}(k)$, is obtained as the sum of the HOA component $\boldsymbol{C}_{\mathrm{DIR}}(k)$ of the directional signals, the HOA component $\boldsymbol{C}_{\mathrm{PD}}(k)$ of the predicted directional signals and the HOA component of the vector-based signals $\boldsymbol{C}_{\mathrm{VEC}}(k)$ by:

$$\boldsymbol{C}_{\mathrm{PS}}(k) = \boldsymbol{C}_{\mathrm{DIR}}(k) + \boldsymbol{C}_{\mathrm{PD}}(k) + \boldsymbol{C}_{\mathrm{VEC}}(k)$$

Note that the HOA components $\boldsymbol{C}_{\mathrm{DIR}}(k)$, $\boldsymbol{C}_{\mathrm{PD}}(k)$, and $\boldsymbol{C}_{\mathrm{VEC}}(k)$ are zero in case they are not processed as defined in the ChannelSideInfoData (Table 193) for frame k.

The modified predominant sound HOA representation $\boldsymbol{C}_{\mathrm{PS,F}}(k)$ is computed from $\boldsymbol{C}_{\mathrm{PS}}(k)$ by fading in coefficient sequences with indices contained in the index set $\mathcal{I}_{\mathrm{E}}(k)$ and fading out coefficient sequences with indices contained in the index set $\mathcal{I}_{\mathrm{D}}(k)$. In particular, the individual samples $c_{\mathrm{PS,F},n}(k,l)$ of the modified predominant sound HOA representation $\boldsymbol{C}_{\mathrm{PS,F}}(k)$ are computed according to:

$$c_{\mathrm{PS,F},n}(k,l) = \begin{cases} c_{\mathrm{PS},n}(k,l) \cdot w_{\mathrm{DIR}}(l) & \text{if } n \in \mathcal{I}_{\mathrm{E}}(k) \\ c_{\mathrm{PS},n}(k,l) \cdot w_{\mathrm{DIR}}(L+l) & \text{if } n \in \mathcal{I}_{\mathrm{D}}(k) \\ c_{\mathrm{PS},n}(k,l) & \text{else} \end{cases} \quad \text{for } n = 1, \ldots, O, l = 1, \ldots, L.$$

12.4.2.5 Ambience synthesis

12.4.2.5.1 General

The ambient HOA component frame $\boldsymbol{C}_{\mathrm{AMB}}(k)$ is assumed to be composed according to:

$$\boldsymbol{C}_{\mathrm{AMB}}(k) = \begin{bmatrix} \boldsymbol{c}_{\mathrm{AMB},1}(k) \\ \boldsymbol{c}_{\mathrm{AMB},2}(k) \\ \vdots \\ \boldsymbol{c}_{\mathrm{AMB},O}(k) \end{bmatrix}$$

with

$$\boldsymbol{c}_{\mathrm{AMB},n}(k) = [c_{\mathrm{AMB},n}(k,1) \quad c_{\mathrm{AMB},n}(k,2) \quad \ldots \quad c_{\mathrm{AMB},n}(k,L)] \quad \text{for} \quad n = 1, \ldots, O.$$

The first O_{MIN} coefficient sequences of the ambient HOA component are computed as outlined in the following two subclauses. The sample values of the remaining higher-order coefficient sequences of the ambient HOA component are set according to:

$$c_{\mathrm{AMB},n}(k,l) = c_{\mathrm{I,AMB},n}(k,l) \quad \text{for} \quad O_{\mathrm{MIN}} < n \leq O.$$

By default the first O_{MIN} HOA coefficient sequences are reconstructed with the method outlined in subclause 12.4.2.8.2. If N_{MIN} is of value 1, an alternative synthesis method described in subclause 12.4.2.8.3 may be used. In this case the flag UsePhaseShiftDecorr signals which of the two processing methods shall be applied.

12.4.2.5.2 Spatial transform

The first O_{MIN} coefficient sequences of the ambient HOA component are obtained by:

$$\begin{bmatrix} \mathbf{c}_{\mathrm{AMB},1}(k) \\ \mathbf{c}_{\mathrm{AMB},2}(k) \\ \vdots \\ \mathbf{c}_{\mathrm{AMB},O_{\mathrm{MIN}}}(k) \end{bmatrix} = \boldsymbol{\Psi}_{\mathrm{MIN}} \cdot \begin{bmatrix} \mathbf{c}_{\mathrm{I,AMB},1}(k) \\ \mathbf{c}_{\mathrm{I,AMB},2}(k) \\ \vdots \\ \mathbf{c}_{\mathrm{I,AMB},O_{\mathrm{MIN}}}(k) \end{bmatrix}$$

where $\boldsymbol{\Psi}^{(N_{\mathrm{MIN}},N_{\mathrm{MIN}})}$ denotes the mode matrix of order N_{MIN} defined in Annex F.1.5 with respect to the predefined directions $\boldsymbol{\Omega}_n^{(N_{\mathrm{MIN}})}$, $n = 1, \ldots, O_{\mathrm{MIN}}$ defined in Tables F.2 - F.11. Note that the multiplication by the mode matrix represents the inverse spatial transform intended to invert the spatial transform applied in the encoder (see subclause C.5.3.3.3) for de-correlating the first O_{MIN} coefficient sequences of the ambient HOA component.

12.4.2.5.3 Phase-based transform

If the flag UsePhaseShiftDecorr is == 1, the following processing is applied to reconstruct the first four coefficient sequences of the ambient HOA component by:

$$\begin{bmatrix} \mathbf{c}_{\mathrm{AMB},1}(k) \\ \mathbf{c}_{\mathrm{AMB},2}(k) \\ \mathbf{c}_{\mathrm{AMB},3}(k) \\ \mathbf{c}_{\mathrm{AMB},4}(k) \end{bmatrix} = \begin{bmatrix} c(3) \cdot A_{+90}(k) + c(2) \cdot \left[\mathbf{c}_{\mathrm{I,AMB},1}(k) + \mathbf{c}_{\mathrm{I,AMB},2}(k)\right] \\ B_{+90}(k) + c(5) \cdot \left[\mathbf{c}_{\mathrm{I,AMB},1}(k) - \mathbf{c}_{\mathrm{I,AMB},2}(k)\right] + c(6) \cdot \mathbf{c}_{\mathrm{I,AMB},3}(k) \\ \mathbf{c}_{\mathrm{I,AMB},4}(k) \\ c(4) \cdot \left[\mathbf{c}_{\mathrm{I,AMB},1}(k) + \mathbf{c}_{\mathrm{I,AMB},2}(k)\right] - A_{+90}(k) \end{bmatrix}$$

with the coefficients c as defined in Table 210 and $A_{+90}(k)$ and $B_{+90}(k)$ are the frames of +90 degrees phase shifted signals A and B defined by:

$$A(k) = c(0) \cdot \left[\mathbf{c}_{\mathrm{I,AMB},1}(k) - \mathbf{c}_{\mathrm{I,AMB},2}(k)\right]$$

$$B(k) = c(1) \cdot \left[\mathbf{c}_{\mathrm{I,AMB},1}(k) + \mathbf{c}_{\mathrm{I,AMB},2}(k)\right]$$

Note that the phase shift operation introduces a delay of one frame. In order to avoid this delay in the decoder in the case that the flag UsePhaseShiftDecorr has a value of 1, it is assumed that the Channel Reassignment processing block (see subclause 12.4.2.3) provides the frame $\boldsymbol{C}_{\mathrm{I,AMB}}(k+1)$ instead of $\boldsymbol{C}_{\mathrm{I,AMB}}(k)$, which can be maintained by a respective modification at the spatial HOA encoding stage.

Table 210 — Coefficients for phase-based transform

n	c(n)
0	1,014 088 753 512 235 6
1	0,229 027 290 950 227 14
2	0,981 999 999 999 999 98
3	0,160 849 826 442 762 05
4	0,513 168 101 113 075 76
5	0,974 896 917 627 704 81
6	-0,880 208 333 333 333 37

12.4.2.6 Preliminary HOA composition

In the case of *single-layered* coding (indicated by SingleLayer==1 (see Table 188), the frame $\boldsymbol{C}_{\mathrm{PRE}}(k)$ of the preliminary decoded HOA representation is computed by:

$$\boldsymbol{C}_{\mathrm{PRE}}(k) = \boldsymbol{C}_{\mathrm{AMB}}(k) + \boldsymbol{C}_{\mathrm{PS}}(k) = \boldsymbol{C}_{\mathrm{AMB}}(k) + \boldsymbol{C}_{\mathrm{DIR}}(k) + \boldsymbol{C}_{\mathrm{PD}}(k) + \boldsymbol{C}_{\mathrm{VEC}}(k)$$

Additionally, the frame $\boldsymbol{C}_{\mathrm{PRE,F}}(k)$ of a modified version of the preliminary decoded HOA representation is computed by:

$$\boldsymbol{C}_{\mathrm{PRE,F}}(k) = \boldsymbol{C}_{\mathrm{AMB}}(k) + \boldsymbol{C}_{\mathrm{PS,F}}(k)$$

This modified HOA representation is assumed to be successively processed by the PAR decoder instead of the original version $\boldsymbol{C}_{\mathrm{PRE}}(k)$ to avoid signal discontinuities after performing on $\boldsymbol{C}_{\mathrm{PRE,F}}(k)$ the truncation and coefficient selection (see subclause 12.4.2.8.3).

In the case of *multiple layered* coding (indicated by SingleLayer==0 (see Table 188), the coefficient sequences $\boldsymbol{c}_{\mathrm{PRE},i}(k)$, $i = 1, \ldots, O$, of the preliminary decoded HOA representation $\boldsymbol{C}_{\mathrm{PRE}}(k)$ are computed by:

$$\boldsymbol{c}_{\mathrm{PRE},i}(k) = \begin{cases} \boldsymbol{c}_{\mathrm{AMB},i}(k) & \text{if } i \in \mathcal{I}_{\mathrm{U}}(k) \\ \boldsymbol{c}_{\mathrm{AMB},i}(k) + \boldsymbol{c}_{\mathrm{PS},i}(k) & \text{if } i \in \mathcal{I}_{\mathrm{E}}(k) \cup \mathcal{I}_{\mathrm{D}}(k). \\ \boldsymbol{c}_{\mathrm{PS},i}(k) & \text{else} \end{cases}$$

This means that the transmitted coefficient sequences with indices $i \in \mathcal{I}_{\mathrm{E}}(k) \cup \mathcal{I}_{\mathrm{D}}(k) \cup \mathcal{I}_{\mathrm{U}}(k)$ actually represent the original HOA representation instead of its ambient component. Hence, for the transmitted coefficient sequences, which are neither faded in nor faded out within the current frame, nothing has to be added to them. For the transmitted coefficient sequences that are faded in (or faded out) within the current frame, i.e. those with indices $i \in \mathcal{I}_{\mathrm{E}}(k) \cup \mathcal{I}_{\mathrm{D}}(k)$, the corresponding coefficient sequences of the predominant sound HOA representation $\boldsymbol{C}_{\mathrm{PS}}(k)$ are added, which are to be appropriately faded out (or faded in) at the predominant sound synthesis.

Similarly, the coefficient sequences $\boldsymbol{c}_{\mathrm{PRE,F},i}(k)$ of the modified version of the preliminary decoded HOA frame $\boldsymbol{C}_{\mathrm{PRE,F}}(k)$ are computed by:

$$\boldsymbol{c}_{\mathrm{PRE,F},i}(k) = \begin{cases} \boldsymbol{c}_{\mathrm{AMB},i}(k) & \text{if } i \in \mathcal{I}_{\mathrm{U}}(k) \\ \boldsymbol{c}_{\mathrm{AMB},i}(k) + \boldsymbol{c}_{\mathrm{PS,F},i}(k) & \text{if } i \in \mathcal{I}_{\mathrm{E}}(k) \cup \mathcal{I}_{\mathrm{D}}(k). \\ \boldsymbol{c}_{\mathrm{PS,F},i}(k) & \text{else} \end{cases}$$

If the number of actually used layers changes between two successive frames (i.e. if $M_{\mathrm{LAY}}(k) \neq M_{\mathrm{LAY}}(k-1)$), with the above computation there can occur, in the general case, a discontinuity in all coefficient sequences of the preliminary decoded HOA representation and its modified version between the $(k-1)$-th and k-th frame. One possible solution for this problem is to introduce an additional delay of one frame within the preliminary HOA composition and to fade out and fade in the coefficient sequences at the discontinuity.

12.4.2.7 Sub-band directional signals synthesis

12.4.2.7.1 General

The purpose of the sub-band directional signals synthesis is to approximate the non-transmitted coefficient sequences of the residual (i.e. ambient) HOA component by a composition of directional sub-band signals, which are predicted by a weighted/scaled sum of the transmitted coefficient sequences of the residual (i.e. ambient) HOA component, where the scaling is complex valued in general. In particular, each directional sub-band signal related to the j-th sub-band, $j \in \{1, \ldots, F\}$, is represented parametrically by complex valued prediction scaling factor matrices $\mathbf{A}(k,b)$ and tuple sets $\widetilde{\mathcal{M}}_{\mathrm{DIR}}(k,b)$ related to the b-th sub-band group ($b \in 1, \ldots, B$) which includes the j-th sub-band. Per sub-band group there are at most D_{SB} potential active direction trajectories, where the indices identifying the active direction trajectories for the b-th sub-band group are assumed to be contained in the set $\tilde{\mathcal{I}}_{\mathrm{DIR}}(k,b) \subseteq \{1, \ldots, D_{\mathrm{SB}}\}$. For each index $d \in \tilde{\mathcal{I}}_{\mathrm{DIR}}(k,b)$ of an active direction trajectory the respective direction is denoted by $\boldsymbol{\Omega}_{\mathrm{SB},d}(k,b)$, both of which are assumed to be contained as tuples in the set $\widetilde{\mathcal{M}}_{\mathrm{DIR}}(k,b)$, i.e.

$$\widetilde{\mathcal{M}}_{\mathrm{DIR}}(k,b) = \left\{ \left(d, \boldsymbol{\Omega}_{\mathrm{SB},d}(k,b) \right) \middle| \, d \in \tilde{\mathcal{I}}_{\mathrm{DIR}}(k,b) \right\}.$$

Note that the set $\tilde{\mathcal{I}}_{\mathrm{DIR}}(k,b)$ is assumed to consist of the first elements of the tuples of $\widetilde{\mathcal{M}}_{\mathrm{DIR}}(k,b)$, and can hence be computed from $\widetilde{\mathcal{M}}_{\mathrm{DIR}}(k,b)$.

Further note, that in the absence of predominant sound signals the ambient component corresponds to a "truncated" version of the original HOA representation. Truncation in this context means that the original HOA representation is approximated by only I of its total O coefficient sequences, i.e. by those that were transmitted within the I transport channels.

The detailed architecture of the sub-band directional signals synthesis is illustrated in Figure 67. The individual processing units to compute the frame $\widetilde{\boldsymbol{C}}_{\mathrm{D}}(k)$ of the HOA representation of the composition of all predicted sub-band directional signals will be described in the following.

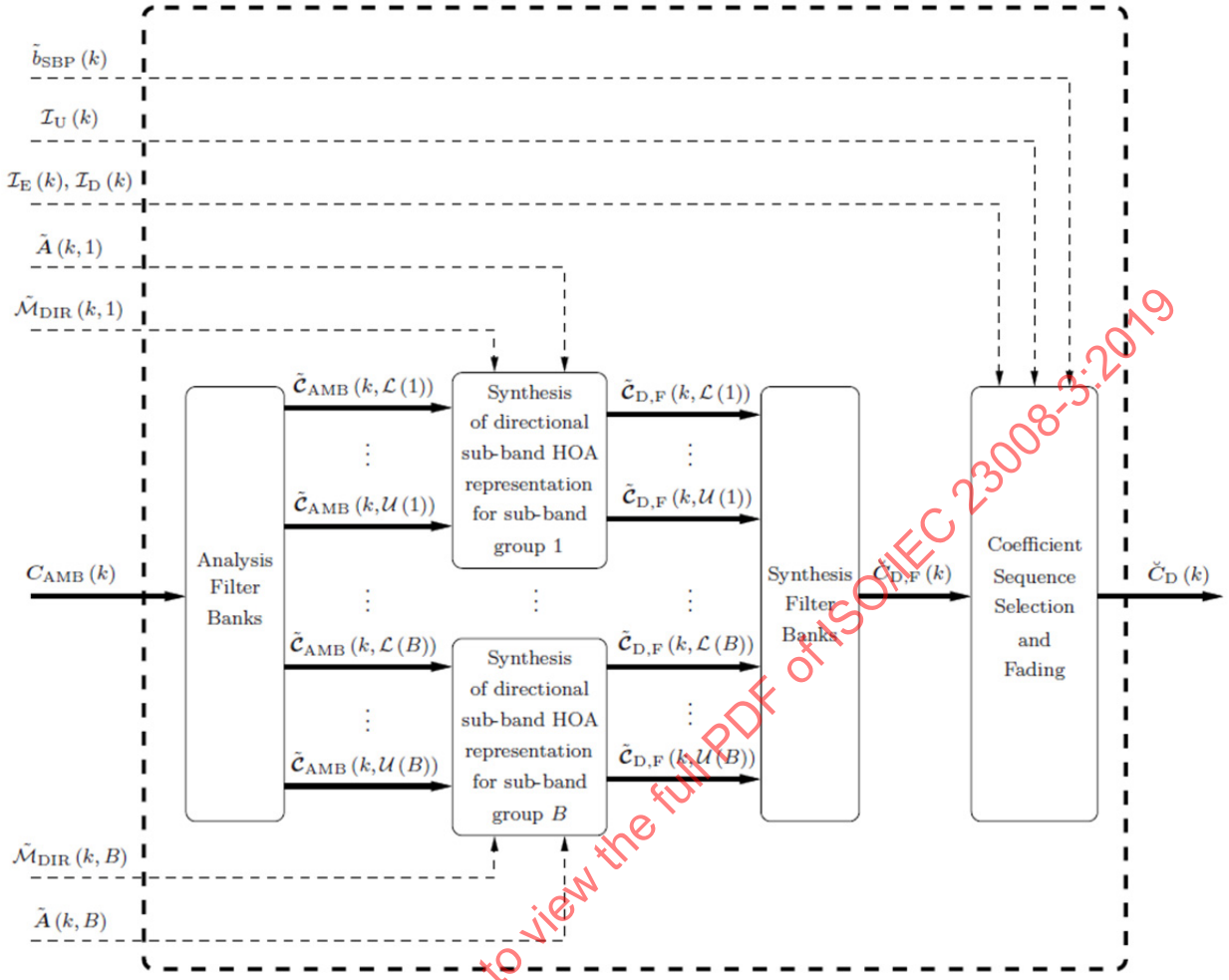


Figure 67 — Sub-band directional signals synthesis

12.4.2.7.2 Analysis filter banks

Each frame $\boldsymbol{c}_{\mathrm{AMB},n}(k)$, $n = 1, \ldots, O$, of an individual coefficient sequence of the ambient HOA representation $\boldsymbol{C}_{\mathrm{AMB}}(k)$ is first decomposed into frames of individual sub-band signals $\tilde{\boldsymbol{c}}_{\mathrm{AMB},n}(k,j)$, $j = 1, \ldots, F$. For each sub-band j, $j = 1, \ldots, F$, the frames of the sub-band signals of the individual HOA coefficient sequences are collected into the sub-band HOA representation $\widetilde{\boldsymbol{C}}_{\mathrm{AMB}}(k,j)$ as

$$\widetilde{\boldsymbol{C}}_{\mathrm{AMB}}(k,j) = \begin{bmatrix} \tilde{\boldsymbol{c}}_{\mathrm{AMB},1}(k,j) \\ \tilde{\boldsymbol{c}}_{\mathrm{AMB},2}(k,j) \\ \vdots \\ \tilde{\boldsymbol{c}}_{\mathrm{AMB},O}(k,j) \end{bmatrix} \quad \text{for} \quad j = 1, \ldots, F.$$

The filter bank shall be based on quadrature mirror filters (QMF) with a total of $F = 64$ sub-bands as defined in ISO/IEC 23003-1. Note that, in contrast to the HOA coefficient sequences $\boldsymbol{c}_{\mathrm{AMB},n}(k)$ their sub-band representations $\tilde{\boldsymbol{c}}_{\mathrm{AMB},n}(k,j)$ are, in general, complex valued. Further, the sub-band signals are decimated in time compared to the original time-domain signals by a factor of F. As a consequence, the number of samples in the frames $\tilde{\boldsymbol{c}}_{\mathrm{AMB},n}(k,j)$ is $L_{\mathrm{SB}} = L/F$. It is assumed that L is an integral multiple of F to assure that L_{SB} has a positive integer value.

A further important implementation issue is that the coefficient sequences of the ambient HOA representation with indices greater than O_{MAX} are assumed to be zero. Hence, the application of the analysis filters can be restricted to the HOA coefficient sequences $\boldsymbol{c}_{\mathrm{AMB},n}(k)$ with indices $n = 1, \ldots, O_{\mathrm{MAX}}$ only. The sub-band signal frames $\tilde{\boldsymbol{c}}_{\mathrm{AMB},n}(k,j)$ with indices $n = O_{\mathrm{MAX}}+1, \ldots, O$ can be set to zero.

12.4.2.7.3 Synthesis of directional sub-band HOA representation for individual sub-band groups

In order to avoid artifacts due to changes of the directions and prediction coefficients between successive frames, the computation of the directional sub-band HOA representation is based on the concept of overlap add in the sub-band domain. Hence, the HOA representation $\tilde{\boldsymbol{C}}_{\mathrm{D,F}}(k,j)$ of active directional sub-band signals related to the j-th sub-band, $j = 1, \ldots, F$, is computed as the sum of a faded out component and a faded in component:

$$\tilde{\boldsymbol{C}}_{\mathrm{D,F}}(k,j) = \tilde{\boldsymbol{C}}_{\mathrm{D,F,OUT}}(k,j) + \tilde{\boldsymbol{C}}_{\mathrm{D,F,IN}}(k,j)$$

To compute the two individual components, firstly, the instantaneous frame of all directional sub-band signals $\tilde{\boldsymbol{X}}_{\mathrm{I}}(k_1;k;j)$ for the j-th sub-band is computed by:

$$\tilde{\boldsymbol{X}}_{\mathrm{I}}(k_1;k;j) = \mathbf{A}(k_1,b)\tilde{\boldsymbol{C}}_{\mathrm{AMB}}(k,j) \qquad \text{for} \quad k_1 \in \{k, k+1\} \text{ and } \mathcal{L}(b) \le j \le \mathcal{U}(b).$$

using the ambient sub-band HOA representation $\tilde{\boldsymbol{C}}_{\mathrm{AMB}}(k,j)$ for the k-th frame and the prediction coefficients matrix $\mathbf{A}(k_1,b)$ for the b-th sub-band group including the j-th sub-band and for the (k_1)-th frame, where $k_1 \in \{k, k+1\}$. The (matrix) frame $\tilde{\boldsymbol{X}}_{\mathrm{I}}(k_1;k;j)$ is assumed to be composed of the (row) frames of the individual directional sub-band signals $\tilde{\boldsymbol{x}}_{\mathrm{I},d}(k_1;k;j)$, $d = 1, \ldots, D_{\mathrm{SB}}$, according to:

$$\tilde{\boldsymbol{X}}_{\mathrm{I}}(k_1;k;j) = \begin{bmatrix} \tilde{\boldsymbol{x}}_{\mathrm{I},1}(k_1;k;j) \\ \vdots \\ \tilde{\boldsymbol{x}}_{\mathrm{I},D_{\mathrm{SB}}}(k_1;k;j) \end{bmatrix}$$

Note that all elements of the frame $\boldsymbol{x}_{\mathrm{I},d}(k_1;k;j)$ of a directional signal are zero if the corresponding direction is not active, i.e., if the corresponding directional trajectory index d is not contained in the set $\tilde{\jmath}_{\mathrm{DIR}}(k,b)$.

Next, the instantaneous sub-band HOA representation $\tilde{\boldsymbol{C}}^{(d)}_{\mathrm{D,F,I}}(k_1;k;j)$ of the active directional sub-band signal $\tilde{\boldsymbol{x}}_{\mathrm{I},d}(k_1;k;j)$ with direction index $d \in \tilde{\jmath}_{\mathrm{DIR}}(k,b)$ with respect to the direction $\boldsymbol{\Omega}_{\mathrm{SB},d}(k,b)$ is obtained as:

$$\tilde{\boldsymbol{C}}^{(d)}_{\mathrm{D,F,I}}(k_1;k;j) = \boldsymbol{\psi}\left(\boldsymbol{\Omega}_{\mathrm{SB},d}(k,b)\right)\tilde{\boldsymbol{x}}_{\mathrm{I},d}(k_1;k;j)$$

where $\boldsymbol{\psi}\left(\boldsymbol{\Omega}_{\mathrm{SB},d}(k,b)\right) \in \mathbb{R}^{O}$ denotes the mode vector with respect to the direction $\boldsymbol{\Omega}_{\mathrm{SB},d}(k,b)$ which is computed as described in Annex F.1.5.

The matrices $\tilde{\boldsymbol{C}}_{\mathrm{D,F,OUT}}(k,j)$, $\tilde{\boldsymbol{C}}_{\mathrm{D,F,IN}}(k,j)$, and $\tilde{\boldsymbol{C}}^{(d)}_{\mathrm{D,F,I}}(k_1;k;j)$ are structured from their samples as follows

$$\tilde{\boldsymbol{C}}_{\mathrm{D,F,OUT}}(k,j) = \begin{bmatrix} \tilde{c}_{\mathrm{D,F,OUT},1}(k,j;1) & \ldots & \tilde{c}_{\mathrm{D,F,OUT},1}(k,j;L_{\mathrm{SB}}) \\ \vdots & \ddots & \vdots \\ \tilde{c}_{\mathrm{D,F,OUT},O}(k,j;1) & \ldots & \tilde{c}_{\mathrm{D,F,OUT},O}(k,j;L_{\mathrm{SB}}) \end{bmatrix} \in \mathbb{R}^{O \times L_{\mathrm{SB}}}$$

$$\widetilde{\boldsymbol{C}}_{\mathrm{D,F,IN}}(k,j) = \begin{bmatrix} \tilde{c}_{\mathrm{D,F,IN},1}(k,j;1) & \dots & \tilde{c}_{\mathrm{D,F,IN},1}(k,j;L_{\mathrm{SB}}) \\ \vdots & \ddots & \vdots \\ \tilde{c}_{\mathrm{D,F,IN},O}(k,j;1) & \dots & \tilde{c}_{\mathrm{D,F,IN},O}(k,j;L_{\mathrm{SB}}) \end{bmatrix} \in \mathbb{R}^{O\times L_{\mathrm{SB}}}$$

$$\widetilde{\boldsymbol{C}}^{(d)}_{\mathrm{D,F,I}}(k_1;k;j) = \begin{bmatrix} \tilde{c}^{(d)}_{\mathrm{D,F,I},1}(k_1;k;j;1) & \dots & \tilde{c}^{(d)}_{\mathrm{D,F,I},1}(k_1;k;j;L_{\mathrm{SB}}) \\ \vdots & \ddots & \vdots \\ \tilde{c}^{(d)}_{\mathrm{D,F,I},O}(k_1;k;j;1) & \dots & \tilde{c}^{(d)}_{\mathrm{D,F,I},O}(k_1;k;j;L_{\mathrm{SB}}) \end{bmatrix} \in \mathbb{R}^{O\times L_{\mathrm{SB}}}$$

the sample values of the faded out and faded in components of the HOA representation of active directional sub-band signals are finally determined for $1 \le l \le L_{\mathrm{SB}}$, $1 \le j \le F$ and $\mathcal{L}(b) \le j \le \mathcal{U}(b)$ by:

$$\tilde{c}_{\mathrm{D,F,OUT},n}(k,j;l) = \sum_{d\in\tilde{\jmath}_{\mathrm{DIR}}(k,b)} \tilde{c}^{(d)}_{\mathrm{D,F,I},n}(k;k;j;l)\cdot w_{\mathrm{SB}}(L_{\mathrm{SB}}+l)$$

$$\tilde{c}_{\mathrm{D,F,IN},n}(k,j;l) = \sum_{d\in\tilde{\jmath}_{\mathrm{DIR}}(k+1,b)} \tilde{c}^{(d)}_{\mathrm{D,F,I},n}(k+1;k;j;l)\cdot w_{\mathrm{SB}}(l)$$

where the vector:

$$\mathbf{w}_{\mathrm{SB}} = [w_{\mathrm{SB}}(1) \quad w_{\mathrm{SB}}(2) \quad \dots \quad w_{\mathrm{SB}}(2L_{\mathrm{SB}})]^T \in \mathbb{R}^{2L_{\mathrm{SB}}}$$

represents an overlap add (periodic Hann) window function to be applied on the sub-band signals, of which the elements are defined by

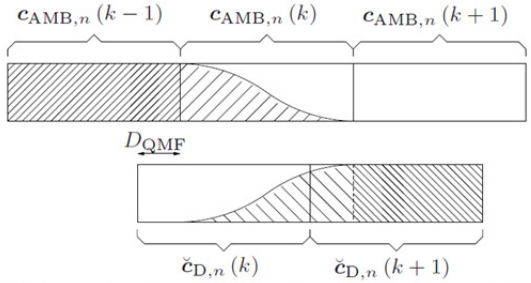
$$w_{\mathrm{SB}}(l) = \frac{1}{2}\left[1-\cos\left(2\pi\frac{l-1}{2L_{\mathrm{SB}}}\right)\right]$$

12.4.2.7.4 Synthesis filter banks

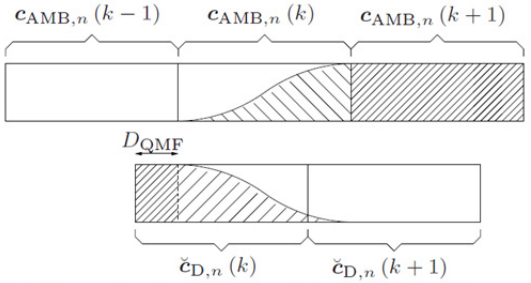
The individual time domain coefficient sequences $\breve{\boldsymbol{c}}_{\mathrm{D,F},n}(k), n = 1,\dots,O$, of the HOA representation $\breve{\boldsymbol{C}}_{\mathrm{D,F}}(k)$ of the composition of all predicted sub-band directional signals are synthesized from the corresponding sub-band coefficient sequences $\tilde{\boldsymbol{c}}_{\mathrm{D,F},n}(k,j), j = 1,\dots,F$ by the synthesis filter banks. Note that the synthesized time domain coefficient sequences have a delay of $D_{\mathrm{QMF}} = 577$ samples due to the successive application of the QMF based analysis and synthesis filter banks, which is expressed by the breve symbol (˘) above the variables.

12.4.2.7.5 Coefficient sequence selection and fading

In the final step of synthesis, the preliminary computed HOA representation $\breve{\boldsymbol{C}}_{\mathrm{D,F}}(\breve{k})$ of the composition of all predicted sub-band directional signals is modified to have only contributions for those coefficient sequences, which have not been explicitly transmitted for the ambient HOA component. Further, for those coefficient sequences of the ambient HOA component that are explicitly additionally transmitted and faded in (or faded out), the respective coefficient sequences of the preliminary HOA representation $\breve{\boldsymbol{C}}_{\mathrm{D,F}}(k)$ of the composition of all predicted sub-band directional signals have to be modified by fading them out (or fading them in), respectively. Due to the delay between $\boldsymbol{C}_{\mathrm{AMB}}(k)$ and $\breve{\boldsymbol{C}}_{\mathrm{D,F}}(k)$ of D_{QMF} samples the fading of the coefficient sequences of $\breve{\boldsymbol{C}}_{\mathrm{D,F}}(k)$ is performed across frame boundaries, as illustrated in Figure 68. The modified HOA representation of the composition of all predicted sub-band directional signals is denoted by $\breve{\boldsymbol{C}}_{\mathrm{D}}(k)$ with its coefficient sequences $\breve{\boldsymbol{c}}_{\mathrm{D},n}(k), n = 1,\dots,O$.



(a) Example of a faded coefficient sequence $\check{c}_{\mathrm{D},n}$ of the HOA representation of the composition of all predicted sub-band directional signals in the case the respective coefficient sequence $c_{\mathrm{AMB},n}$ of the ambient HOA component is faded out, i.e. if $n \in \mathcal{I}_{\mathrm{U}}(k-1) \wedge n \in \mathcal{I}_{\mathrm{D}}(k) \wedge n \notin (\mathcal{I}_{\mathrm{U}}(k+1) \cup \mathcal{I}_{\mathrm{E}}(k+1) \cup \mathcal{I}_{\mathrm{D}}(k+1))$



(b) Example of a faded coefficient sequence $\check{c}_{\mathrm{D},n}$ of the HOA representation of the composition of all predicted sub-band directional signals in the case the respective coefficient sequence $c_{\mathrm{AMB},n}$ of the ambient HOA component is faded in, i.e. if $n \notin (\mathcal{I}_{\mathrm{U}}(k-1) \cup \mathcal{I}_{\mathrm{E}}(k-1) \cup \mathcal{I}_{\mathrm{D}}(k-1)) \wedge n \in \mathcal{I}_{\mathrm{E}}(k) \wedge n \in \mathcal{I}_{\mathrm{U}}(k+1)$

Figure 68 — Illustration of faded coefficient sequences of $\check{\boldsymbol{c}}_{\mathrm{D}}(k)$

In the case that a coefficient sequence $\boldsymbol{c}_{\mathrm{AMB},n}$ of the ambient HOA component is faded out in the k-th frame (i.e. $n \in \mathcal{I}_{\mathrm{D}}(k)$) as illustrated in Figure 68a, the fade in of the coefficient sequence $\check{\boldsymbol{c}}_{\mathrm{D},n}$ in the k-th frame begins D_{QMF} samples later, where in particular the fading in is finished only at the D_{QMF}-th sample of the $(k+1)$-th frame.

Similarly, in the case that a coefficient sequence $\boldsymbol{c}_{\mathrm{AMB},n}$ of the ambient HOA component is faded in in the k-th frame (i.e. $n \in \mathcal{I}_{\mathrm{E}}(k)$) as illustrated in Figure 68b, the fade out of the coefficient sequence $\check{\boldsymbol{c}}_{\mathrm{D},n}$ in the k-th frame begins D_{QMF} samples later, where in particular the fading out is finished only at the D_{QMF}-th sample of the $(k+1)$-th frame.

Finally, it has to be considered that a fade in or fade out of the coefficient sequences $\check{\boldsymbol{c}}_{\mathrm{D,F},n}(k)$ of the HOA representation of the composition of all predicted sub-band directional signals is only required if it is not already present, resulting from overlap-add processing.

In the case that $\tilde{b}_{\mathrm{SBP}}(k-1) = 0$ and $\tilde{b}_{\mathrm{SBP}}(k) = 1$, there is already a fade in within each of the k-th frames $\check{\boldsymbol{c}}_{\mathrm{D,F},n}(k), n = 1, \ldots, O$, such that is not necessary to apply an additional fade in. Similarly, in the case that $\tilde{b}_{\mathrm{SBP}}(k-1) = 1$ and $\tilde{b}_{\mathrm{SBP}}(k) = 0$, there is already a fade out within each of the k-th frames $\check{\boldsymbol{c}}_{\mathrm{D,F},n}(k), n = 1, \ldots, O$, and hence it is not necessary to apply an additional fade out. Altogether, the computation of the sample values $\check{c}_{\mathrm{D},n}(k,l), n = 1, \ldots, O, l = 1, \ldots, L$, of the coefficient sequences of the HOA representation of the composition of all predicted sub-band directional signals is formally expressed by:

$$\check{c}_{\mathrm{D},n}(k,l) = \check{c}_{\mathrm{D,F},n}(k,l) \cdot \begin{cases} w_{\mathrm{DIR}}(l + L - D_{\mathrm{QMF}}) & \text{if } 1 \le l \le D_{\mathrm{QMF}} \wedge n \in \mathcal{I}_{\mathrm{D}}(k-1) \wedge \tilde{b}_{\mathrm{SBP}}(k-2) = 1 \\ w_{\mathrm{DIR}}(l + 2L - D_{\mathrm{QMF}}) & \text{if } 1 \le l \le D_{\mathrm{QMF}} \wedge n \in \mathcal{I}_{\mathrm{E}}(k-1) \wedge \tilde{b}_{\mathrm{SBP}}(k-1) = 1 \\ 1 & \text{if } 1 \le l \le D_{\mathrm{QMF}} \wedge \{n \notin \mathcal{I}_{\mathrm{U}}(k-1) \cup \mathcal{I}_{\mathrm{E}}(k-1) \cup \mathcal{I}_{\mathrm{D}}(k-1) \vee (n \in \mathcal{I}_{\mathrm{D}}(k-1) \wedge \tilde{b}_{\mathrm{SBP}}(k-2) = 0) \vee (n \in \mathcal{I}_{\mathrm{E}}(k-1) \wedge \tilde{b}_{\mathrm{SBP}}(k-1) = 0)\} \\ w_{\mathrm{DIR}}(l - D_{\mathrm{QMF}}) & \text{if } D_{\mathrm{QMF}} < l \le L \wedge n \in \mathcal{I}_{\mathrm{D}}(k) \wedge \tilde{b}_{\mathrm{SBP}}(k-1) = 1 \\ w_{\mathrm{DIR}}(l + L - D_{\mathrm{QMF}}) & \text{if } D_{\mathrm{QMF}} < l \le L \wedge n \in \mathcal{I}_{\mathrm{E}}(k) \wedge \tilde{b}_{\mathrm{SBP}}(k) = 1 \\ 1 & \text{if } D_{\mathrm{QMF}} < l \le L \wedge \{n \notin \mathcal{I}_{\mathrm{U}}(k) \cup \mathcal{I}_{\mathrm{E}}(k) \cup \mathcal{I}_{\mathrm{D}}(k) \vee (n \in \mathcal{I}_{\mathrm{D}}(k) \wedge \tilde{b}_{\mathrm{SBP}}(k-1) = 0) \vee (n \in \mathcal{I}_{\mathrm{E}}(k) \wedge \tilde{b}_{\mathrm{SBP}}(k) = 0)\} \\ 0 & \text{else} \end{cases}$$

12.4.2.8 Parametric ambience replication (PAR) decoder

12.4.2.8.1 General

The parametric ambience replication (PAR) decoder, as illustrated in Figure 69, replicates an ambient HOA component $\boldsymbol{C}_{\mathrm{PA}}(k)$ to complement the missing ambience in the preliminary reconstructed HOA component $\boldsymbol{C}_{\mathrm{PRE}}(k)$. The replicated ambient component is created in the sub-band domain, where its sub-band representation $\boldsymbol{\mathcal{C}}_{\mathrm{PA}}(k,j)$ related to the j-th sub-band is assumed to be of order $N_{\mathrm{PAR}}(g)$ depending on the corresponding g-th sub-band group $g = 1, \dots, G$. The orders $N_{\mathrm{PAR}}(g)$ for each sub-band group $g = 1, \dots, G$ are specified in subclause 12.4.1.2.3. The sub-band representation $\boldsymbol{\mathcal{C}}_{\mathrm{PA}}(k,j)$ is represented and created by means of $O_{\mathrm{PAR}}(g) = (N_{\mathrm{PAR}}(g) + 1)^2$ virtual loudspeaker sub-band signals $\boldsymbol{\mathcal{X}}_{\mathrm{PA}}(k,j)$ at directions $\boldsymbol{\Omega}_d^{(N_{\mathrm{PAR}}(g))}$, $d = 1, \dots, O_{\mathrm{PAR}}(g)$, defined in the tables in Annexes F.2 to F.11. These upmix sub-band signals are computed as a mixture of the sub-band signals $\boldsymbol{\mathcal{X}}_{\mathrm{DEC}}(k,j)$, which are themselves created by de-correlation filters from the virtual loudspeaker sub-band signals representing a so-called truncated and sparse sub-band HOA representation $\boldsymbol{\mathcal{C}}_{\mathrm{SP}}(k,j)$. The latter is obtained from the sub-band HOA representation $\boldsymbol{\mathcal{C}}_{\mathrm{PRE}}(k,j)$ by reducing its order to $N_{\mathrm{PAR}}(g)$ and setting all coefficient sequences to zero which are zero in the intermediate representation of the ambient HOA component $\mathrm{C}_{\mathrm{I,AMB}}(\mathrm{k})$. The number of de-correlated sub-band signals to be mixed for the creation of each upmix sub-band signal is allowed to vary over time according to the values of NumOfDecorrSigs PerParSubbandTable in order to adapt to the diffuseness of the ambient HOA component to be replicated. This number, denoted by $N_{\mathrm{SIG}}(k,g)$ specified in subclause 12.4.1.16, offers the possibility to control the amount of side information required to code the mixing matrices $\boldsymbol{M}_{\mathrm{PAR}}(k,g)$ for the individual sub-band groups $g = 1, \dots, G$. Further, for $N_{\mathrm{SIG}}(k,g) < O_{\mathrm{PAR}}(g)$ the mixing uses de-correlated sub-band signals obtained from virtual loudspeaker signals $\boldsymbol{\mathcal{X}}_{\mathrm{SP}}(k,j)$ at directions in the neighbourhood of the direction of the upmix signal. This operation prevents that directional components of the truncated and sparse sub-band HOA representation $\boldsymbol{\mathcal{C}}_{\mathrm{SP}}(k,j)$ are undesirably spatially distributed over all directions for the replication of the ambient HOA component. An additional aspect is that for each number $N_{\mathrm{SIG}}(k,g)$ and each individual upmix sub-band signal it is specified in Table F.40, which de-correlated sub-band signals have to be mixed. In order to decrease the mutual correlation between each group of de-correlated sub-band signals to be mixed, the assignment of the virtual loudspeaker signals to the de-correlation filters is adapted to the choice of de-correlated sub-band signals. This assignment is expressed through the permutation matrices $\boldsymbol{P}_{\mathrm{PAR}}(k,g)$ for the individual sub-band groups $g = 1, \dots, G$. The individual processing units of the PAR decoder to compute the frame $\boldsymbol{C}_{\mathrm{PA}}(k)$ of the replicated ambient HOA component is described in Figure 69.

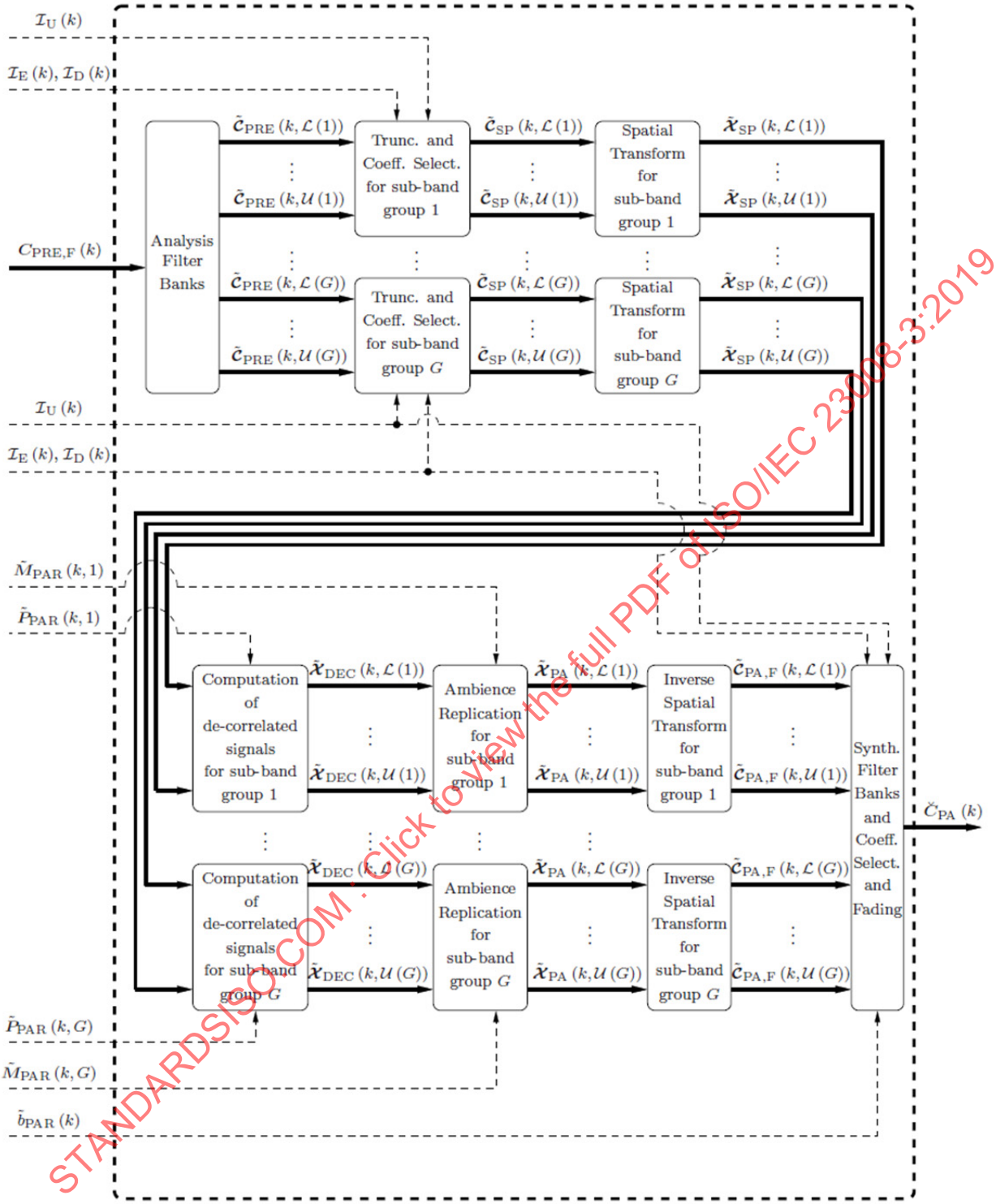


Figure 69 — PAR decoder

12.4.2.8.2 Analysis filter banks

Each frame $\boldsymbol{c}_{\mathrm{PRE,F},n}(k), n = 1, \ldots, O_{\mathrm{PAR,MAX}}$, of the first

$$O_{\mathrm{PAR,MAX}} = \max_{g} O_{\mathrm{PAR}}(g)$$

coefficient sequences of the modified version $\boldsymbol{C}_{\mathrm{PRE,F}}(k)$ of the preliminary decoded HOA representation is first decomposed into frames of individual sub-band signals $\tilde{\boldsymbol{c}}_{\mathrm{PRE},n}(k,j), j = 1, \ldots, F$. For each sub-band $j = 1, \ldots, F$, the frames of the sub-band signals of the individual HOA coefficient sequences are collected into the sub-band HOA representation $\tilde{\boldsymbol{C}}_{\mathrm{PRE}}(k,j)$ as:

$$\tilde{\boldsymbol{C}}_{\mathrm{PRE}}(k,j) = \begin{bmatrix} \tilde{\boldsymbol{c}}_{\mathrm{PRE},1}(k,j) \\ \tilde{\boldsymbol{c}}_{\mathrm{PRE},2}(k,j) \\ \vdots \\ \tilde{\boldsymbol{c}}_{\mathrm{PRE},O_{\mathrm{PAR,MAX}}}(k,j) \end{bmatrix} \quad \text{for} \quad j = 1, \ldots, F$$

The filter bank is assumed to be based on quadrature mirror filters (QMF) with a total of $F = 64$ sub-bands as defined in ISO/IEC 23003-1.

12.4.2.8.3 Truncation and coefficient selection

For each j-th sub-band belonging to the g-th sub-band group, $g = 1, \ldots, G$, a truncated version $\tilde{\boldsymbol{C}}_{\mathrm{SP}}(k,j)$ of $\tilde{\boldsymbol{C}}_{\mathrm{PRE}}(k,j)$ of order $N_{\mathrm{PAR}}(g)$ is computed, which is composed of the individual coefficient sequences according to

$$\tilde{\boldsymbol{C}}_{\mathrm{SP}}(k,j) = \begin{bmatrix} \tilde{\boldsymbol{c}}_{\mathrm{SP},1}(k,j) \\ \tilde{\boldsymbol{c}}_{\mathrm{SP},2}(k,j) \\ \vdots \\ \tilde{\boldsymbol{c}}_{\mathrm{SP},O_{\mathrm{PAR}}(g)}(k,j) \end{bmatrix} \text{for} \quad \mathcal{L}_{\mathrm{PAR}}(g) \le j \le \mathcal{U}_{\mathrm{PAR}}(g) \quad g = 1, \ldots, G.$$

The coefficient sequences $\tilde{\boldsymbol{c}}_{\mathrm{SP},n}(k,j)$ with $n = 1, \ldots, O_{\mathrm{PAR}}(g)$ of the truncated HOA representation $\tilde{\boldsymbol{C}}_{\mathrm{SP}}(k,j)$ are either taken from $\tilde{\boldsymbol{C}}_{\mathrm{PRE}}(k,j)$ if their index is contained in the index set of the transmitted coefficient sequences of the ambient HOA component, defined by

$$\mathcal{I}_{\mathrm{AMB,ACT}}(k) \coloneqq \mathcal{I}_{\mathrm{E}}(k) \cup \mathcal{I}_{\mathrm{D}}(k) \cup \mathcal{I}_{\mathrm{U}}(k),$$

or set to zero else, i.e.:

$$\tilde{\boldsymbol{c}}_{\mathrm{SP},n}(k,j) = \begin{cases} \tilde{\boldsymbol{c}}_{\mathrm{PRE},n}(k,j) & \text{if} \quad n \in \mathcal{I}_{\mathrm{AMB,ACT}}(k) \\ \boldsymbol{0} & \text{else} \end{cases}$$

12.4.2.8.4 Spatial transform

Each truncated HOA sub-band representation $\tilde{\boldsymbol{C}}_{\mathrm{SP}}(k,j)$ of the j-th sub-band belonging to the g-th sub-band group, $g = 1, \ldots, G$, is subjected to a spatial transform. This spatial transform is equivalent to performing the rendering to $O_{\mathrm{PAR}}(g)$ virtual loudspeaker signals $\mathbf{x}_{\mathrm{SP},d}(k,j)$, at the directions $\boldsymbol{\Omega}_d^{(N_{\mathrm{PAR}}(g))}$, $d = 1, \ldots, O_{\mathrm{PAR}}(g)$, defined in the tables in Annexes F.2 to F.11. Arranging the individual virtual loudspeaker signals in the matrix $\tilde{\boldsymbol{X}}_{\mathrm{SP}}(k,j)$ according to

$$\widetilde{\boldsymbol{\mathcal{X}}}_{\mathrm{SP}}(k,j) = \begin{bmatrix} \boldsymbol{x}_{\mathrm{SP},1}(k,j) \\ \boldsymbol{x}_{\mathrm{SP},2}(k,j) \\ \vdots \\ \boldsymbol{x}_{\mathrm{SP},O_{\mathrm{PAR}}(g)}(k,j) \end{bmatrix} \quad \text{for} \quad \mathcal{L}_{\mathrm{PAR}}(g) \le j \le \mathcal{U}_{\mathrm{PAR}}(g), \quad g = 1, \ldots, G,$$

the spatial transform is expressed by means of multiplying the truncated HOA representation $\widetilde{\boldsymbol{\mathcal{C}}}_{\mathrm{SP}}(k,j)$ with the inverse of the mode matrix $\boldsymbol{\Psi}^{(N_{\mathrm{PAR}}(g),N_{\mathrm{PAR}}(g))}$ (defined in Annex F.1.5) with respect to these directions by:

$$\widetilde{\boldsymbol{\mathcal{X}}}_{\mathrm{SP}}(k,j) = \left(\boldsymbol{\Psi}^{(N_{\mathrm{PAR}}(g),N_{\mathrm{PAR}}(g))}\right)^{-1} \cdot \widetilde{\boldsymbol{\mathcal{C}}}_{\mathrm{SP}}(k,j) \quad \text{for} \quad \mathcal{L}_{\mathrm{PAR}}(g) \le j \le \mathcal{U}_{\mathrm{PAR}}(g).$$

12.4.2.8.5 Computation of de-correlated sub-band signals

For the computation of the de-correlated signals $\widetilde{\boldsymbol{\mathcal{X}}}_{\mathrm{DEC}}(k,j)$ for the j-th sub-band belonging to the g-th sub-band group, the virtual loudspeaker sub-band signals $\widetilde{\boldsymbol{x}}_{\mathrm{SP},d}(k,j)$, $d = 1, \ldots, O_{\mathrm{PAR}}(g)$ are first assigned to the $O_{\mathrm{PAR}}(g)$ de-correlation filters. To obtain continuous input signals for the de-correlation filters, over-lap add processing is employed. In particular, the input signals $\widetilde{\boldsymbol{\mathcal{X}}}_{\mathrm{INDEC}}(k,j)$ to the de-correlation filters for the j-th sub-band are computed as the sum of a faded out component and a faded in component:

$$\widetilde{\boldsymbol{\mathcal{X}}}_{\mathrm{INDEC}}(k,j) = \widetilde{\boldsymbol{\mathcal{X}}}_{\mathrm{INDEC,OUT}}(k,j) + \widetilde{\boldsymbol{\mathcal{X}}}_{\mathrm{INDEC,IN}}(k,j)$$

To compute the two individual components, firstly, the instantaneous frame $\widetilde{\boldsymbol{\mathcal{X}}}_{\mathrm{INDEC,I}}(k_1;k;j)$ of all permuted virtual loudspeaker sub-band signals of the sparse and truncated HOA sub-band representation $\widetilde{\boldsymbol{\mathcal{C}}}_{\mathrm{SP}}(k,j)$ for the k-th frame is computed by:

$$\widetilde{\boldsymbol{\mathcal{X}}}_{\mathrm{INDEC,I}}(k_1;k;j) = \widetilde{\boldsymbol{P}}_{\mathrm{PAR}}(k_1,g) \cdot \widetilde{\boldsymbol{\mathcal{X}}}_{\mathrm{SP}}(k,j) \quad \text{for} \quad k_1 \in \{k-1,k\} \text{ and } \mathcal{L}_{\mathrm{PAR}}(g) \le j \le \mathcal{U}_{\mathrm{PAR}}(g)$$

where $\widetilde{\boldsymbol{P}}_{\mathrm{PAR}}(k_1,g)$ is the permutation matrix for the g-th sub-band group including the j-th sub-band and for the (k_1)-th frame, where $k_1 \in \{k-1,k\}$. The sample values of the faded out and faded in components of the input signal frames to the de-correlation filters are determined for $1 \le l \le L_{\mathrm{SB}}$, $\mathcal{L}_{\mathrm{PAR}}(g) \le j \le \mathcal{U}_{\mathrm{PAR}}(g)$, $g = 1, \ldots, G$, and $d = 1, \ldots, O_{\mathrm{PAR}}(g)$ by:

$$\tilde{x}_{\mathrm{INDEC,OUT},d}(k,j;l) = \tilde{x}_{\mathrm{INDEC,I},d}(k-1;k;j;l) \cdot w_{\mathrm{SB}}(L_{\mathrm{SB}}+l)$$

$$\tilde{x}_{\mathrm{INDEC,IN},d}(k,j;l) = \tilde{x}_{\mathrm{INDEC,I},d}(k;k;j;l) \cdot w_{\mathrm{SB}}(l)$$

where $w_{\mathrm{SB}}(l)$ denotes the elements of the window function vector $\mathbf{w}_{\mathrm{SB}}$ defined in subclause 12.4.2.7.3. In a next step, each d-th output signal $\widetilde{\boldsymbol{x}}_{\mathrm{DEC},d}(k,j)$, $d = 1, \ldots, O_{\mathrm{PAR}}(g)$, shall be computed by applying to the d-th input sub-band signal $\widetilde{\boldsymbol{x}}_{\mathrm{INDEC},d}(k,j)$ one of the ten different 3rd order IIR de-correlation filters (indexed from 0 to 9) as specified in ISO/IEC 23003-1:2007, Table A.29, where the index of the applied all-pass filter shall be selected via the signal index d according to Table 211.

Table 211 — Assignment of signal index to filter index

Signal index d	Filter set X
1	0
2	1
3	2
4	3
5	4
6	5
7	7
8	8
9	9

12.4.2.8.6 Ambience replication

The replicated ambient HOA component for the j-th sub-band belonging to the g-th sub-band group, $g = 1, \ldots, G$, is represented by means of upmix signals being virtual loudspeaker sub-band signals $\tilde{\boldsymbol{x}}_{\mathrm{PA},d}(k,j)$ at the directions $\boldsymbol{\Omega}_d^{(N_{\mathrm{PAR}}(g))}$, $d = 1, \ldots, O_{\mathrm{PAR}}(g)$, defined in the tables in Annexes F.2 to F.11. The upmix sub-band signals are computed by re-assigning the de-correlated signals back to the virtual loudspeaker directions and successively mixing them. For the purpose of signal continuity overlap-add processing on the upmix signals is carried out. In particular, the upmix signals $\tilde{\boldsymbol{X}}_{\mathrm{PA}}(k,j)$ for the j-th sub-band are computed as the sum of a faded out component and a faded in component:

$$\tilde{\boldsymbol{X}}_{\mathrm{PA}}(k,j) = \tilde{\boldsymbol{X}}_{\mathrm{PA,OUT}}(k,j) + \tilde{\boldsymbol{X}}_{\mathrm{PA,IN}}(k,j)$$

To compute the two individual components, in a first step the instantaneous frame $\tilde{\boldsymbol{X}}_{\mathrm{PA,I}}(k_1;k;j)$ of all upmix sub-band signals for the k-th frame is computed by

$$\tilde{\boldsymbol{X}}_{\mathrm{PA,I}}(k_1;k;j) = \tilde{\boldsymbol{M}}_{\mathrm{PAR}}(k_1,g)\left(\tilde{\boldsymbol{P}}_{\mathrm{PAR}}(k_1,g)\right)^{-1} \cdot \tilde{\boldsymbol{X}}_{\mathrm{DEC}}(k,j)$$

$$\text{for} \quad k_1 \in \{k-1,k\}, \mathcal{L}_{\mathrm{PAR}}(g) \leq j \leq \mathcal{U}_{\mathrm{PAR}}(g) \quad \text{and} \quad g = 1, \ldots, G$$

Here, $\boldsymbol{P}_{\mathrm{PAR}}(k_1,g)$ denotes the inverse of the permutation matrix describing the re-assignment for the g-th sub-band group and the k_1-th frame, where $k_1 \in \{k-1,k\}$. Further, $\boldsymbol{M}_{\mathrm{PAR}}(k_1,g)$ is the corresponding mixing matrix. Assuming the matrices $\boldsymbol{X}_{\mathrm{PA,OUT}}(k,j)$, $\tilde{\boldsymbol{X}}_{\mathrm{PA,IN}}(k,j)$, and $\tilde{\boldsymbol{X}}_{\mathrm{PA,I}}(k_1;k;d)j$ to be composed of their samples by:

$$\tilde{\boldsymbol{X}}_{\mathrm{PA,OUT}}(k,j) = \begin{bmatrix} \tilde{x}_{\mathrm{PA,OUT},1}(k,j;1) & \ldots & \tilde{x}_{\mathrm{PA,OUT},1}(k,j;L_{\mathrm{SB}}) \\ \vdots & \ddots & \vdots \\ \tilde{x}_{\mathrm{PA,OUT},O_{\mathrm{PAR}}(g)}(k,j;1) & \ldots & \tilde{x}_{\mathrm{PA,OUT},O_{\mathrm{PAR}}(g)}(k,j;L_{\mathrm{SB}}) \end{bmatrix} \in \mathbb{C}^{O_{\mathrm{PAR}}(g)\times L_{\mathrm{SB}}}$$

$$\tilde{\boldsymbol{X}}_{\mathrm{PA,IN}}(k,j) = \begin{bmatrix} \tilde{x}_{\mathrm{PA,IN},1}(k,j;1) & \ldots & \tilde{x}_{\mathrm{PA,IN},1}(k,j;L_{\mathrm{SB}}) \\ \vdots & \ddots & \vdots \\ \tilde{x}_{\mathrm{PA,IN},O_{\mathrm{PAR}}(g)}(k,j;1) & \ldots & \tilde{x}_{\mathrm{PA,IN},O_{\mathrm{PAR}}(g)}(k,j;L_{\mathrm{SB}}) \end{bmatrix} \in \mathbb{C}^{O_{\mathrm{PAR}}(g)\times L_{\mathrm{SB}}}$$

$$\tilde{\boldsymbol{X}}_{\mathrm{PA,I}}(k_1;k;j) = \begin{bmatrix} \tilde{x}_{\mathrm{PA,I},1}(k_1;k;j;1) & \ldots & \tilde{x}_{\mathrm{PA,I},1}(k_1;k;j;L_{\mathrm{SB}}) \\ \vdots & \ddots & \vdots \\ \tilde{x}_{\mathrm{PA,I},O_{\mathrm{PAR}}(g)}(k_1;k;j;1) & \ldots & \tilde{x}_{\mathrm{PA,I},O_{\mathrm{PAR}}(g)}(k_1;k;j;L_{\mathrm{SB}}) \end{bmatrix} \in \mathbb{C}^{O_{\mathrm{PAR}}(g)\times L_{\mathrm{SB}}}$$

the sample values of the faded out and faded in components of the upmix sub-band signals are determined for $1 \le l \le L_{\mathrm{SB}}, \mathcal{L}_{\mathrm{PAR}}(g) \le j \le \mathcal{U}_{\mathrm{PAR}}(g), g = 1, \ldots, G$, and $d = 1, \ldots, O_{\mathrm{PAR}}(g)$ by

$$\tilde{x}_{\mathrm{PA,OUT},d}(k, j; l) = \tilde{x}_{\mathrm{PA,I},d}(k-1; k; j; l) \cdot w_{\mathrm{SB}}(L_{\mathrm{SB}} + l)$$

$$\tilde{x}_{\mathrm{PA,IN},d}(k, j; l) = \tilde{x}_{\mathrm{PA,I},d}(k; k; j; l) \cdot w_{\mathrm{SB}}(l)$$

where $w_{\mathrm{SB}}(l)$ denote the elements of the window function vector $\mathbf{w}_{\mathrm{SB}}$ defined in subclause 12.4.2.7.3.

12.4.2.8.7 Inverse spatial transform

The virtual loudspeaker signals $\widetilde{\boldsymbol{\mathcal{X}}}_{\mathrm{PA}}(k, j)$ representing the replicated ambient HOA component for the j-th sub-band belonging to the g-th sub-band group, $g = 1, \ldots, G$, are subjected to an inverse spatial transform to provide their HOA representation $\widetilde{\boldsymbol{\mathcal{C}}}_{\mathrm{PA,I}}(k, j)$, which is expressed by means of multiplication with the mode matrix $\boldsymbol{\Psi}^{(N_{\mathrm{PAR}}(g), N_{\mathrm{PAR}}(g))}$ (defined in Annex F.1.5) as

$$\widetilde{\boldsymbol{\mathcal{C}}}_{\mathrm{PA,I}}(k, j) = \boldsymbol{\Psi}^{(N_{\mathrm{PAR}}(g), N_{\mathrm{PAR}}(g))} \cdot \widetilde{\boldsymbol{\mathcal{X}}}_{\mathrm{PA}}(k, j) \quad \text{for} \quad \mathcal{L}_{\mathrm{PAR}}(g) \le j \le \mathcal{U}_{\mathrm{PAR}}(g) \quad \text{and} \quad g = 1, \ldots, G$$

The final output HOA representation $\widetilde{\boldsymbol{\mathcal{C}}}_{\mathrm{PA}}(k, j)$ is obtained from $\widetilde{\boldsymbol{\mathcal{C}}}_{\mathrm{PA,I}}(k, j)$ by padding it with zeros to order $N_{\mathrm{PAR,MAX}}$, i.e.

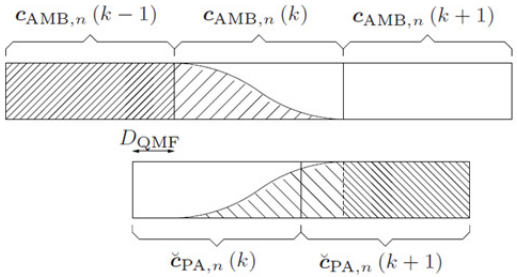
$$\widetilde{\boldsymbol{\mathcal{C}}}_{\mathrm{PA,F}}(k, j) = \begin{bmatrix} \widetilde{\boldsymbol{\mathcal{C}}}_{\mathrm{PA,I}}(k, j) \\ \mathbf{0} \end{bmatrix} \in \mathbb{C}^{N_{\mathrm{PAR,MAX}} \times L_{\mathrm{SB}}} \quad \text{for} \quad j = 1, \ldots, F$$

12.4.2.8.8 Synthesis filter banks and coefficient selection and fading

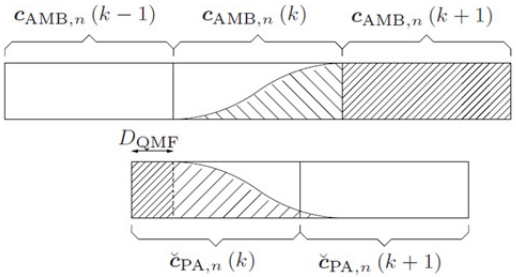
Firstly, the frame $\breve{\boldsymbol{\mathcal{C}}}_{\mathrm{PA,F}}(k) \in \mathbb{R}^{O \times L}$ of the preliminary parametrically replicated ambient HOA component is computed as follows:

Its first $O_{\mathrm{PAR,MAX}}$ time domain coefficient sequences $\breve{\boldsymbol{c}}_{\mathrm{PA},n}(k), n = 1, \ldots, O_{\mathrm{PAR,MAX}}$, are synthesized from the corresponding sub-band coefficient sequences $\tilde{\boldsymbol{c}}_{\mathrm{PA},n}(k, j), j = 1, \ldots, F$, by the Synthesis filter banks. The remaining time-domain coefficient sequences $\breve{\boldsymbol{c}}_{\mathrm{PA},n}(k)$ with indices $n = O_{\mathrm{PAR,MAX}} + 1, \ldots, O$ are set to zero. Note that the synthesized time domain coefficient sequences have a delay of $D_{\mathrm{QMF}} = 577$ samples due to the successive application of the QMF based analysis and synthesis filter banks, which is expressed by the breve symbol ($\breve{}$) above the variables.

Next, the frame $\breve{\boldsymbol{\mathcal{C}}}_{\mathrm{PA,F}}(k)$ of the preliminary parametrically replicated ambient HOA component is modified so that only contributions for those coefficient sequences, which have not been explicitly transmitted for the ambient HOA component. Further, for those coefficient sequences of the ambient HOA component that are transmitted in addition and faded in (or faded out), the respective coefficient sequences of the HOA representation $\breve{\boldsymbol{\mathcal{C}}}_{\mathrm{PA,F}}(k)$ of the parametrically replicated ambient HOA component have to be modified by fading them out (or fading them in), respectively. Due to the delay between $\boldsymbol{\mathcal{C}}_{\mathrm{AMB}}(k)$ and $\breve{\boldsymbol{\mathcal{C}}}_{\mathrm{PA,F}}(k)$ of D_{QMF} samples the fading of the coefficient sequences of $\breve{\boldsymbol{\mathcal{C}}}_{\mathrm{PA,F}}(k)$ is performed across frame boundaries as illustrated in Figure 70. The resulting HOA representation of the parametrically replicated ambient HOA component is denoted by $\breve{\boldsymbol{\mathcal{C}}}_{\mathrm{PA}}(k)$ with its coefficient sequences $\breve{\boldsymbol{c}}_{\mathrm{PA},n}(k), n = 1, \ldots, O$.



(a) Example of a faded coefficient sequence $\check{\boldsymbol{c}}_{\mathrm{PA},n}$ of the HOA representation of the parametrically replicated ambient HOA component in the case the respective coefficient sequence $\boldsymbol{c}_{\mathrm{AMB},n}$ of the ambient HOA component is faded out, i.e. if $n \in \mathcal{I}_{\mathrm{U}}(k-1) \wedge n \in \mathcal{I}_{\mathrm{D}}(k) \wedge n \notin (\mathcal{I}_{\mathrm{U}}(k+1) \cup \mathcal{I}_{\mathrm{E}}(k+1) \cup \mathcal{I}_{\mathrm{D}}(k+1))$



(b) Example of a faded coefficient sequence $\check{\boldsymbol{c}}_{\mathrm{PA},n}$ of the HOA representation of the parametrically replicated ambient HOA component in the case the respective coefficient sequence $\boldsymbol{c}_{\mathrm{AMB},n}$ of the ambient HOA component is faded in, i.e. if $n \notin (\mathcal{I}_{\mathrm{U}}(k-1) \cup \mathcal{I}_{\mathrm{E}}(k-1) \cup \mathcal{I}_{\mathrm{D}}(k-1)) \wedge n \in \mathcal{I}_{\mathrm{E}}(k) \wedge n \in \mathcal{I}_{\mathrm{U}}(k+1)$

Figure 70 — Illustration of faded coefficient sequences of $\check{\boldsymbol{c}}_{\mathrm{PA}}(k)$

In the case that a coefficient sequence $\boldsymbol{c}_{\mathrm{AMB},n}$ of the ambient HOA component is faded out in the k-th frame (i.e. $n \in \mathcal{I}_{\mathrm{D}}(k)$) as illustrated in Figure 70a, the fade in of the coefficient sequence $\check{\boldsymbol{c}}_{\mathrm{PA},n}$ in the k-th frame begins D_{QMF} samples later, where in particular the fading in is finished only at the D_{QMF}-th sample of the $(k+1)$-th frame.

Similarly, in the case that a coefficient sequence $\boldsymbol{c}_{\mathrm{AMB},n}$ of the ambient HOA component is faded in in the k-th frame (i.e. $n \in \mathcal{I}_{\mathrm{E}}(k)$) as illustrated in Figure 70b, the fade out of the coefficient sequence $\check{\boldsymbol{c}}_{\mathrm{PA},n}$ in the k-th frame begins D_{QMF} samples later, where in particular the fading out is finished only at the D_{QMF}-th sample of the $(k+1)$-th frame.

Finally, it has to be considered that a fade in or fade out of the coefficient sequences $\check{\boldsymbol{c}}_{\mathrm{PA,F},n}(k)$ of the HOA representation of the composition of all predicted sub-band directional signals is only required if it is not already present, resulting from overlap-add processing.

In the case that $\tilde{b}_{\mathrm{PAR}}(k-1) = 0$ and $\tilde{b}_{\mathrm{PAR}}(k) = 1$, there is already a fade in within each of the k-th frames $\check{\boldsymbol{c}}_{\mathrm{PA,F},n}(k)$, $n = 1, \ldots, O$, and hence it is not necessary to apply an additional fade in. Similarly, in the case that $\tilde{b}_{\mathrm{PAR}}(k-1) = 1$ and $\tilde{b}_{\mathrm{PAR}}(k) = 0$, there is already a fade out within each of the k-th frames $\check{\boldsymbol{c}}_{\mathrm{PA,F},n}(k)$, $n = 1, \ldots, O$, and hence it is not necessary to apply an additional fade out. Altogether, the computation of the sample values $\check{c}_{\mathrm{PA},n}(k,l)$, $n = 1, \ldots, O$, $l = 1, \ldots, L$, of the coefficient sequences of the HOA representation of the parametrically replicated ambient HOA component is hence formally expressed by:

$$\check{c}_{\mathrm{PA},n}(k,l) = \check{c}_{\mathrm{PA,F},n}(k,l) \cdot \begin{cases} w_{\mathrm{DIR}}(l + L - D_{\mathrm{QMF}}) & \text{if } 1 \le l \le D_{\mathrm{QMF}} \wedge n \in \mathcal{I}_{\mathrm{D}}(k-1) \wedge \tilde{b}_{\mathrm{PAR}}(k-2) = 1 \\ w_{\mathrm{DIR}}(l + 2L - D_{\mathrm{QMF}}) & \text{if } 1 \le l \le D_{\mathrm{QMF}} \wedge n \in \mathcal{I}_{\mathrm{E}}(k-1) \wedge \tilde{b}_{\mathrm{PAR}}(k-1) = 1 \\ 1 & \text{if } 1 \le l \le D_{\mathrm{QMF}} \wedge \\ & \{n \notin \mathcal{I}_{\mathrm{U}}(k-1) \cup \mathcal{I}_{\mathrm{E}}(k-1) \cup \mathcal{I}_{\mathrm{D}}(k-1) \\ & \quad \vee \left(n \in \mathcal{I}_{\mathrm{D}}(k-1) \wedge \tilde{b}_{\mathrm{PAR}}(k-2) = 0\right) \\ & \quad \vee \left(n \in \mathcal{I}_{\mathrm{E}}(k-1) \wedge \tilde{b}_{\mathrm{PAR}}(k-1) = 0\right)\} \\ w_{\mathrm{DIR}}(l - D_{\mathrm{QMF}}) & \text{if } D_{\mathrm{QMF}} < l \le L \wedge n \in \mathcal{I}_{\mathrm{D}}(k) \wedge \tilde{b}_{\mathrm{PAR}}(k-1) = 1 \\ w_{\mathrm{DIR}}(l + L - D_{\mathrm{QMF}}) & \text{if } D_{\mathrm{QMF}} < l \le L \wedge n \in \mathcal{I}_{\mathrm{E}}(k) \wedge \tilde{b}_{\mathrm{PAR}}(k) = 1 \\ 1 & \text{if } D_{\mathrm{QMF}} < l \le L \wedge \\ & \{n \notin \mathcal{I}_{\mathrm{U}}(k) \cup \mathcal{I}_{\mathrm{E}}(k) \cup \mathcal{I}_{\mathrm{D}}(k) \\ & \quad \vee \left(n \in \mathcal{I}_{\mathrm{D}}(k) \wedge \tilde{b}_{\mathrm{PAR}}(k-1) = 0\right) \\ & \quad \vee \left(n \in \mathcal{I}_{\mathrm{E}}(k) \wedge \tilde{b}_{\mathrm{PAR}}(k) = 0\right)\} \\ 0 & \text{else} \end{cases}$$

12.4.2.9 HOA composition

The frame $\breve{\boldsymbol{C}}(k)$ of the finally reconstructed HOA representation is computed by a superposition of the frame $\boldsymbol{C}_{\mathrm{PRE}}(k)$ of the preliminary decoded HOA representation, the frame $\breve{\boldsymbol{C}}_{\mathrm{D}}(k)$ of the HOA representation of the composition of all predicted sub-band directional signals and the frame $\breve{\boldsymbol{C}}_{\mathrm{PA}}(k)$ of the replicated ambient HOA component. For the superposition the delay between the individual HOA representations to be superposed shall be taken into consideration. Hence, the computation of the sample values $\check{c}_n(k,l), n = 1, \ldots, O, l = 1, \ldots, L$, of the individual coefficient sequences of $\breve{\boldsymbol{C}}(k)$ is given by:

$$\check{c}_n(k,l) = \check{c}_{\mathrm{D},n}(k,l) + \check{c}_{\mathrm{PA},n}(k,l) + \begin{cases} c_{\mathrm{PRE},n}(k-1, L+l-D_{\mathrm{QMF}}) & \text{if } 1 \le l \le D_{\mathrm{QMF}} \\ c_{\mathrm{PRE},n}(k, l-D_{\mathrm{QMF}}) & \text{if } D_{\mathrm{QMF}} < l \le L \end{cases}$$

12.4.3 HOA renderer

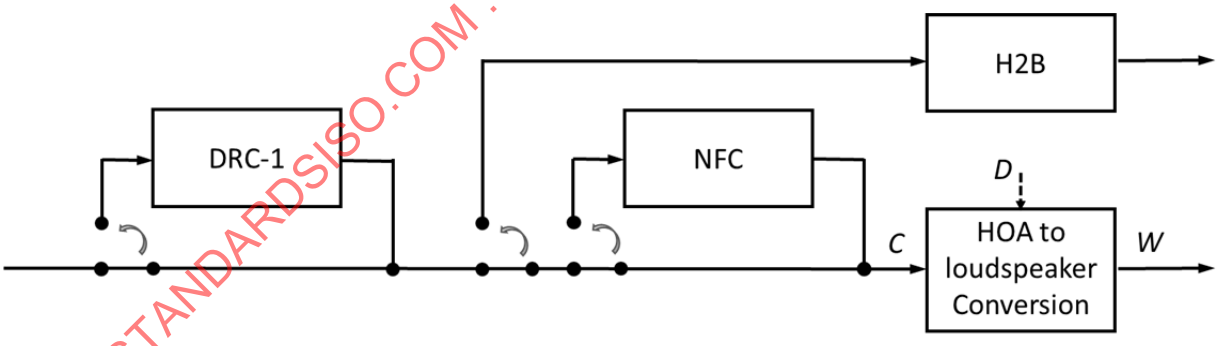
12.4.3.1 General

This subclause describes the conversion of the decoded higher order ambisonics (HOA) representation to loudspeaker signals. In subclause 12.4.2.4.2 the renderer architecture is presented; the design process of the rendering matrix for flexible rendering is given in subclause 12.4.3.3. Clause 16 gives information about a gain and delay compensation for non-spherical placed loudspeaker configurations and subclause 12.4.3.4 explains NFC processing.

12.4.3.2 Architecture overview

The conversion process is shown in Figure 71.

The decoded HOA representation $\boldsymbol{C}$, here described as a matrix of $(N+1)^2$ rows and T columns. N denotes the HOA order (HoaOrder) and T is the block size in number of samples. $\boldsymbol{C}$ is converted to the representation of loudspeaker signals $\boldsymbol{W}$ of size $L \times T$, where L is the number of loudspeaker channels, by multiplication with the rendering matrix $\boldsymbol{D}$: $\boldsymbol{W} = \boldsymbol{D}\,\boldsymbol{C}$. A comprehensive description of the expected HOA format ($i.e.\ \boldsymbol{C}$) can be found in Annexes F.1 and C.5.1.



NOTE Preprocessing block NFC processing (should be switched active if UseNfc is true and the loudspeaker listener distance $\mathbf{r}_{max}$ is smaller than NfcReference Distance), pre-processing block DRC-1 for HOA and as an alternative to rendering to loudspeakers, a computational efficient binaural rendering directly using the HOA coefficients (H2B, see subclause 13.3.1). Computational more efficient but mathematically equivalent ways to implement the processing chain may be found in Annex G.

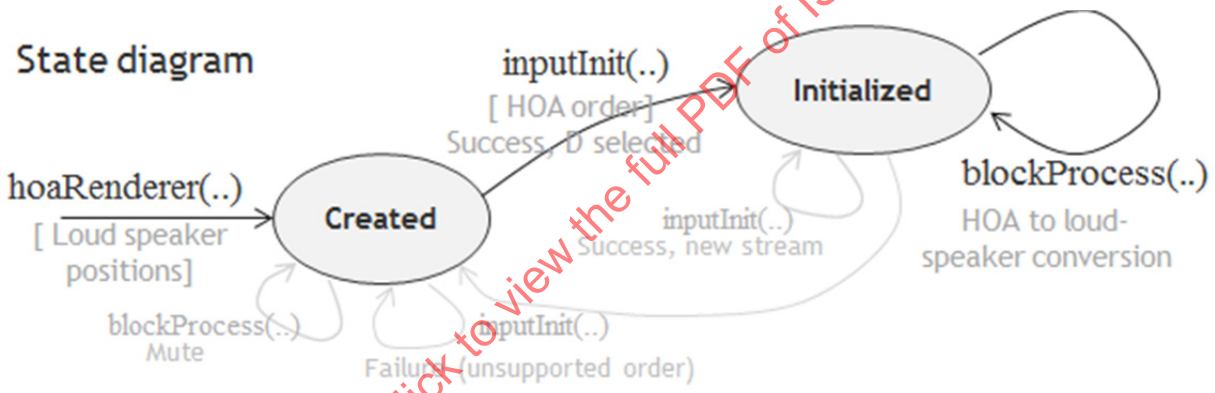
Figure 71 — HOA rendering process

The rendering process assumes that the loudspeakers are positioned at equal distances from the sweet spot (spherical setup) and that the gain and delay compensation is performed as post processing of the

loudspeaker signals before play out to create a virtual spherical setup. Clause 16 provides an informative description. A Peak Limiter should be implemented as part of the post-processing.

The rendering state diagram is shown in Figure 72. On creation the renderer receives the positions of the loudspeaker setup $\mathbf{\Omega}_{\text{SPEAKER}}$. $\mathbf{\Omega}_{\text{SPEAKER}} = [\mathbf{r}_1,\ \mathbf{r}_2, \ldots, \mathbf{r}_L]$ is a matrix of size $3xL$, where L gives the number of loudspeakers and $\mathbf{r}_l = [r_l, \theta_l, \phi_l]^T = \left[r_l, \widehat{\mathbf{\Omega}}_l^T\right]^T$ is the position vector of loudspeaker l. r_l is the distance from the sweet spot (listening position) to loudspeaker l and $\widehat{\mathbf{\Omega}}_l^T = [\theta_l, \phi_l]$ are the related spherical angles. For the definition of the spherical coordinate system of the HOA rendering process see Annex F.1.1, Figure F.1. Note that the definition of the spherical coordinate system used here differs from the one used within other parts of the standard. The inclination angle θ_l here is related to the elevation angle in rad by $\theta_l = \frac{\pi}{2} - \theta_{l,elevation}$.

For all HOA supported orders N the renderer may create rendering matrices and store these in a data base or calculate these matrices on the fly when required (see subclause 12.4.3.3 for details on matrix construction). In case an HOA rendering matrix was signalled via HoaRenderingMatrixSet() (subclause 5.2.2.5) which matches the actual reproduction setup according to the matching rules defined in subclause 10.3.1, the signalled HOA rendering matrix shall be applied for the rendering process.



NOTE There are two states of initialization. First the rendering matrices are created depending on the loudspeaker positions. A matrix suited for content to process is then selected using content meta data information.

Figure 72 — Renderer state diagram

When new content is streamed to the renderer, it receives content information extracted from HOAConfig (). The renderer will try to select a rendering matrix ***D*** using the HoaOrder and create/select NFC filters for pre-processing if necessary, i.e. if UsesNfc is active in the bitstream and **NfcReferenceDistance** > r_{max}, with r_{max} the maximum loudspeaker distance of the listening setup. If successful, the renderer will reach state *Initialized*. Rendering block processing of HOA data then creates L loudspeaker output signals (blockProcess()).

12.4.3.3 Matrix design for flexible rendering

12.4.3.3.1 General

This subclause describes the design of energy preserving rendering matrices, where the number of HOA coefficients $(N+1)^2$ can be larger than the number of loudspeakers L. Energy preservation describes the characteristics that the HOA signal's loudness is preserved independent of the loudspeaker setup and that constant amplitude spatial sweeps can be perceived equal loud after rendering. The subclause is divided into two parts: First the design for 3D loudspeaker setups are described, then the design method for 2D loudspeaker setups is presented. The 2D design method makes use of the 3D method

because it uses virtual loudspeakers placed at the pole positions of a (virtual) spherical loudspeaker setup.

12.4.3.3.2 Matrix design for 3D loudspeaker setups

12.4.3.3.2.1 Building block overview

The building blocks are shown in Figure 73. For each supported HOA order N a matrix $\boldsymbol{D}$ is created. Main input to the design are the loudspeaker positions which are here indicated by $\mathfrak{D}_L$ as directions. L loudspeaker directions are given by $\mathfrak{D}_L = [\,\widehat{\boldsymbol{\Omega}}_1, \ldots, \widehat{\boldsymbol{\Omega}}_\mathrm{L}]$ with spherical angle $\widehat{\boldsymbol{\Omega}}_\mathrm{l} = \left[\hat{\theta}_l, \hat{\phi}_l\right]^T$. A spherical setup is assumed and the loudspeaker distances are neglected. $\hat{\theta}_l$ indicates the inclination and $\hat{\phi}_l$ the loudspeaker azimuth, both in radians. The related coordinate system in Cartesian coordinates is centered at the sweet spot, the X-axis is horizontal with the positive direction pointing towards the ideal centre loudspeaker position; the Y-axis is horizontal with the positive direction pointing to the left of centre and the Z-axis pointing vertically upwards. An ideally placed centre loudspeaker thus would have a $[\frac{\pi}{2}, 0]$ position.

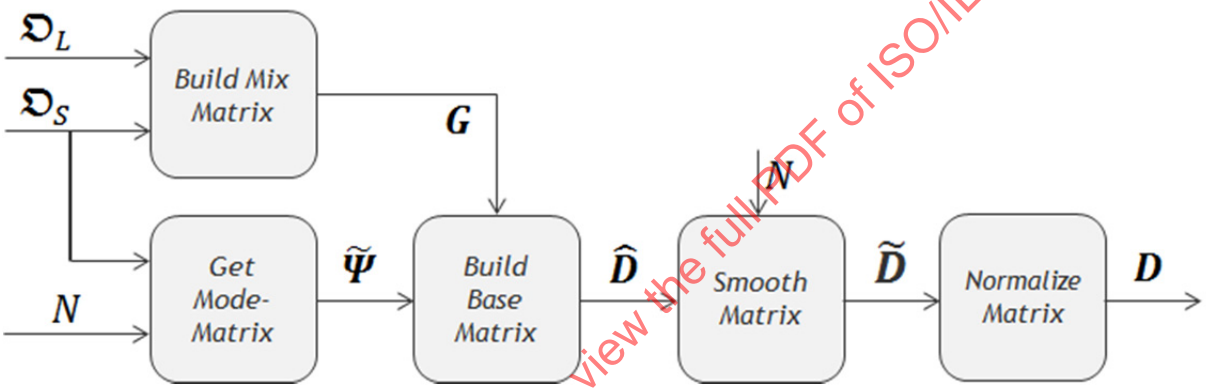


Figure 73 — Building blocks of the matrix design procedure

The matrix design process also requires the ideal spherical design positions $\mathfrak{D}_S = [\,\boldsymbol{\Omega}_1, \ldots, \boldsymbol{\Omega}_\mathrm{S}]$ with $\boldsymbol{\Omega}_\mathrm{s} = [\theta_s, \phi_s]^T$, characterized such that they adequately sample the surface of the unit sphere very regularly. A spherical grid of a $S = 324$ positions is defined, which enable the construction of matrices up to HOA order $N = 9$. The position tables are defined in F.11.

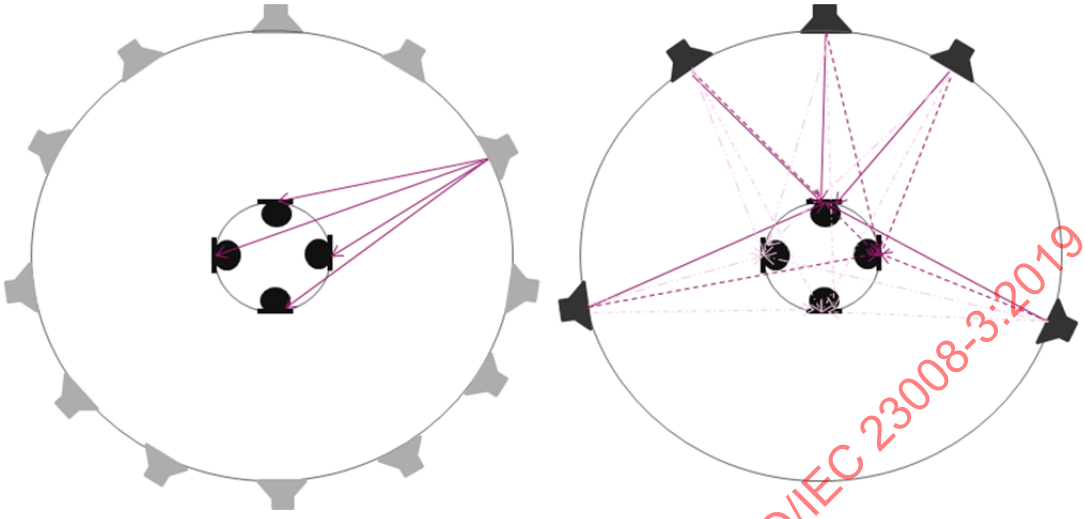
12.4.3.3.2.2 Building block — Get mode matrix

A mode matrix $\widetilde{\boldsymbol{\Psi}}$ dependent upon $\mathfrak{D}_S$ and N is calculated or read from memory. $\widetilde{\boldsymbol{\Psi}} = [\boldsymbol{y_1}, \ldots \boldsymbol{y_S}]$ with $\boldsymbol{y}_s = [S_0^0(\boldsymbol{\Omega}_s),\ S_1^{-1}(\boldsymbol{\Omega}_s), \ldots, S_N^N(\boldsymbol{\Omega}_S)]^H$, where vector $\boldsymbol{y}_s$ holds the real valued spherical harmonic coefficients $S_n^m(\boldsymbol{\Omega}_s)$. For the calculation of $S_n^m(\boldsymbol{\Omega}_s)$ see the definition in Annex F.1.3.

12.4.3.3.2.3 Building block — Build mix matrix

A mix matrix $\boldsymbol{G} = [\boldsymbol{g_1}, \ldots, \boldsymbol{g_S}]$ of size LxS is created which holds gain vectors $\boldsymbol{g_s}$ to achieve a panned mix in direction $\boldsymbol{\Omega}_\mathrm{s}$ with loudspeakers positioned at $\mathfrak{D}_L$. The method of robust panning is presented here which has been shown to be ideal when constructed according to the following processing steps. The principle is depicted in Figure 74. For each and every plane wave from direction $\boldsymbol{\Omega}_\mathrm{s}$, L panning gains in vector $\boldsymbol{g_s}$ ($\boldsymbol{g_s} = \left[g_{s,1}, .., g_{s,l}\right]^T$), intended to attenuate plane waves emitted from $\mathfrak{D}_L$ (loudspeakers) are created in a way that the error recorded at virtual microphones becomes minimal. Let $\boldsymbol{m_s} \in C^{Mx\,1}$ denote M microphone signals receiving the sound radiated from sources placed at $\boldsymbol{\Omega}_\mathrm{s}$ and $\boldsymbol{m_L} \in C^{Mx1}$ the sound from sources placed at $\mathfrak{D}_L$. The virtual microphone signals are given by the plane wave

transfer functions and an excitation signal u by $\boldsymbol{m_s} = \boldsymbol{h}_{Ms} u$ and $\boldsymbol{m_L} = \boldsymbol{H_{ML}}\, \boldsymbol{g_s}\, u$. The error $|\boldsymbol{m_s} - \boldsymbol{m_L}|_2^2$ becomes independent of u.



NOTE Virtual microphones capture sound from direction $\boldsymbol{\Omega_s}$ as a plane wave. Panning gains are calculated such that the sound recorded from loudspeaker directions $\mathfrak{O}_L = [\,\widehat{\boldsymbol{\Omega}}_1, \ldots, \widehat{\boldsymbol{\Omega}}_L]$ becomes as close as possible compared to first measurement.

Figure 74 — Principle of robust panning

Minimizing the error term $\varepsilon_1 = |\boldsymbol{h}_{Ms} - \boldsymbol{H_{ML}}\, \boldsymbol{g_s}|_2^2$ leads to the LMS-error solution for $\boldsymbol{g_s}$. To make the gain vector solution more robust a regularization parameter is introduced that increases the costs if the gains become large for sources positioned far away from direction $\boldsymbol{\Omega}_s$:

$$\textit{Minimize}:\ \varepsilon_2 = |\boldsymbol{h}_{Ms} - \boldsymbol{H_{ML}}\, \boldsymbol{g_s}|_2^2 + \beta \left| \boldsymbol{\mathscr{b}}^{\boldsymbol{T}} \boldsymbol{g_s} \right|_2^2$$

The LMS solution of this equation is given by:

$$\boldsymbol{g_s} = \left(\boldsymbol{H_{ML}^H}\, \boldsymbol{H_{ML}} + \beta\, \boldsymbol{\mathcal{B}^H \mathcal{B}}\right)^{-1} \boldsymbol{H_{ML}^H}\ \ \boldsymbol{h}_{Ms},$$

With $\boldsymbol{\mathcal{B}}$ an LxL diagonal matrix with diagonal elements of vector $\boldsymbol{\mathscr{b}}$: $\boldsymbol{\mathcal{B}} = diag(\boldsymbol{\mathscr{b}})$; $(\,)^T$ the transpose, and $(\,)^H$ the complex conjugate transpose. $\boldsymbol{\mathscr{b}} = [\mathscr{b}_1, \ldots, \mathscr{b}_L]^T$, where $\mathscr{b}_l$ needs to become larger the bigger the spherical angle of direction $\boldsymbol{\Omega}_s$ to loudspeaker direction $\widehat{\boldsymbol{\Omega}}_l$ gets: $\mathscr{b}_l \sim (\measuredangle \boldsymbol{\Omega}_s, \widehat{\boldsymbol{\Omega}}_l)$.

The transfer functions $\boldsymbol{h}_{Ms}$, $\boldsymbol{H_{ML}}$ are functions of frequency. To ensure that the gains $\boldsymbol{g_s}$ are independent of frequency, the virtual microphone radius is selected as a function of frequency.

Detailed description:

1) A test frequency $f = 1\,000\ Hz$ is defined. Calculation of
$k = \frac{2\,\pi\, f}{c_s}$,
with c_s is the speed of sound (340m/s). β is selected dependent of the HOA order of the content to be rendered, see Table 187.

2) The spherical angles of virtual microphone positions $\mathfrak{O}_M$, with $\mathfrak{O}_M = [\,\ddot{\boldsymbol{\Omega}}_1, \ldots, \ddot{\boldsymbol{\Omega}}_M]$ with $\ddot{\boldsymbol{\Omega}}_m = \left[\ddot{\theta}_m, \ddot{\phi}_m\right]^T$, are defined with $M = 256$ positions in Annex F.11. The microphone position radius is calculated by:

$$r_M = \frac{0.4667\left(\sqrt{L}-1\right)}{k}$$

with L the number of loudspeakers.

3) Calculation of $\boldsymbol{H_{ML}} \in C^{Mx\,L}$ with matrix elements

$H_{m,l} = e^{i\,k\,r_M\,\cos(\gamma_{l,m})}$

where $\gamma_{l,m}$ is the spherical angle between $\ddot{\boldsymbol{\Omega}}_{\mathrm{m}}$ and $\hat{\boldsymbol{\Omega}}_{\mathrm{l}}$:

$\cos(\gamma_{l,m}) = \cos(\hat{\theta}_l)\cos(\ddot{\theta}_m) + \sin(\hat{\theta}_l)\,sin\left(\ddot{\theta}_m\right)\,\cos(\hat{\phi}_l - \ddot{\phi}_m)$. This is the plane wave transfer function and equal to $e^{i\,(k\,\hat{\boldsymbol{X}}_l^T)\ddot{\boldsymbol{X}}_m}$ where $\hat{\boldsymbol{X}}_l$ are the loudspeaker positions in Cartesian coordinates (with radius 1) and $\ddot{\boldsymbol{X}}_{\boldsymbol{m}}$ the microphone positions in Cartesian coordinates (with radius r_M).

4) Loop: $for(s = 1, s \le S, s++)$

a) Calculation of $\boldsymbol{h}_{Ms} \in C^{Mx1}$ with vector elements

$h_{m,s} = e^{i\,k\,r_M\,\cos(\hat{\gamma}_{s,m})}$

where $\hat{\gamma}_{s,m}$ is the spherical angle between $\ddot{\boldsymbol{\Omega}}_{\mathrm{m}}$ and $\boldsymbol{\Omega}_{\mathrm{s}}$:

$\cos(\hat{\gamma}_{s,m}) = \cos(\theta_s)\cos(\ddot{\theta}_m) + \sin(\theta_s)\,sin\left(\ddot{\theta}_m\right)\,\cos(\phi_s - \ddot{\phi}_m)$.

b) Calculation of $\boldsymbol{B} = diag(\boldsymbol{\mathscr{b}})$ with:

$\mathscr{b}_l = 1 - \left(0.5 + 0.5\cos\left(\bar{\gamma}_{s,l}\right)\right)^2$

and $\cos\left(\bar{\gamma}_{s,l}\right) = \cos(\theta_s)\cos(\hat{\theta}_l) + \sin(\theta_s)\,sin\left(\hat{\theta}_l\right)\,\cos(\phi_s - \hat{\phi}_l)$.

c) Calculation of $\boldsymbol{g_s} = \left(\boldsymbol{H_{ML}^H}\,\boldsymbol{H_{ML}} + \beta\,\boldsymbol{\mathcal{B}^H\mathcal{B}}\right)^{-1}\,\boldsymbol{H_{ML}^H}\;\;\boldsymbol{h}_{Ms}$.

d) The L gain values are complex with a very small imaginary part. We use only real valued gains: $\hat{\boldsymbol{g}}_{\boldsymbol{s}} = Re(\boldsymbol{g_s})$.

e) Fill in the gain vector into the Mix Matrix :

$\boldsymbol{G} = [\hat{\boldsymbol{g}}_{\boldsymbol{1}}, .., \hat{\boldsymbol{g}}_{\boldsymbol{s}}, ..]$

Table 212 — Values of regularization parameter β depending on HOA content order

HOA order N	1	2	3	4	>4
β	1 000	1 000	100	100	50

12.4.3.3.2.4 Building block — Build base matrix

The compact singular value decomposition of the matrix product of the mode matrix and the transposed mixing matrix is calculated by:

$$\boldsymbol{U}\,\boldsymbol{S}\,\boldsymbol{V}^H = \tilde{\boldsymbol{\Psi}}\boldsymbol{G}^T$$

Matrix $\boldsymbol{S} = diag(S_1, \ldots, S_K)$ is diagonal with the singular values as diagonal elements. Let S_{max} denote the maximal singular value.

A new diagonal matrix with $\hat{\boldsymbol{S}} = diag(\hat{S}_1, \ldots, \hat{S}_K)$ is created with $\hat{S}_k = \begin{cases} 1 & if\ S_k/S_{max} \geq a \\ 0 & else \end{cases}$

A threshold value a of –60dB was selected: $a = 10^{-3}$.

The base matrix is calculated as follows: $\widehat{\boldsymbol{D}} = \boldsymbol{V}\ \widehat{\boldsymbol{S}}\ \boldsymbol{U}^H$

12.4.3.3.2.5 Building block — Smooth matrix

The task of this building block is to smooth the directive properties of the renderer, i.e. to attenuate the back and side lobes of the loudspeaker panning pattern with the cost of widening the main lobe. The smoothed rendering matrix $\widetilde{\boldsymbol{D}}$ is created by:

$$\widetilde{\boldsymbol{D}} = \widehat{\boldsymbol{D}}\ \boldsymbol{diag}\ (\mathfrak{w})$$

where $\mathfrak{w}$ is a gain vector of size $(N+1)^2$. The process is analogous to windowing in time-frequency processing and is equal to a left convolution on $\mathcal{S}^2$ (on the unit sphere) in the spatial domain. The smoothing gains of vector $\mathfrak{w}$ are constructed from a helper vector $\mathfrak{v}$ with $N+1$ elements.

To construct the helper vector $\mathfrak{v}$ two different approaches are to be used.

— If $L \geq (N+1)^2$, i.e. if the number of loudspeakers is larger or equal to the number of HOA coefficients, so called $max\_\boldsymbol{r}_E$ coefficients are used. Algorithm:

  — The rightmost zeros of the Lengendre Polynomials of increasing order N, starting with order 1 to order 13 are given in the vector $\boldsymbol{Z}$: $\boldsymbol{Z}$ = `[0.0, 0.5574, 0.7746, 0.8611, 0.9062, 0.9325, 0.9491, 0.9603, 0.9682, 0.9739, 0.9782, 0.9816, 0.9842];`

  — select value $rE = z_{N+1}$ from $\boldsymbol{Z}$. (In C++ notation rE = z[N])

  — set $\mathfrak{v}_1 = 1$ and obtain the remaining values by calculating the Legendre polynomial $P_n(\,)$ of order n for the value rE:
  for$(n = 1; n < N; n++)$
  $\mathfrak{v}_{n+1} = P_n(rE)$

— if $L < (N+1)^2$ a Kaiser-Bessel right half-window design is used to calculate the coefficients $\mathfrak{v}_n$:

$$\mathfrak{v}_n = \frac{I_0\left(width\sqrt{1-\left(\frac{2(n+N-1)}{len-1}-1\right)^2}\right)}{I_0(width)} \qquad \text{for } n = 1 \ldots N+1$$

where $I_0(\,)$ denotes the zero-order Modified Bessel function of the first kind with parameters $len = 2N+1$ and $width = N+1$.

Vector $\mathfrak{w}$ is now constructed from $\mathfrak{v}_n$:$\mathfrak{w} = [\,\mathfrak{v}_1, \mathfrak{v}_2, \mathfrak{v}_2, \mathfrak{v}_2, \mathfrak{v}_3, \mathfrak{v}_3, \mathfrak{v}_3, \mathfrak{v}_3, \mathfrak{v}_3, \mathfrak{v}_4, .., \mathfrak{v}_N]^T$

which is to be computed by:

for $(n = 0;\ n \leq N, n++)$

for$(m = -n; m \leq n; m++)$

$$\mathfrak{w}_{n^2+n+m+1} = \mathfrak{v}_{n+1}$$

12.4.3.3.2.6 Building block0020— Normalize matrix

The rendering matrix is derived by normalization by its Frobenius norm:

$$\boldsymbol{D} = \frac{\widetilde{\boldsymbol{D}}}{||\,\widetilde{\boldsymbol{D}}||_f}$$

where $||\,\widetilde{\boldsymbol{D}}||_f$ denotes the Frobenius matrix norm, $||\,\widetilde{\boldsymbol{D}}||_f = \sqrt{\sum_{l=1}^{L}\sum_{n=1}^{(N+1)^2}\widetilde{D}_{l,n}^2}$

12.4.3.3.3 Matrix design for 2D loudspeaker setups

A 2D loudspeaker setup is detected if all loudspeaker elevation positions are within 7 degrees to the horizontal plane or expressed in loudspeaker inclinations $\hat{\theta}_l$:

$$if\ (\hat{\theta}_l > \frac{\pi}{180}(90-7)\ \&\&\ \hat{\theta} < \frac{\pi}{180}(90+7)) \text{ is true for all loudspeakers } l = 1,2,..,L.$$

Then two more (virtual) loudspeakers are added so that the new number of loudspeakers L_2 becomes $L_2 = L + 2$ and the new loudspeaker directions are given by:

$\mathfrak{D}_{L_2} = [\,\widehat{\boldsymbol{\Omega}}_1, \ldots, \widehat{\boldsymbol{\Omega}}_{\mathrm{L}} \ldots, \widehat{\boldsymbol{\Omega}}_{\mathrm{L+1}} \ldots, \widehat{\boldsymbol{\Omega}}_{\mathrm{L+2}}]$ with spherical angle $\widehat{\boldsymbol{\Omega}}_{\mathrm{l}} = [\hat{\theta}_l, \hat{\phi}_l]^T$ and $\widehat{\boldsymbol{\Omega}}_{\mathrm{L+1}} = [0,0]^T$ and $\widehat{\boldsymbol{\Omega}}_{\mathrm{L+2}} = [\pi, 0]^T$.

A matrix $\widetilde{\boldsymbol{D_2}} \in R^{L_2 \mathrm{x}\,(\mathrm{N}+1)^2}$ is designed using the design method for 3D loudspeaker setups with the $\mathfrak{D}_{L_2}$ loudspeaker position directions.

A matrix $\boldsymbol{D_2} \in R^{L\,\mathrm{x}\,(\mathrm{N}+1)^2}$ with matrix elements $D_{2_{l,n}}$ is created from $\widetilde{\boldsymbol{D_2}}$ by:

$$D_{2_{l,n}} = \widetilde{D_2}_{l,n} + g\left(\widetilde{D_2}_{L+1,n} + \widetilde{D_2}_{L+2,n}\right),$$

for $= 1,2,..,L\ , n = 1,2,..,(N+1)^2$ and $g = \frac{1}{\sqrt{L}}$.

Finally, the rendering matrix $\boldsymbol{D}$ is derived by repeating the building block – normalize matrix:

$$\boldsymbol{D} = \frac{\boldsymbol{D_2}}{||\,\boldsymbol{D_2}||_f},$$

where $||\,\boldsymbol{D_2}||_f$ denotes the Frobenius matrix norm, $||\,\boldsymbol{D_2}||_f = \sum_{l=1}^{L}\sum_{n=1}^{(N+1)^2} D_{2_{l,n}}^2$.

12.4.3.4 NFC processing

This subclause describes the design of the near-field compensating filters. These are intended to be applied to HOA signals before the building block called “HOA to loudspeaker conversion” in Figure 71.

NFC compensation is advantageously used for spherical microphone recordings or artificial mixing with very close sound sources. In this description it is assumed that the HOA content has maximum order N. For a given HOA component of order n, where n is $0 \le n \le N$, NFC HOA uses $\mathrm{d_n} = \frac{\mathrm{h_0^{(2)}(k\,r_s)}}{\mathrm{h_n^{(2)}(kr_s)}}\frac{\mathrm{h_n^{(2)}(k\,r_{NFC})}}{\mathrm{h_0^{(2)}(k\,r_{NFC})}}$, with $\mathrm{h_n^{(2)}}$ as the spherical Hankel functions of the second kind, $\mathrm{r_s}$ is the source-sweet spot distance, and $\mathrm{r_{NFC}}$ is the compensation radius ($\mathrm{r_{NFC}}$ = NfcReferenceDistance).

The transfer function d_n introduced above applies to any HOA signal of order n. In the renderer, it is implemented in the time domain as an n-th order IIR filter under the Direct Form II, with $n/2$ second order sections (or "cells") for even n, or $(n-1)/2$ second order sections plus one first order section for odd n:

$$H_n(z) = \prod_{q=1}^{n/2} \frac{b_0^q + b_1^q z^{-1} + b_2^q z^{-2}}{a_0^q + a_1^q z^{-1} + a_2^q z^{-2}} \times \frac{b_0^{\frac{n+1}{2}} + b_1^{\frac{n+1}{2}} z^{-1}}{a_0^{\frac{n+1}{2}} + a_1^{\frac{n+1}{2}} z^{-1}}$$

$$= g \prod_{q=1}^{n/2} \frac{1 + b'^q_1 z^{-1} + b'^q_2 z^{-2}}{1 + a'^q_1 z^{-1} + a'^q_2 z^{-2}} \times \frac{1 + b'^{\frac{n+1}{2}}_1 z^{-1}}{1 + a'^{\frac{n+1}{2}}_1 z^{-1}}$$

the right factor (first order cell) being present only for odd orders n.

For a given order n, filter coefficients a_i^q are computed as follows.

1) Set $\tau = r_{NFC}/c$ with the sound speed c having a typical value of 340 m/s.
2) Set $\alpha = 4 f_s \tau$
3) For second order cells (for $1 \le q \le n/2$), derive coefficients a_0^q, a_1^q and a_2^q from conjugate complex roots $X_{n,q}$ and $X_{n,n-q+1} = X_{n,q}^*$ exhibited in Table 213:

$$a_0^q = 1 - 2\frac{\mathrm{Re}(X_{n,q})}{\alpha} + \frac{|X_{n,q}|^2}{\alpha^2}$$

$$a_1^q = -2\left(1 - \frac{|X_{n,q}|^2}{\alpha^2}\right)$$

$$a_2^q = 1 + 2\frac{\mathrm{Re}(X_{n,q})}{\alpha} + \frac{|X_{n,q}|^2}{\alpha^2}$$

4) For odd order filters, derive the coefficients of the additional first order cell as follows:

$$a_0^{\frac{n+1}{2}} = 1 - \frac{X_{n,(n+1)/2}}{\alpha}, \quad a_1^{\frac{n+1}{2}} = -\left(1 + \frac{X_{n,(n+1)/2}}{\alpha}\right)$$

Filter coefficients b_i^q are computed according the same procedure, where coefficients a_i^q just have to be replaced by b_i^q and step 1 is replaced by:

1) Set $\tau = r_{max}/c$

Table 213 — Approximative values X_{nq} of the roots of generalized Bessel polynomials for the first few orders n

n	q		
	1	2	3
1	-2		
2	-3.0000+1.7321*j*		
3	-3.6778+3.5088*j*	-4.6444	
4	-4.2076+5.3148*j*	-5.7924+1.7345*i*	
5	-4.6493+7.1420*j*	-6.7039+3.4853*j*	-7.2935
6	-5.0319+8.9853j	-7.4714+5.2525j	-8.4967+1.7350j

12.4.4 Layered coding for HOA

12.4.4.1 General

This subclause describes the layered coding for HOA sound field representations. First the structure of the compressed HOA sound field representation is described. Based on this description it is shown how the side information is structured in the case of the mode of HOA layered coding. In the following subclauses the initialization of the decoder and the behaviour of the decoder in case of drop outs of enhancement layers are described.

12.4.4.2 Structure of compressed HOA sound field representation

In the following the compressed HOA representation is described and structured from the perspective of layered coding. It can be generally decomposed into the following components:

— A basic compressed HOA sound field representation consisting of a number of complementary components (ID_USAC_SCE, ID_USAC_CPE type of usacElementType in Table 41), being the monaural transport signals representing either predominant sound signals or ambient HOA coefficient signals.

— Basic side information (ID_EXT_ELE_HOA type of usacExtElementType in Table 23) which describes for each of these monaural signals how it spatially contributes to the sound field. This can be separated into the following two different components.

  — Side information related to individual (monaural) transport signals, which is independent of the existence of other transport signals. Such side information may for instance specify a monaural signal to represent a directional signal (meaning a general plane wave) with a certain direction of incidence. Alternatively, a monaural signal may be specified as ambient HOA coefficient signals having certain indices.

  — Side information related to vector-based signals in the mode CodedVVecLength = 1, of which the directional distribution is specified by means of a vector. This side information is dependent on the transmitted coefficient sequences of the original HOA component. In the mentioned mode particular components of this vector are implicitly set to zero and are not part of the compressed vector representation. These components are those with indices equal to those of coefficient sequence of the original HOA representation, which are part of the basic compressed sound field representation. That means that if individual components of the vector are coded, their total number depends on the basic compressed sound field representation, in particular on which coefficient sequences of the original HOA representation it contains.
  If no coefficient sequences of the original HOA representation are contained in the basic compressed sound field representation, the dependent basic side information for each vector-based signal consists of all the vector components and has its greatest size. In the case that coefficient sequences of the original HOA representation with certain indices are added to the basic compressed sound field representation, the vector components with those indices are removed from the side information for each vector-based signal, thereby reducing the size of the dependent basic side information for the vector-based signals.

— Enhancement side information (ID_EXT_ELE_HOA_ENH_LAYER type of usacExtElementType in Table 23) to parametrically improve the basic compressed HOA sound field representation consisting of the following components:

— Parameters related to the (broadband) spatial prediction to (linearly) predict missing portions of the sound field from the directional signals.

— Parameters related to the sub-band directional signals synthesis and the parametric ambience replication, which allow a frequency dependent, parametric prediction of additional monaural signals to be spatially distributed in order to complement a so far spatially incomplete or deficient compressed HOA representation. The prediction is based on coefficient sequences of the basic compressed sound field representation. An important aspect is that the mentioned complementary contribution to the sound field is represented within the compressed HOA representation not by means of additional quantized signals, but rather by means of extra side information of a comparably much smaller size. Hence, the two mentioned coding tools are especially suited for the compression of HOA representations at low data rates.

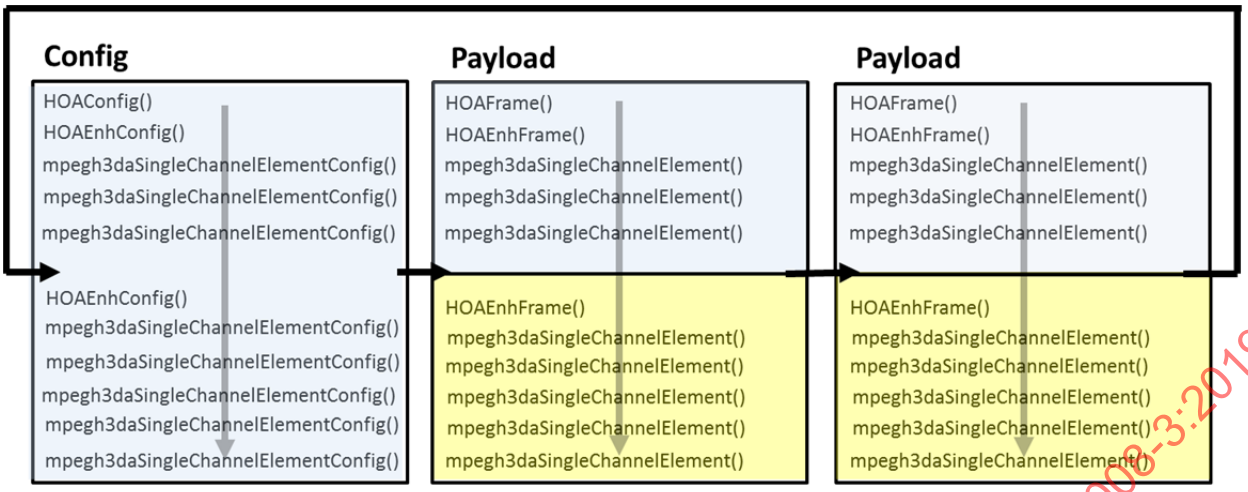
The layered coding mode is indicated by SingleLayer==0 in the HOADecoderConfig(). Referring to the above described structure of the compressed HOA sound field representation, the base layer for one single frame is composed of the basic side information contained in the HOAFrame(), the payloads containing the transport signals included in the base layer (SCEs and/or CPEs) and zero or one optional payload HOAEnhFrame() with enhancement side information.

Each enhancement layer contains payloads with additional transport signals (SCEs and/or CPEs) and zero or one optional payload HOAEnhFrame() with corresponding enhancement side information adapted to the increased number of transport signals.

This structuring of the information contained in the bitstream corresponds to the different types of HOA extension elements provided in the mpegh3daExtElement()s. The same structuring of information can be found in the corresponding configuration information, which consists of the configuration information for the HOA transport channels (ID_USAC_SCE and ID_USAC_CPE in Table 18) and the HOA configuration information separated into base (ID_EXT_ELE_HOA) and enhancement (ID_EXT_ELE_HOA_ENH_LAYER) configuration information. Each layer has its own enhancement configuration information. Whereas only the base layer (**LayerIdx** == 0) additionally contains the basic HOA configuration information. Note: in case of single layer HOA coding the enhancement information in the configuration, as well as in the frame payload is embedded into the basis information and not sent as separate extension payload packet.

In case of layered coding the information of the sound field is still part of one and the same SignalGroup of SignalGroupTypeHOA.

Figure 75 shows an example of the ordering of MHAS packets in a stream for two layer HOA coding.



NOTE The packets are sent starting from the left box to the right box and in the boxes from top to bottom. The light blue blocks compile the base layer and the yellow blocks compile the enhancement layer.

Figure 75 — Example of the ordering of MHAS packets in a stream for HOA coding with 2 layers (base and enhancement layer)

The figure clarifies an exception for the general ordering of the payloads in 3daFrame(). Generally, the information of one SignalGroup is ordered in such a way that, firstly, all extension payloads are sent followed by the SCE and CPE packets belonging to this group. In case of layered coding this convention is modified to allow for easy extraction of the packets belonging to the different layers. As shown in Figure 75 the optional HOAEnhFrame() packet shall directly precede the SCE or CPE packets belonging to the same layer. The partitioning of the payload information and the ordering shall be the same as signalled in the HOAConfig().

Note: To enable a reasonable separation of the HOA transport channels into the base and enhancement layers, described by their assignment to the mpegh3daChannelElments(), the parameter MinAmbHoaOrder in HOADecoderConfig() has to be set to -1 by the encoder. If not set to -1 then the transport channels corresponding to the ambiance which is always transmitted, described by the minimum ambiance order, would be in the highest layers, i.e. in the last mpegh3daChannelElments(). Typically, this information should be sent in the base layer. In case MinAmbHoaOrder=-1 the assignment of the transport channels to the mpegh3daChannelElments() is completely flexible and can be defined by the encoder.

12.4.4.3 Initialization of the decoder for HOA layered coding

As already shown in Figure 75, the complete configuration information shall be sent in the base layer. This ensures that for the initialization of the decoder, e.g. in case of a configuration change, all configuration information is available to initialize the complete decoder including all necessary core decoder instances for all layers. This means the decoder is initialized in the same way as in the single layer mode.

12.4.4.4 Decoder behaviour in HOA layered coding mode

The information contained in the different layers is incremental. This means, the information of an enhancement layer can only be meaningfully decoded if all lower layers are present. For example, in case of three layers, the third layer can only be decoded if the base layer and the first enhancement layer are both available. For the HOA spatial decoding process only the HOAEnhFrame() information of

the highest available layer, for which all lower layers are also present, may be used. All other HOAEnhFrame() packets shall not be used.

In case an enhancement layer is not available, in the loop over all elements of the frame in mpegh3daFrame() the corresponding elements of the missing enhancement layer will not be present. To allow for consistently similar processing of all decoder instances any missing information starting with the first missing enhancement layer may be replaced by dummy payloads and the core decoder instances may be set to concealment mode and their output signals faded out appropriately. The parsing of the HOAEnhFrame() blocks are skipped for non-available layers and the HOAEnhFrame() of the next lower layer shall be used for decoding.

After a dropout of an enhancement layer the decoding can only be resumed at a random access point. If the random access point is an IPF, the pre-roll information should be used start the decoding of the newly available HOA transport channels. An appropriate fade-in may be performed on the output signals of the transport channels.

13 Binaural renderer

13.1 General

Two binaural rendering tools are specified in subsequent subclauses:

— time-domain binaural renderer;

— frequency-domain binaural renderer.

13.2 Frequency-domain binaural renderer

13.2.1 General

The frequency-domain binaural renderer may be used for generating the 3D audio headphone signal for all types of input content (channel and/or object and/or HOA). The Frequency-domain binaural renderer takes loudspeaker feeds as input signals.

The frequency-domain binaural processing is carried out as a decoder process converting the decoded signal into a binaural downmix signal that provides a surround sound experience when listened to over headphones.

The binaural renderer has as input the decoded data stream. The signal is processed by a QMF analysis filterbank as outlined in ISO/IEC 14496-3:2009, 4.B.18.2 with the modifications stated in ISO/IEC 14496-3:2009, 8.6.4.2. The renderer may also process QMF domain input data and in this case the analysis filterbank is omitted.

The binaural room impulse responses (BRIRs) are represented as complex QMF domain filters. The conversion of the time domain binaural room impulse responses to the complex QMF filter representation shall be as defined in ISO/IEC 23003-1:2007, Annex B.

The frequency-domain binaural renderer consists of three processing blocks, a variable order filtering in the frequency domain (VOFF), a sparse frequency reverberator (SFR), and a QMF domain tapped-delay line (QTDL). Figure 76 illustrates the time-frequency processing regions for the three processing blocks.

The QMF domain BRIRs are truncated such that they only contain direct sound and early reflections (D&E). The transition point from early reflections to late reverberation N_{Filter} is determined in a

frequency-dependent manner by a bandwise reverberation time analysis. The QMF domain audio signals and the QMF domain D&E BRIRs are then processed by a bandwise partitioned fast convolution to perform the binaural processing. Since the filter order of the QMF domain D&E BRIRs is frequency dependent, this processing is referred to as *variable order filtering in the frequency domain* (VOFF).

A QMF domain sparse frequency reverberator (SFR) is used to generate 2-channel QMF domain late reverberation. The reverberation module uses a set of frequency-dependent reverberation times and energy values to adapt the characteristics of the reverberation. The waveforms of the reverberation are based on a stereo downmix of the audio input signal and it is adaptively scaled in amplitude depending on an analysis of the audio signal.

QMF domain tapped-delay line (QTDL) processing is used for high frequency bands. The QTDL processing module employs a time lag and a gain for each band pf each channel to mimic the most significant binaural cues in the band. The VOFF and the QMF domain reverberator are turned off in the bands where the QTDL processing is active.

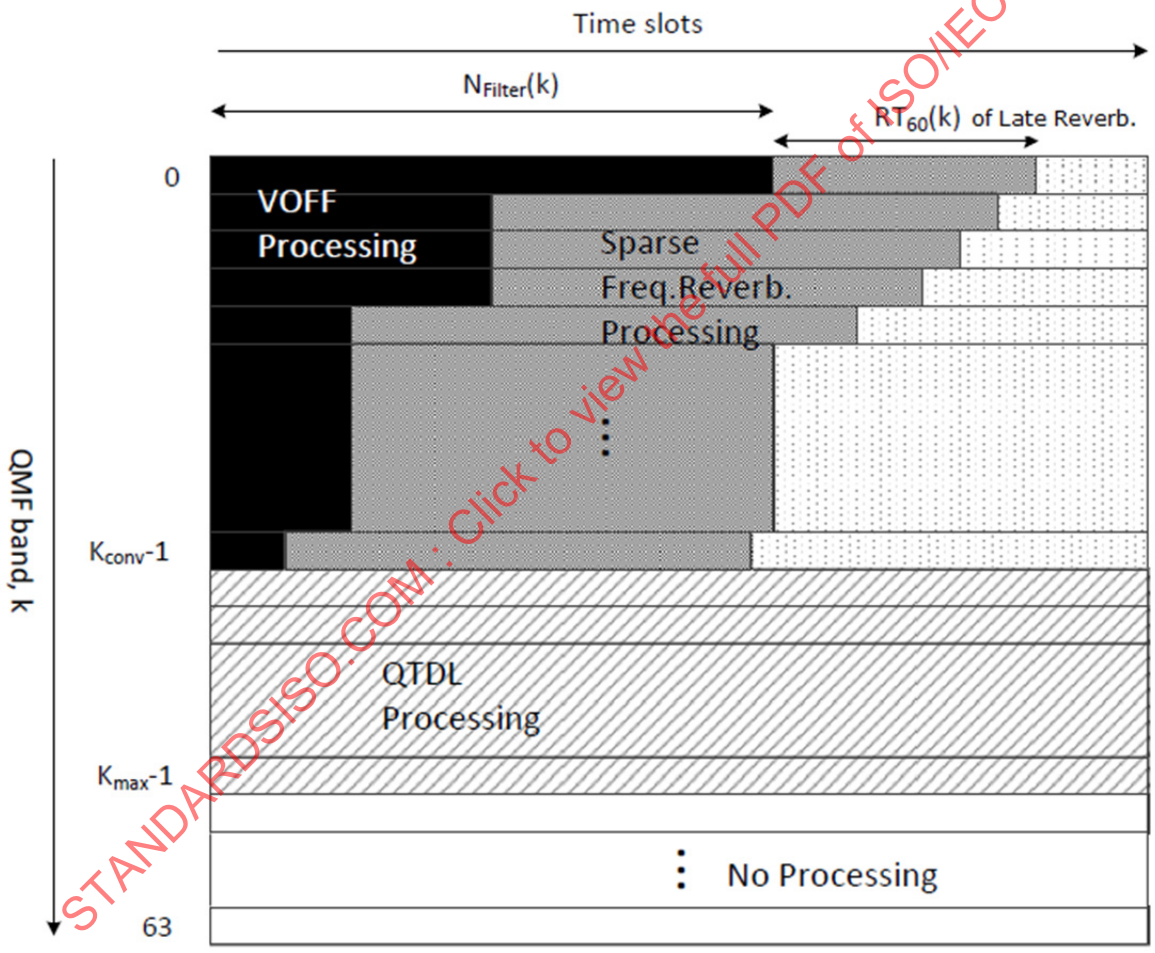


Figure 76 — Graphical representation of the processing regions for the three processing blocks of the binaural renderer

The 2-channel VOFF output and the 2-channel SFR output are then combined and mixed with the QTDL processed output and finally, two QMF synthesis filter banks compute the binaural time domain output signals as outlined in ISO/IEC 14496-3:2009, 4.6.18.4.2. The renderer may also produce QMF domain output data in which case the synthesis filterbank is omitted.

An overview of the QMF domain binaural processing is given in Figure 77.

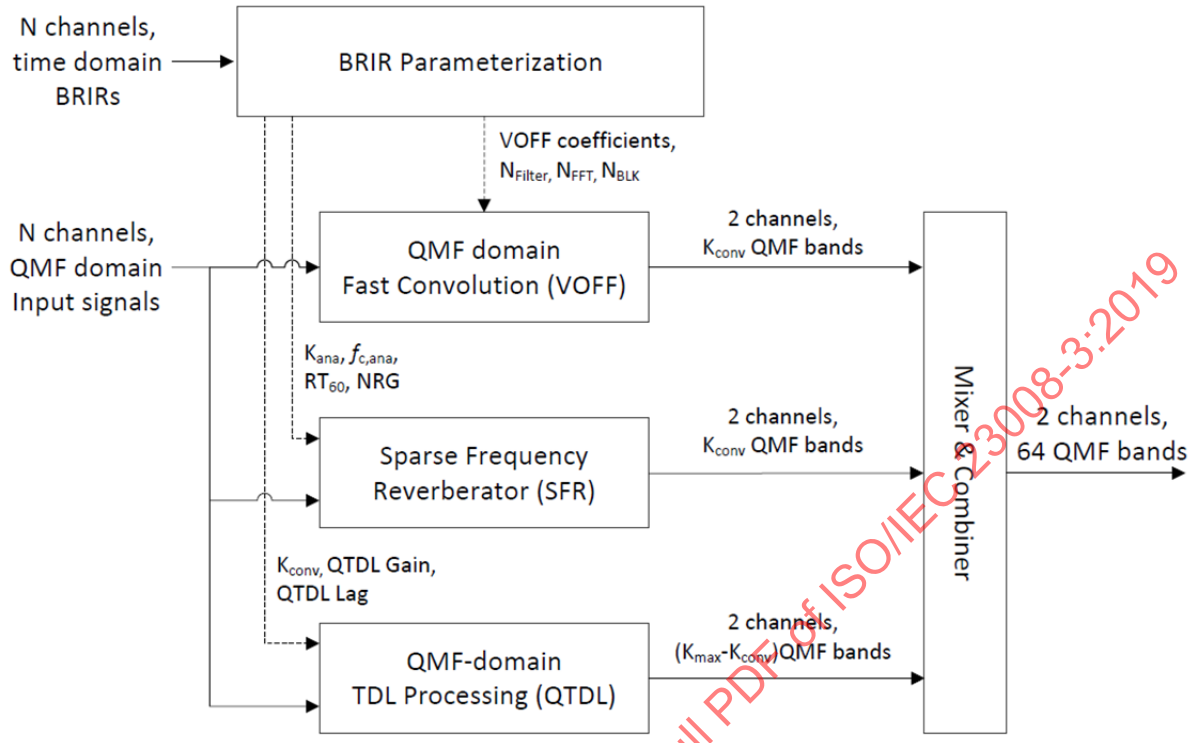


Figure 77 — Overview of the binaural processing

13.2.2 Definitions

Audio signals that are fed into the binaural renderer are referred to as *input signals* in the following. Audio signals that are the result of the binaural processing are referred to as *output signals*. The input signals of the binaural renderer are audio output signals from the core decoder.

The following variable definitions are used.

α	Azimuth angle
a	Factor in downmix matrix
b_{reg}	Axis intercept of a regression line
β	Elevation angle
c	BRIR output channel index (left/right), $c \in \{0,1\}$
c_{corr}	Correlation coefficient
$c_{\text{eq},k}$	Energy equalizing factor
$\mathbf{c}_{\text{scale}} = [c_{\text{scale},0}, c_{\text{scale},1}]$	Scaling factors
$\tilde{\mathbf{c}}_{\text{scale}} = [\tilde{c}_{\text{scale},0}, \tilde{c}_{\text{scale},1}]$	Smoothed scaling factors

c_{60}	Real-valued correction factor for the calculation of the RT60 reverberation time
EDC	Energy decay curve
d_{FB}	Frequency-dependent group delay of filter bank
d	Delay in time domain samples
d_{init}	Initial delay of a BRIR set in time domain samples, propagation time
$d_{i,m}^{k}$	QTDL lag
$f_{c,\mathrm{ana}}$	Centre frequencies of the late reverberation analysis bands
f_s	Sampling rate
$g_{i,m,real}^{k}$	Real value of QTDL Gain
$g_{i,m,imag}^{k}$	Imaginary value of QTDL Gain
$\tilde{h}$	1-channel time domain impulse response
$\tilde{\mathbf{h}}_0^{v} = [\tilde{h}_{0,0}^{v} \cdots \tilde{h}_{0,\mathrm{N_{BRIR}}-1}^{v}]$	Time domain representation of the left impulse response set excluding the LFE channels
$\tilde{\mathbf{h}}_1^{v} = [\tilde{h}_{1,0}^{v} \cdots \tilde{h}_{1,\mathrm{N_{BRIR}}-1}^{v}]$	Time domain representation of the right impulse response set excluding the LFE channels
$\hat{\mathbf{h}}_0^{n,k} = [\hat{h}_{0,0}^{n,k} \cdots \hat{h}_{0,\mathrm{N_{BRIR}}-1}^{n,k}]$	Complex valued QMF domain representation of the left impulse response set excluding the LFE channels
$\hat{\mathbf{h}}_1^{n,k} = [\hat{h}_{1,0}^{n,k} \cdots \hat{h}_{1,\mathrm{N_{BRIR}}-1}^{n,k}]$	Complex valued QMF domain representation of the right impulse response set excluding the LFE channels
$\breve{\mathbf{H}}_{\mathbf{0}}^{\boldsymbol{n,k,l}} = [\breve{H}_{0,0}^{n,k,l} \cdots \breve{H}_{0,\mathrm{N_{BRIR}}-1}^{n,k,l}]$	VOFF Coefficient for output channel 0 (Left) - Block-based pseudo-FFT domain representation of the left impulse response set excluding the LFE channels
$\breve{\boldsymbol{H}}_{\mathbf{1}}^{\boldsymbol{n,k,l}} = [\breve{H}_{1,0}^{n,k,l} \cdots \breve{H}_{1,\mathrm{N_{BRIR}}-1}^{n,k,l}]$	VOFF Coefficient for output channel 1 (Right) - Block-based pseudo-FFT domain representation of the right impulse response set excluding the LFE channels
$\tilde{h}_{k_{\mathrm{FB}}}$	Filtered version of a 1-channel time domain impulse response
$\tilde{h}_{k_{\mathrm{FB}},\mathrm{dB}}$	Filtered normalized squared logarithmic 1-channel BRIR in analysis band k_{FB}
$\acute{h}_{k_{\mathrm{FB}},\mathrm{dB}}$	Filtered smoothed logarithmic 1-channel BRIR in analysis band k_{FB}

$\tilde{h}_{k_{\mathrm{FB}},\mathrm{norm}}$	Normalized filtered 1-channel BRIR
K_{ana}	Number of analysis frequency bands
K_{ch}	Number of BRIR pairs in a BRIR data set (number of measured positions)
K_{conv}	Number of QMF bands used for convolution
K_{LFE}	Number of LFE BRIR pairs in a BRIR data set
K_{LR}	Number of late reverberation analysis bands that are fed into the reverberator
K_{max}	Number of QMF bands used for binaural processing
K_{trans}	Number of QMF bands used in the analysis of the transition from early to late reflections
k	QMF domain frequency band index
K	Number of QMF domain frequency bands, $K = 64$
k_{FB}	Frequency band index of the late reverberation analysis
k_s	Number of windows for the smoothing of a BRIR
L	Length of audio frame in time domain samples, L = outputFrameLength
l_{ana}	Analysis length of the late reverberation analysis
L_{BRIR}	Length of the measured BRIRs in time domain samples
L_n	Length of an audio frame in QMF domain time slots
$\mathbf{m}_{\mathrm{conv}}$	Vector to signal which channel of the input signal corresponds to which BRIR pair in the BRIR data set
$\mathrm{M}_{\mathrm{DMX}}$	Downmix matrix
m_{minv}	Frequency-dependent minimum value in dB
m_{reg}	Gradient of a regression line
n	QMF domain time slots index
N_{Blk}	Number of blocks
N_{BRIR}	Number of BRIRs used in the parameterization and processing
$N_{\mathrm{DMX,act}}$	Number of active downmix channels

n_{DS}	Maximum amplitude of a smoothed logarithmic BRIR
$\tilde{n}_{DS}$	Mean amplitude around maximum of a smoothed logarithmic BRIR
N_{Frm}	Number of sub-frames
N_{in}	Number of input audio channels
N_{out}	Number of output audio channels, $N_{out} = 2$
$\boldsymbol{N}_{Filter}[k]$	VOFF filter order corresponding to the k-th QMF domain frequency band
$\boldsymbol{N}_{FFT}[k]$	FFT length
n_{noise}	Noise level in dB
NRG_{LR}	Energy of the late reverberation
P_i	Position (Azimuth and elevation angle) of a BRIR or audio channel
RT_{30} , RT_{60}	Reverberation times
ν	Time domain sample index
ν_0, ν_5, ν_{35}	Time domain indices with specified amplitude in the EDC
ν_{cross}	Crossing sample of two lines
ν_{DS}	Index of the maximum of a smoothed logarithmic BRIR
ν_{end}	Maximum frequency-independent sample index used for the late reverberation analysis
$\nu_{end,f}$	Frequency-dependent ending time domain sample index of the late reverberation analysis
ν_{noise}	Index where a smoothed logarithmic BRIR is smaller than $n_{noise} + 2$ for the first time
ν_{start}	Frequency-dependent starting time domain sample index of the late reverberation analysis
ν_{trans}	Frequency-dependent transition from early reflections to late reverberation in time domain samples (beginning from the direct sound)
$\tilde{\mathbf{y}}^{\nu} = [\tilde{y}_0^{\nu} \ \cdots \tilde{y}_{N_{in}-1}^{\nu}]$	N_{in}-channel time domain input audio signal
$\hat{\mathbf{y}}^{n,k} = [\hat{y}_0^{n,k} \ \cdots \hat{y}_{N_{in}-1}^{n,k}]$	N_{in}-channel QMF domain input audio signal

$\hat{\mathbf{z}}^{n,k}$	Combined QMF domain signal of convolution output, reverb generator output and QTDL processing output
$\tilde{\mathbf{z}}^{v} = [\tilde{z}_0^{v}, \tilde{z}_1^{v}]$	2-channel time domain output signal
$\check{\mathbf{z}}_{\text{conv}}^{n,k} = [\check{z}_{0,\text{conv}}^{n,k}, \check{z}_{1,\text{conv}}^{n,k}]$	VOFF processed signal in QMF domain frequency band k
$\hat{\mathbf{z}}_{\text{rev}}^{n,k} = [\hat{z}_{0,\text{rev}}^{n,k}, \hat{z}_{1,\text{rev}}^{n,k}]$	Intermediate reverberation signal generated by the reverberator module in QMF domain frequency band k
$\hat{\mathbf{z}}_{\text{QTDL}}^{n,k} = [\hat{z}_{0,\text{QTDL}}^{n,k}, \hat{z}_{1,\text{QTDL}}^{n,k}]$	QTDL processed signal in QMF domain frequency band k

13.2.3 Parameterization of binaural room impulse responses

13.2.3.1 General

The frequency-domain binaural renderer requires specific metadata information describing the properties of the BRIR set. These values are calculated by a parameterization procedure and are stored in a dedicated file in a defined order. The file is a binary file written with 32 bits per sample, float values, little-endian ordering.

The parameterization takes into account as input parameters:

— N channels, time domain BRIRs

— Corresponding elevation and azimuth angles

It returns four groups of parameters:

— General metadata: Elevation and azimuth of the BRIRs, the propagation time d_{init}, the number of processing band K_{max}, the number of LFEs K_{LFE}, the LFE channel indices, the sampling rate of the BRIRs;

— VOFF parameters: The left/right VOFF coefficients $[\breve{\mathbf{H}}_{\mathbf{0}}^{\boldsymbol{n,k,l}}\ \breve{\mathbf{H}}_{\mathbf{1}}^{\boldsymbol{n,k,l}}]$, the VOFF filter length $\boldsymbol{N}_{\text{Filter}}$, the FFT size per band $\boldsymbol{N}_{\text{FFT}}$ and the number of blocks per bands $\boldsymbol{N}_{\text{Blk}}$;

— Reverberator parameters: the number of analysis bands K_{ana} used in the analysis of late reverberation, the centre frequencies of the late reverberation analysis bands $f_{c,\text{ana}}$, the reverberation time RT_{60} and energy NRG_{LR} of the late reverberation;

— QTDL parameters: the left/right QTDL gain $[g_{i,m,real}^{k}\ g_{i,m,imag}^{k}]$, the left/right QTDL time lag $d_{i,m}^{k}$, the maximum band used for convolution K_{conv}.

An overview of the BRIR parameterization is given in Figure 78.

In the block *Propagation Time Calculation*, the propagation time of the BRIRs is calculated to truncate them at the direct sound. In the block "Filter Converter", the truncated time domain BRIRs shall be transformed to QMF domain BRIRs according to ISO/IEC 23003-1:2007, Annex B. The prototype filter coefficients for the filter conversion shall be used according to ISO/IEC 23003-1:2007, Table B.1.

In the block *VOFF Parameter Generation*, a frequency-dependent RT20 reverberation time analysis is performed to determine the frequency-dependent transition from early reflections to late reverberation. Next, the truncated complex-valued QMF domain BRIRs up to the frequency-dependent transition are

transformed into a VOFF coefficients $[\check{h}_0^{n,k,l}\ \check{h}_1^{n,k,l}]$. The VOFF coefficients $[\check{h}_0^{n,k,l}\ \check{h}_1^{n,k,l}]$ will be used to perform a fast convolution in the complex-valued QMF domain.

In the block *Sparse Frequency Reverberator Parameter Generation* (SFR Parameter Generation), the time domain BRIRs are analysed by a one-third octave filter bank to determine the characteristics of the late reverberation. The reverberation time RT60 of the late reverberation and energy of the reverberation are used to control the sparse frequency reverberator.

In the block *QTDL Parameter Generation*, the peak locations and square root of real and imaginary energy of QMF domain BRIRs are analysed to obtain QTDL parameters used in QTDL processing.

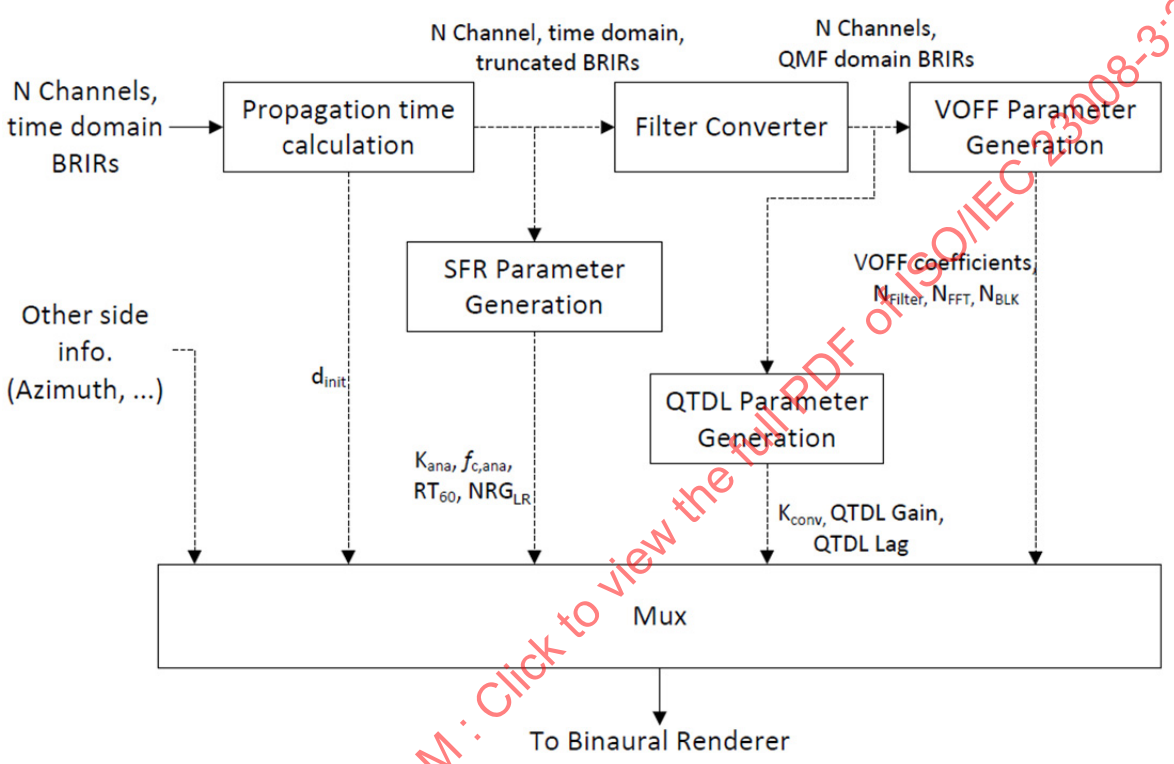


Figure 78 — Details of BRIR parameterization

The analysis steps are specified in detail in the following subclauses.

13.2.3.2 Propagation time d_{init} calculation

The BRIRs contain a redundant portion between the 0th sample and the location of the impact of the direct sound. The time between the start of the BRIR and the direct sound is referred to as the propagation time. Since this portion only imposes additional delay for the binaural rendered output as well as additional complexity for processing, the BRIRs are trimmed to start at the direct sound. Therefore, the propagation time d_{init} is determined by calculating the frame energy.

The following intermediate parameters are determined depending on the overall length of the input BRIRs.

If the length of the BRIRs in time domain samples exceeds 80 ms:

$flag\_HRIR = 0,$
$L_{frm} = 32,$

N_{hop} = 8, and
$N_{iter} = 80e^{-3} f_s$ / N_{hop}

else

flag_HRIR = 1,
L_{frm} = 8,
N_{hop} = 1, and
$N_{iter} = 80e^{-3} f_s$ / N_{hop}

Then, the frame energy is obtained by

$$E(k) = \frac{1}{2N_{BRIR}} \sum_{m=1}^{N_{BRIR}} \sum_{i=0}^{1} \frac{1}{L_{frm}} \sum_{n=0}^{L_{frm}-1} \tilde{h}_{i,m}^{kN_{hop}+n} \quad k = 1, \ldots, N_{iter}$$

resulting in the propagation time:

$$d_{init} = \frac{L_{frm}}{2} + N_{hop} * min\left[\arg_k\left(\frac{E(k)}{\max(E)} > -60dB\right)\right].$$

The sets of truncated left and right BRIRs (excluding the LFE channels) are then defined as:

$$\tilde{\mathbf{h}}_0^v = \left[\tilde{h}_{\mathbf{0,0}}^{v+d_{init}} \cdots \tilde{h}_{0,N_{BRIR}-1}^{v+d_{init}}\right] \text{ and } \tilde{\mathbf{h}}_1^v = \left[\tilde{h}_{1,0}^{v+d_{init}} \cdots \tilde{h}_{1,N_{BRIR}-1}^{v+d_{init}}\right],$$

with:

$$N_{BRIR} = 2 \cdot K_{ch} - 2 \cdot K_{LFE} \text{ and } 0 \le \nu < L_{BRIR} - d_{init} .$$

13.2.3.3 Filter conversion from time-domain BRIRs to QMF-domain BRIRs

The conversion of the time domain filters to the complex-valued QMF domain shall be carried out according to ISO/IEC 23003-1:2007, Annex B. The prototype filter coefficients for the filter conversion shall be used according to ISO/IEC 23003-1:2007, Table B.1.

With the time domain representation of the impulse response sets excluding the LFE channels $[\tilde{\mathbf{h}}_0^v \; \tilde{\mathbf{h}}_1^v]$, a complex-valued QMF domain representation shall be obtained according to ISO/IEC 23003-1:2007, Annex B as:

$$\hat{\mathbf{h}}_0^{n,k} = \left[\hat{h}_{0,0}^{n,k} \cdots \hat{h}_{0,N_{BRIR}-1}^{n,k}\right] \text{ and } \hat{\mathbf{h}}_1^{n,k} = \left[\hat{h}_{1,0}^{n,k} \cdots \hat{h}_{1,N_{BRIR}-1}^{n,k}\right]$$

with

$$0 \le n < n_{end}$$

$$n_{end} = \lceil (L_{BRIR} - d_{init})/64 \rceil + 2$$

13.2.3.4 Definition of the number of processing bands and convolution bands

The maximum processing band K_{max} and the number of convolution bands K_{conv} can be received from Table 251 (**kMax** and **kConv**, respectively). In the bands 0 to K_{conv} –1, the VOFF processing shall be performed in the bands K_{conv} to K_{max} –1, the QTDL processing shall be applied. When the QTDL processing is not used, K_{conv} is overridden by K_{max}. The SFR processing shall be performed in the VOFF processing bands as illustrated in Figure 76.

13.2.3.5 Parameter generation for the variable order filtering in frequency domain (VOFF)

13.2.3.5.1 General

A frequency-dependent RT20 reverberation time analysis shall be performed to determine filter order N_{Filter} for the VOFF processing up to band $K_{\text{conv}} - 1$. The filter order N_{Filter} for the VOFF is derived as specified in the following subclauses:

13.2.3.5.2 Derivation of RT20

The energy decay relief of a BRIR is obtained in the complex-valued QMF domain to derive the RT20 reverberation time as a measure for the transition time from the early reflections to the late reverberation.

The energy decay relief of a BRIR (channel m of the BRIR set) in band k is defined as:

$$EDR_{i,m}^{n,k} = \sum_{l=n}^{n_{\text{end}}-1} \left|\hat{\mathfrak{h}}_{i,m}^{l,k}\right|^2$$

Then the averaged reverberation time RT20 is obtained by:

$$RT_{20}^{k} = \frac{1}{2\text{N}_{\text{BRIR}}} \sum_{i=0}^{1} \sum_{m=0}^{\text{N}_{\text{BRIR}}-1} \min\left[\arg_{n}\left(\frac{EDR_{i,m}^{n,k}}{EDR_{i,m}^{0,k}} < -20dB\right)\right]$$

13.2.3.5.3 Filter order decision

The VOFF filter order is determined using the logarithmic curve fitting method for obtaining the reverberation time RT20. The curve fitting shall not be conducted for band 0. In the case of HRIR (*flag_HRIR*=1), curve fitting is not to be applied.

To perform the curve fitting, two coefficients $a, and\ b$ should be obtained.

$$\bar{x} = \frac{1}{K_{\text{max}} - 1} \sum_{k=1}^{K_{\text{max}}-1} k$$

$$\bar{y} = \frac{1}{K_{\text{max}} - 1} \sum_{k=1}^{K_{\text{max}}-1} \log_2 \text{RT}_{20}^{k}$$

$$s_{xx} = \sum_{k=1}^{K_{\text{max}}-1} k^2 - (K_{\text{max}} - 1)\bar{x}^2$$

$$s_{xy} = \sum_{k=1}^{K_{\text{max}}-1} k\left(\log_2 \text{RT}_{20}^{k}\right) - (K_{\text{max}} - 1)\bar{x}\bar{y}$$

$$b = s_{xy}/s_{xx}$$

$$a = \bar{y} - b\bar{x}$$

The filter order N_{Filter} for each band k is derived as:

if $k = 0$

$$N_{Filter}[0] = \min(2^{\lceil log_2 \mathrm{RT}_{20}^0 \rceil}, n_{end})$$

else if $k < K_{\max}$

if *flag_HRIR* = 0

$$N_{Filter}[k] = \min(2^{\lfloor bk+a+0.5 \rfloor}, n_{end})$$

else

$$N_{Filter}[k] = \min(2^{\lfloor log_2 \mathrm{RT}_{20}^k + 0.5 \rfloor}, n_{end})$$

The transition $\nu_{\mathrm{trans}}[k]$ from direct sound and early reflections to the late reverberation tail in time domain samples is calculated from $N_{Filter}[k]$:

$$\nu_{\mathrm{trans}}[k] = 64 \cdot (N_{Filter}[k] - 2)$$

13.2.3.5.4 VOFF coefficient generation

The complex valued QMF domain BRIRs are truncated frequency-dependently at the corresponding value of the VOFF filter order $N_{\mathrm{Filter}}[k]$. The truncated complex valued QMF domain BRIRs are split into blocks to perform a block-wise fast convolution. These block-wise complex valued QMF domain BRIRs are therefore transformed to VOFF coefficients.

First, a frequency-dependent FFT size with a maximum of $2L_N$ is calculated according to:

$$N_{\mathrm{FFT}}[k] = \min(2L_N,\ 2^{\lceil \log_2 2 \cdot N_{\mathrm{filter}}[k] \rceil})$$

As the next step, the number of blocks per QMF band is determined:

$$N_{\mathrm{Blk}}[k] = \frac{2^{\lceil \log_2 2 \cdot N_{\mathrm{filter}}[k] \rceil}}{N_{\mathrm{FFT}}[k]}$$

The truncated complex valued QMF domain representations are obtained as:

$$\tilde{\mathbf{h}}_c^{n,k} = \begin{cases} \hat{\mathbf{h}}_c^{s,k} & n < N_{filter}[k] \\ 0 & otherwise \end{cases}$$

If *flag_HRIR* == 1, the sparse frequency reverberator is switched off. To prevent energy mismatch due to residue of BRIRs, energy compensation is carried out.

$$\tilde{\mathbf{h}}_c^{n,k} = \begin{cases} \sqrt{\frac{\sum_{s=0}^{s=n_{end}-1} real\left(\hat{\mathbf{h}}_c^{s,k}\right)^2}{\sum_{s=0}^{s=N_{\mathrm{filter}}[k]-1} real\left(\hat{\mathbf{h}}_c^{s,k}\right)^2}} real\left(\hat{\mathbf{h}}_c^{n,k}\right) + i \sqrt{\frac{\sum_{s=0}^{s=n_{end}-1} imag\left(\hat{\mathbf{h}}_c^{s,k}\right)^2}{\sum_{s=0}^{s=N_{\mathrm{filter}}[k]-1} imag\left(\hat{\mathbf{h}}_c^{n,k}\right)^2}} imag\left(\hat{\mathbf{h}}_c^{n,k}\right) & n < N_{filter}[k] \\ 0 & otherwise \end{cases}$$

The block-wise complex valued QMF domain representations are obtained by partitioning the truncated complex valued QMF domain representation.

$$\check{\mathbf{h}}_c^{n,k,l} = \begin{cases} \tilde{\mathbf{h}}_c^{(l-1)N_{FFT}[k]/2+n,k} & 0 \le n < N_{FFT}[k]/2 \\ 0 & otherwise \end{cases}$$

In each QMF band k each block of the block-wise complex valued QMF domain representations are then transformed to a pseudo-FFT domain by a complex-valued FFT transform of length $N_{\mathrm{FFT}}[k]$ forming the block-based pseudo-FFT domain representation $\check{\boldsymbol{H}}_0^{n,k,l}$ (left ear) and $\check{\mathbf{H}}_1^{n,k,l}$ (right ear) for the blocks $0 \le l < N_{\mathrm{Blk}}[k]$. The $\check{\boldsymbol{H}}_c^{n,k,l}$ are the VOFF coefficients.

$$\check{\mathbf{H}}_{\boldsymbol{c}}^{\boldsymbol{n,k,l}} = \boldsymbol{FFT}\{\check{\mathbf{h}}_c^{n,k,l}, N_{FFT}[k]\}$$

13.2.3.6 Parameter generation for the sparse frequency reverberator

13.2.3.6.1 General

If *flag_HRIR* == 0, an analysis of the late reverberation parts of the BRIRs is carried out. Therefore, all of the BRIRs excluding the LFE BRIRs are filtered by a one-third octave filter bank.

13.2.3.6.2 One-third octave filterbank analysis

The one-third octave filter bank is realized by an infinite-impulse-response (IIR) filter with the coefficients given in Table 215 and Table 216. The filter coefficients are given in double precision. The filter is designed for a sampling frequency of 48 kHz and a maximum processing frequency of 18 kHz, resulting in 24 one-third octave bands.

If the BRIRs are sampled at a different sampling frequency, the same filter coefficients shal lbe used, resulting again in a maximum processing frequency of $0.75 \cdot f_s$.

The bands of the filtered BRIRs are delay aligned according to the group delay of the filters. The delay values measured in time domain samples are given in Table 214.

Table 214 — Delay for the delay alignment of the filtered BRIRs

Band k'	**Delay** $d_{\mathrm{FB}}[k']$ time domain samples
0	702
1	1448
2	1202
3	870
4	731
5	478
6	368
7	284
8	224
9	177
10	142

11	110
12	88
13	69
14	55
15	44
16	35
17	27
18	22
19	17
20	14
21	11
22	8
23	6

To delay align the frequency bands, they are zero-padded out to the maximum delay number $d_{\mathrm{FB,max}}$ at the beginning. Afterwards, each band is time shifted to the beginning according to its delay number and after that the first $d_{\mathrm{FB,max}}$ samples are again removed.

Table 215 — Forward filter coefficients of the used 1/3 octave filter bank

Band	f_{center} [Hz]	Filter coefficients, forward path (B coefficients)							
0	99,21	0.000000002978966	0	-0.000000008936898	0	0.000000008936898	0	-0.000000002978966
1	125,00	0.000000005899584	0	-0.000000017698751	0	0.000000017698751	0	-0.000000005899584
2	157,49	0.000000011789497	0	-0.000000035368490	0	0.000000035368490	0	-0.000000011789497
3	198,43	0.000000023548606	0	-0.000000070645819	0	0.000000070645819	0	-0.000000023548606
4	250,00	0.000000047026089	0	-0.000000141078268	0	0.000000141078268	0	-0.000000047026089
5	314,98	0.000000093874874	0	-0.000000281624623	0	0.000000281624623	0	-0.000000093874874
6	396,85	0.000000187307829	0	-0.000000561923486	0	0.000000561923486	0	-0.000000187307829
7	500,00	0.000000373504653	0	-0.000001120513960	0	0.000001120513960	0	-0.000000373504653
8	629,96	0.000000744228866	0	-0.000002232686597	0	0.000002232686597	0	-0.000000744228866
9	793,70	0.000001481490914	0	-0.000004444472741	0	0.000004444472741	0	-0.000001481490914
10	1 000,00	0.000002945568317	0	-0.000008836704950	0	0.000008836704950	0	-0.000002945568317
11	1 259,92	0.000005847694618	0	-0.000017543083853	0	0.000017543083853	0	-0.000005847694618
12	1 587,40	0.000011587291750	0	-0.000034761875251	0	0.000034761875251	0	-0.000011587291750
13	2 000,00	0.000022906469387	0	-0.000068719408160	0	0.000068719408160	0	-0.000022906469387
14	2 519,84	0.000045150610814	0	-0.000135451832443	0	0.000135451832443	0	-0.000045150610814
15	3 174,80	0.000088673275133	0	-0.000266019825400	0	0.000266019825400	0	-0.000088673275133
16	4 000,00	0.000173370214378	0	-0.000520110643135	0	0.000520110643135	0	-0.000173370214378
17	5 039,68	0.000337103447126	0	-0.001011310341379	0	0.001011310341379	0	-0.000337103447126
18	6 349,60	0.000651075235971	0	-0.001953225707913	0	0.001953225707913	0	-0.000651075235971
19	8 000,00	0.001247284713280	0	-0.003741854139840	0	0.003741854139840	0	-0.001247284713280
20	10 079,37	0.002366322106562	0	-0.007098966319686	0	0.007098966319686	0	-0.002366322106562
21	12 699,21	0.004438169688363	0	-0.013314509065088	0	0.013314509065088	0	-0.004438169688363
22	16 000,00	0.008214651510826	0	-0.024643954532479	0	0.024643954532479	0	-0.008214651510826
23	20 158,74	0.014980703571592	0	-0.044942110714776	0	0.044942110714776	0	-0.014980703571592

Table 216 — Backward filter coefficients of the used 1/3 octave filter bank

Band	f_{center} [Hz]	Filter coefficients, backward path (A coefficients)						
0	99,21	1	-5,993 751 016 763 135	14,969 279 072 784 566	-19,939 603 992 021 354	14,940 647 795 274 490	-5,970 844 821 630 246	0,994 272 962 360 464
1	125,00	1	-5,991 961 445 134 535	14,960 639 671 319 811	-19,922 939 882 503 549	14,924 596 293 523 440	-5,963 124 398 568 642	0,992 789 761 382 594
2	157,49	1	-5,989 609 767 964 868	14,949 371 631 318 428	-19,901 379 920 011 749	14,904 008 213 044 975	-5,953 314 349 635 694	0,990 924 193 325 314
3	198,43	1	-5,986 493 089 373 522	14,934 568 071 720 706	-19,873 322 981 098 088	14,877 492 822 310 906	-5,940 823 535 229 365	0,988 578 711 974 606
4	250,00	1	-5,982 322 583 166 484	14,914 956 502 895 425	-19,836 561 944 151 942	14,843 176 218 160 425	-5,924 879 679 481 255	0,985 631 486 962 872
5	314,98	1	-5,976 681 836 082 384	14,888 729 126 620 820	-19,788 018 121 780 837	14,798 506 980 371 531	-5,904 466 854 668 645	0,981 930 710 405 792
6	396,85	1	-5,968 963 204 438 088	14,853 287 019 657 653	-19,723 347 050 471 695	14,739 973 841 057 109	-5,878 238 363 575 107	0,977 287 777 184 542
7	500,00	1	-5,958 270 126 407 491	14,804 853 333 473 961	-19,636 351 633 672 902	14,662 693 694 537 232	-5,844 394 434 554 799	0,971 469 244 020 969
8	629,96	1	-5,943 266 716 356 360	14,737 888 463 499 093	-19,518 109 356 078 305	14,559 810 012 283 357	-5,800 509 649 746 865	0,964 187 554 640 025
9	793,70	1	-5,921 945 879 281 351	14,644 209 432 330 609	-19,355 682 487 394 688	14,421 618 545 659 840	-5,743 289 060 834 721	0,955 090 675 488 847
10	1 000,00	1	-5,891 272 053 317 806	14,511 677 079 902 135	-19,130 239 207 390 304	14,234 312 652 347 755	-5,668 224 665 450 618	0,943 751 061 239 065
11	1 259,92	1	-5,846 632 613 931 893	14,322 276 215 973 289	-18,814 390 396 441 560	13,978 227 786 574 614	-5,569 116 536 160 665	0,929 654 821 150 504
12	1 587,40	1	-5,781 001 286 253 467	14,049 405 018 511 159	-18,368 601 439 334 121	13,625 500 348 622 651	-5,437 419 214 533 596	0,912 192 664 215 007
13	2 000,00	1	-5,683 678 077 357 811	13,654 293 601 443 985	-17,736 819 219 552 288	13,137 230 968 058 500	-5,261 383 586 695 196	0,890 655 254 306 507
14	2 519,84	1	-5,538 431 150 435 243	13,081 896 267 444 138	-16,842 288 758 666 236	12,460 762 205 420 412	-5,025 009 003 406 554	0,864 237 092 185 886
15	3 174,80	1	-5,320 855 496 910 868	12,257 825 487 897 835	-15,586 541 172 177 160	11,528 970 962 883 021	-4,706 945 111 668 716	0,832 054 991 343 535
16	4 000,00	1	-4,994 862 939 199 035	11,090 899 392 402 983	-13,858 558 427 508 518	10,266 297 975 027 744	-4,279 774 723 365 612	0,793 189 508 854 480
17	5 039,68	1	-4,508 633 849 835 456	9,492 307 997 838 553	-11,567 081 708 283 867	8,611 562 042 517 182	-3,710 717 285 192 899	0,746 759 879 904 296
18	6 349,60	1	-3,791 566 479 016 410	7,433 709 378 886 649	-8,711 008 375 712 691	6,575 213 188 380 568	-2,965 921 277 956 436	0,692 044 028 301 664
19	8 000,00	1	-2,756 730 866 820 556	5,078 384 616 947 228	-5,476 030 603 444 230	4,351 407 566 193 580	-2,022 277 271 752 888	0,628 653 072 944 205
20	10 079,37	1	-1,319 726 797 918 420	3,001 645 881 863 894	-2,235 723 204 296 885	2,473 284 706 980 526	-0,892 513 256 112 752	0,556 761 178 207 907
21	12 699,21	1	0,543 821 839 225 891	2,367 900 783 391 379	0,842 176 405 781 307	1,858 884 029 029 895	0,331 706 060 878 868	0,477 372 283 671 088
22	16 000,00	1	2,667 654 841 026 451	4,479 530 540 428 327	4,526 976 093 111 068	3,286 179 336 952 533	1,426 496 187 396 251	0,392 570 938 427 190
23	20 158,74	1	4,472 803 785 543 190	8,630 679 353 924 112	9,234 059 565 895 342	5,792 824 924 160 126	2,020 263 319 760 948	0,305 653 299 838 751

In addition to the 1/3 octave filter bank, IIR low pass filtering shall be conducted. The filter coefficients of the IIR LPF are given in Table 217. The resulting 25 frequency bands are now addressed by the index$k_{\mathrm{FB}}$with the lowest band $k_{\mathrm{FB}} = 0$ and the 24 bands from the 1/3 octave filter bank analysis $k_{\mathrm{FB}} = 1$ to $k_{\mathrm{FB}} = 24$. The filtered version of a 1-channel time domain impulse response $\tilde{h}$ in band k_{FB} is called $\tilde{h}_{k_{\mathrm{FB}}}$.

Table 217 — Filter coefficients of the additional low pass filter

Forward path, B coefficients	Backward path, A coefficients
4.07451850037432e-14	1
2.44471110022459e-13	-5.95448186943460
6.11177775056149e-13	14.7734441411088
8.14903700074865e-13	-19.5489429752463
6.11177775056149e-13	14.5509829161372
2.44471110022459e-13	-5.77650412293260
4.07451850037432e-14	0.955501910370147

The transition values $\nu_{\mathrm{trans}}[k]$ from the frequency-dependent RT20 reverberation time analysis are mapped from QMF bands to the 25 analysis frequency bands. For each of the analysis bands the transition from the one specific QMF band is used where the difference of the centre frequency of the analysis band and the QMF band is minimal.

Then, the frequency-dependent value $\nu_{\mathrm{start}}[k_{\mathrm{FB}}] = \nu_{\mathrm{trans}}[k_{\mathrm{FB}}] + d_{\mathrm{init}}$ (transition from early reflections to late reverberation in time domain samples plus the initial delay of the BRIR set) is used as a starting point for the following analysis.

The last maximally used sample for the analysis is calculated by $\nu_{\mathrm{end}} = 2^{\lfloor \log_2(L_{\mathrm{BRIR}}/4) \rfloor + 1}$.

A frequency-dependent terminal point sample is calculated:

$$\nu_{\mathrm{end,f}}[\mathrm{k_{FB}}] = \begin{cases} \nu_{\mathrm{end}} & \text{if } \mathrm{k_{FB}} > 3 \; and \; \mathrm{k_{FB}} < 22 \\ \nu_{\mathrm{end}} - \nu_{\mathrm{start}}[\mathrm{k_{FB}}] & \text{else} \end{cases}$$

For each of the BRIRs, omitting the LFE BRIRs, having non-zero values and for each of the 25 analysis bands the steps in the following subclauses are to be carried out between $\nu_{\mathrm{start}}[k_{\mathrm{FB}}]$ and $\nu_{\mathrm{end},f}[k_{\mathrm{FB}}]$.

13.2.3.6.3 Noise floor estimation

A noise floor estimation is carried out for band $k_{\mathrm{FB}} = 4$ or higher to determine the amount of noise in the filtered BRIRs. Therefore, the filtered BRIR is normalized, squared and the logarithm is calculated:

$$\tilde{\mathrm{h}}_{\mathrm{k_{FB},dB}}[\nu] = 10 \cdot \log_{10}\left(\frac{\tilde{\mathrm{h}}_{\mathrm{k_{FB}}}[\nu] + 10^{-20}}{\max(|\tilde{\mathrm{h}}_{\mathrm{k_{FB}}}|)}\right)^2$$

A smoothed logarithmic BRIR $\acute{h}_{k_{\mathrm{FB}},\mathrm{dB}}$ is then determined by calculating the mean value in overlapping windows (50 % overlap) of 0,002 5 seconds duration. The number of windows is k_s.

It is then determined where the smoothed logarithmic BRIR rises above a defined frequency-dependent minimum value in dB. The minimum value is defined as:

$$m_{minv}[k_{FB}] = \begin{cases} -98 & k_{FB} = 0 \\ m_{minv}[k_{FB} - 1] + 2 & k_{FB} < 22 \\ \left\lfloor \left(\frac{1}{k_s} \cdot \sum_{\nu=0}^{k_s-1} \acute{h}_{k_{FB},dB}[\nu] \right) + 0.5 \right\rfloor & \text{else} \end{cases}$$

resulting in a minimum value of -90dB in band 4.

The last 20 % of the smoothed logarithmic BRIR above the minimum value defined above is used to determine the noise level n_{noise} by a mean-value calculation as defined below.

If the smoothed logarithmic BRIR never rises above the minimum value, the noise level is determined to be the mean value of the whole smoothed logarithmic BRIR.

The time domain sample where the BRIR is assumed to have reached this noise level is then defined by calculating the crossing point of two lines, with the one line being parallel to the noise level plus 2 dB and the other line being a decaying regression line starting near the direct sound with a defined gradient.

The gradient of the regression line is defined as:

$$m_{reg} = \frac{((n_{noise} + 2) - ((n_{DS} + \tilde{n}_{DS})/2)}{\left(\left\lfloor (\nu_{noise} + 1) \cdot \left(0.0025 \cdot \frac{f_s}{2} \right) + 0.5 \right\rfloor \right) - \left(\left\lfloor (\nu_{DS} + 1) \cdot \left(0.0025 \cdot \frac{f_s}{2} \right) + 0.5 \right\rfloor \right)}$$

with

$$\tilde{n}_{DS} = \frac{1}{l} \cdot \sum_{\nu=\max(0,\nu_{DS}-10)}^{\nu_{DS}+10} \acute{h}_{k_{FB},dB}[\nu]$$

$$l = \nu_{DS} + 10 - \max(0, \nu_{DS} - 10) + 1$$

ν_{DS} being the index of the maximum absolute value of the smoothed logarithmic BRIR, n_{DS} the amplitude at ν_{DS} and ν_{noise} being the index where the smoothed logarithmic BRIR is smaller than $n_{noise} + 2$ for the first time. The axis intercept of the regression line is defined as:

$$b_{reg} = \frac{(n_{DS} + \tilde{n}_{DS})}{2} - m_{reg} \cdot \left\lfloor (\nu_{DS} + 1) \cdot \left(0.0025 \cdot \frac{f_s}{2} \right) + 0.5 \right\rfloor.$$

The time domain sample ν_{cross} where the regression line crosses the parallel line for the first time specifies the beginning of the noise floor. This value is used to define the end of analysis in the corresponding frequency bands for the following parameterization steps.

$$\nu_{end,f}[k_{FB}] = \nu_{start}[k_{FB}] + \nu_{cross}[k_{FB}] \qquad \text{for } k_{FB} > 3$$

The analysis length l_{ana} is defined as $\nu_{cross}[k_{FB}] + 1$.

13.2.3.6.4 Determination of the RT60 reverberation time

Between $\nu_{start}[k_{FB}]$ and $\nu_{end,f}[k_{FB}]$ the RT60 time is determined. To achieve this, the EDC (energy decay curve, Schroeder integral) of the filtered BRIR is calculated by performing the following steps.

— The filtered BRIR is normalized:

$$\tilde{h}_{k_{FB},norm}[\nu] = \frac{|\tilde{h}_{k_{FB}}[\nu]|}{\max(|\tilde{h}_{k_{FB}}|)}$$

— A factor of 1.0E-20 is added where $h_{k_{FB},norm}[\nu]$ is equal to zero.

— The EDC is defined as:

$$EDC[\nu] = 10 \cdot \log_{10}\left(\frac{\sum_{k=\nu}^{l_{ana}-1} \tilde{h}_{k_{FB},norm}[k]}{\sum_{k=0}^{l_{ana}-1} \tilde{h}_{k_{FB},norm}[k]}\right)$$

Index ν_0 is defined as the index of the sample where the EDC is bigger than zero for the last time. If such an index does not exist, then ν_0 is set to zero.

Initially, the index is determined where the EDC gets smaller than $EDC[\nu_0] - 5$ for the first time. This index is called index ν_5. Next, the index is determined where the EDC gets smaller then $EDC[\nu_5] - 35$ for the first time. Accordingly, this index is called index ν_{35}. The number of samples between ν_5 and ν_{35} defines the reverberation time RT30. To obtain the reverberation time RT60, the RT30 value is multiplied by a factor of $c_{60} = 2$.

An additional correction of the calculated RT60 reverberation time is introduced. If ν_{35} is larger than $l_{ana} \cdot 0.95$ then ν_{35} is set to:

$$\nu_{35} = \lfloor l_{ana} \cdot 0.95 + 0.5 \rfloor$$

and the correction factor is defined as:

$$c_{60} = \frac{-65}{EDC[\nu_{35}]}$$

If ν_5 is larger than $l_{ana} \cdot 0.5$, then ν_5 is set to zero.

The frequency-dependent reverberation time RT60 is calculated by:

$$RT_{60} = RT_{30} \cdot c_{60} = \left(\frac{\nu_{35}}{f_s} - \frac{\nu_5}{f_s}\right) \cdot c_{60}$$

13.2.3.6.5 Determination of the energy

The energy (absolute value squared) of each sample between $\nu_{start}[k_{FB}]$ and $\nu_{end}[k_{FB}]$ is summed to give the total energy of the late reverberation:

$$NRG_{LR} = \sum_{\nu=0}^{l_{ana}-1} |\tilde{h}_{k_{FB}}[\nu]|^2.$$

The values of RT60 and the energy are now averaged for each of the analysis bands to reflect the characteristics of the whole BRIR set.

13.2.3.7 QTDL parameter generation

The QMF domain TDL processing is an efficient binaural rendering tool for the high frequency bands. In a QTDL-enabled QMF band, a single-tap delay-line FIR filter mimics the most significant binaural cues of the interaural time difference (ITD) and the interaural level difference (ILD) in the band.

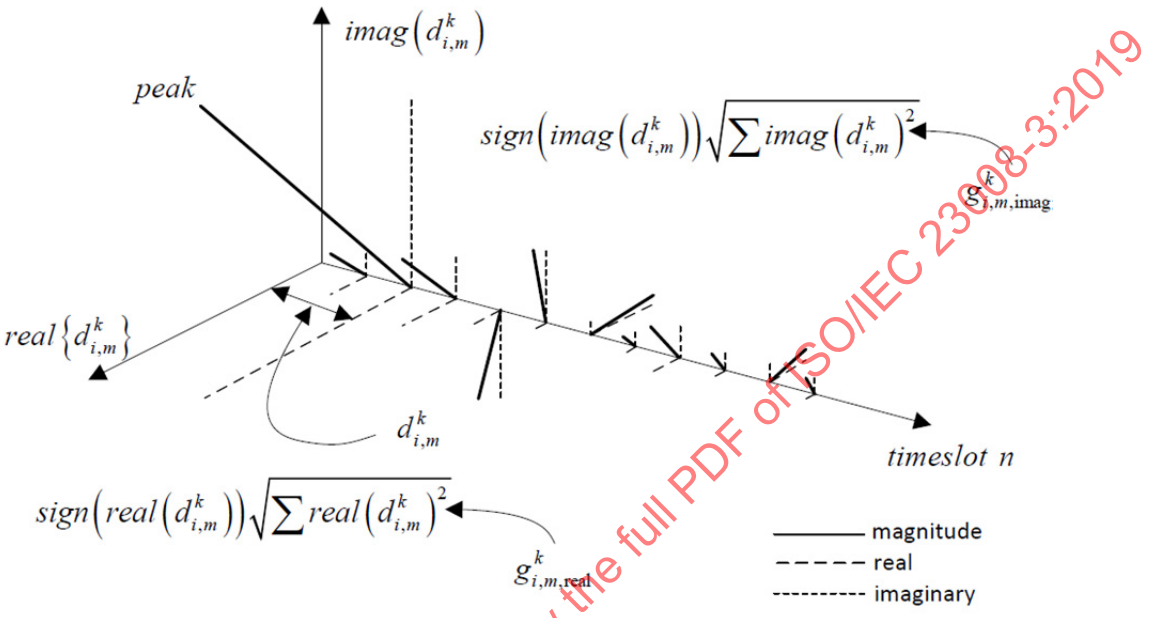


Figure 79 — Geometric descriptions of QTDL parameters

As described in Figure 79 the parameter generation for QTDL is carried out to obtain a QTDL lag parameter and two QTDL gain parameters per output channel (left/right), per band from K_{conv} to K_{max}-1, and per input channel.

The QTDL lag $d_{i,m}^{k}$ is obtained as the location of the maximum peak of the magnitude of the complex-valued QMF domain representation.

$$d_{i,m}^{k} = \arg\max_{n}(|\hat{\mathrm{h}}_{i,m}^{n,k}|^{2})$$

The QTDL gain $g_{i,m}^{k}$ is the square root of the energy of the real/imaginary values of QMF domain representation.

$$g_{i,m,real}^{k} = \text{sign}\left\{real\left(\hat{\mathrm{h}}_{i,m}^{d_{i,m}^{k},k}\right)\right\}\sqrt{\sum_{l=0}^{n_{end}} real(\hat{\mathrm{h}}_{i,m}^{l,k})^{2}}$$

$$g_{i,m,imag}^{k} = \text{sign}\left\{imag(\hat{\mathrm{h}}_{i,m}^{d_{i,m}^{k},k})\right\}\sqrt{\sum_{l=0}^{n_{end}} imag(\hat{\mathrm{h}}_{i,m}^{l,k})^{2}}$$

13.2.3.8 Multiplexing of BRIR parameters

The calculated BRIR metadata information is stored according to the defined interface (see subclause 17.4.2).

13.2.4 Frequency-domain binaural processing

The frequency-domain binaural renderer operates on contiguous, non-overlapping frames of length L samples of the input audio signals and outputs one frame of L samples. A standard frame length of L = 4096 is defined.

13.2.4.1 Initialization and preprocessing

13.2.4.1.1 General

The initialization of the binaural renderer is carried out before the processing of the audio samples delivered by the core decoder takes place. The initialization consists of several processing steps.

13.2.4.1.2 Reading of BRIR metadata and pseudo-FFT BRIRs

The binaural renderer reads BRIR metadata according to the defined interface (see subclause 17.4.2).

13.2.4.2 Audio signal processing

13.2.4.2.1 General

The audio processing of the binaural renderer obtains N_{in} input channels from the core decoder and generates a binaural output signal consisting of N_{out}= 2 channels.

The processing takes as input:

— the decoded audio data from the core decoder;

— the BRIR information from the parameterization, as is:

  — the general BRIR side information;

  — the VOFF coefficients set and side information;

  — the frequency-dependent parameter set that is used by the QMF domain reverberator to generate the late reverberation;

  — and the QTDL gains and QTDL time lag set and side information;

— and a parameter signalling the channel configuration of the audio data to process.

The channel configuration parameter is defined by the geometric position data (i.e. azimuth and elevation angles) associated with the input channels. The geometric data can either be signalled implicitly by CICPspeakerLayoutIdx or CICPspeakerIdx (see ISO/IEC 23001-8) or explicitly.

13.2.4.2.2 QMF analysis of the audio signal

As the first processing step, the binaural renderer transforms L time domain samples of the N_{in}-channel time domain input signal $\tilde{\mathbf{y}}^{\nu} = \left[\tilde{y}_0^{\nu} \ \cdots \tilde{y}_{N_{\text{in}}-1}^{\nu}\right]$ to an N_{in} -channel QMF domain signal representation of dimension L_{n} QMF time slots (slot index n) and K = 64 frequency bands (band index k). In case of the standard framelength of L =4096 samples, L_{n} is equal to 64 (i.e. $L_{\text{n}} = L/64$).

A QMF analysis according to as outlined in ISO/IEC 14496-3:2009, 4.B.18.2 with the modifications stated in ISO/IEC 14496-3:2009, 8.6.4.2 is performed on a frame of the time domain signal $\tilde{\mathbf{y}}^{v}$ to gain a frame of the QMF domain signal $\hat{\mathbf{y}}^{n,k} = [\hat{y}_0^{n,k} \cdots \hat{y}_{\mathrm{N_{in}}-1}^{n,k}]$ with $0 \leq k < K$ and $0 \leq n < L_n$.

If the core decoder already uses QMF domain signals, such that the QMF domain decoded audio data can be obtained, then the QMF analysis step may be omitted.

13.2.4.2.3 Partitioned fast convolution for VOFF processing

A band-wise partitioned fast convolution is carried out to process the QMF domain audio signal and the VOFF coefficients. Therefore, an FFT analysis is performed for each QMF frequency band k with $0 \leq k < \mathrm{K}_{conv}$ with for each channel of the input audio signal.

The audio frames are split into sub-frames when the number of timeslots L_n per frame is larger than $\frac{N_{\mathrm{FFT}}[k]}{2}$. The number of sub-frames is determined according to

$$\mathrm{N_{Frm}[k]} = \max(1, \frac{L_n}{\frac{\mathrm{N_{FFT}[k]}}{2}}).$$

A vector $\mathbf{m}_{\mathrm{conv}}$ is used to indicate which channel of the input signal corresponds to which BRIR pair in the BRIR data set. The audio channels and BRIRs are matched according to their position data. The BRIR position data is taken from the metadata file, the position data of the audio channels is derived from the input parameter signaling the channel configuration.

If there are audio channels with no associated BRIR after the matching of the BRIR positions and the audio channel positions, a fallback BRIR is determined for these channels.

As a first step it is determined whether a BRIR is available with the same elevation and with a maximum azimuth deviation from the desired position of +/- 20°. If such a BRIR is not available, the geometrically closest BRIR is chosen. The geometrically closest BRIR is determined to be the BRIR from the set of available BRIRs that has a minimum geometric distance to the desired position. The geometric distance between two positions P_1, P_2 (each defined by azimuth α and elevation β) therefore is defined as:

$$\Delta(P_1, P_2) = |\beta_1 - \beta_2| + |\alpha_1 - \alpha_2|$$

For the set of known BRIR positions $[P_0, P_{N_{\mathrm{BRIR}}-1}]$ and a wanted position P_W, the geometrically closest BRIR is the entry where $\Delta(P_W, P_i)$ is minimized for $\forall\, i \in [0, N_{\mathrm{BRIR}} - 1]$.

After a BRIR is determined for each audio channel, the partitioned fast convolution is conducted band-wise for all QMF frequency bands k with $0 \leq k < K_{\mathrm{conv}}$ and $K_{\mathrm{conv}} = 32$. This processing step is computed independently of the sampling rate of the audio signal. If the audio sampling rate and the sampling rate of the BRIRs used for the generation of the VOFF coefficients do not correspond, they are nonetheless used for the VOFF processing. If this behaviour is not desired, then the BRIRs should be resampled to the required audio sampling rate and the parameterization recalculated. Figure 80 outlines the partitioned fast convolution process.

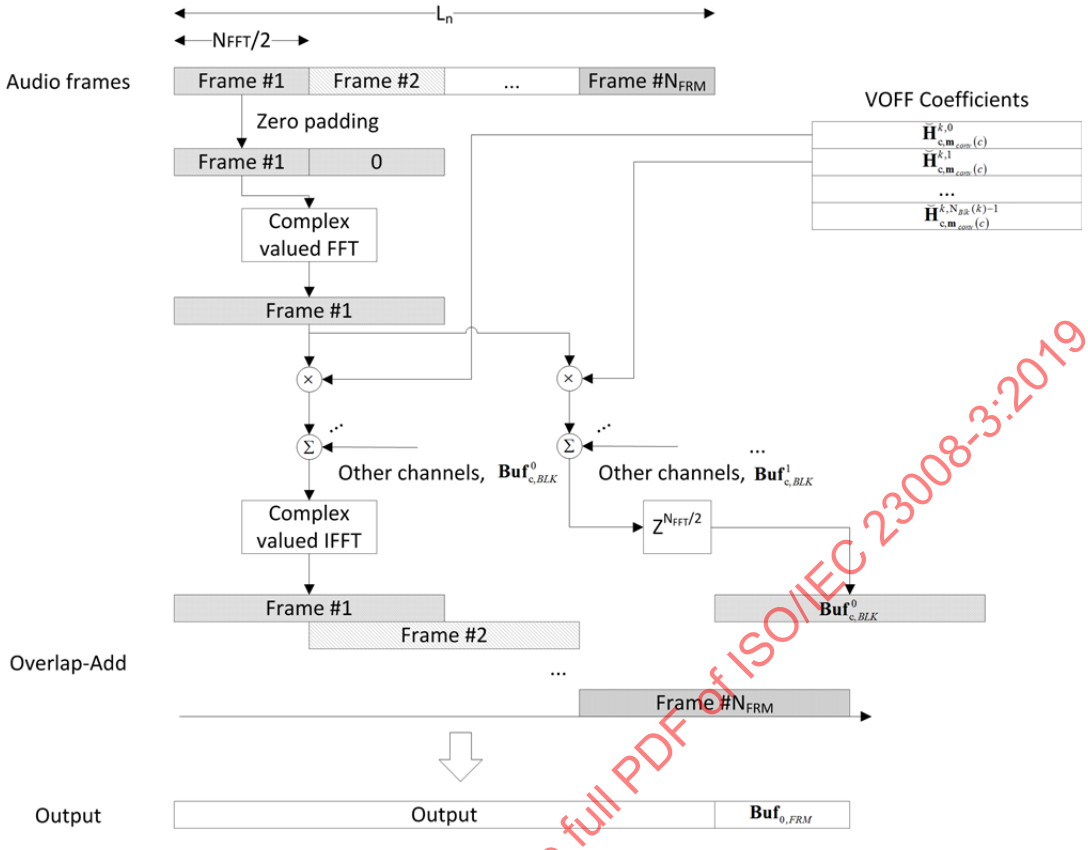


Figure 80 — Outlines of the partitioned fast convolution process

The sub-frames (referred to as "audio frames" in Figure 80) of channel c of the overall audio frame of length L_N are zero-padded.

$$\dot{\mathbf{y}}_{\mathrm{c}}^{i,k} = \left[\hat{y}_{\mathrm{c}}^{\frac{N_{FFT}}{2}i+0,k} \cdots \hat{y}_{\mathrm{c}}^{\frac{N_{FFT}}{2}i+q,k} \cdots \hat{y}_{\mathrm{c}}^{\frac{N_{FFT}}{2}(i+1)-1,k} \quad 0 \cdots 0\right]$$

The zero-padded sub-frames are transformed to the pseudo-FFT domain via complex-valued FFT.

$$\dot{\mathbf{Y}}_{\mathrm{c}}^{i,k} = FFT\{\dot{\mathbf{y}}_{\mathrm{c}}^{i,k}\} = [\hat{Y}_{\mathrm{c}}^{0,i,k} \cdots \hat{Y}_{\mathrm{c}}^{w,i,k} \cdots \hat{Y}_{\mathrm{c}}^{N_{FFT}-1,i,k}]$$

To perform the block-wise fast convolution, the transformed audio frame $\dot{\mathbf{Y}}_{\mathrm{c}}^{i,k}$ is complex multiplied by the VOFF coefficients of channel $\mathbf{m}_{\mathrm{conv}}[c]$ and the complex multiplied outputs of all channels are summed up. From the second blocks, the complex multiplication results are shifted and stored to the buffers:

$$\boldsymbol{Buf}_{0,BLK}^{l} = [Buf_{0,BLK}^{0,l} \cdots Buf_{0,BLK}^{N_{FFT}-1,l}] \text{ and } \boldsymbol{Buf}_{1,BLK}^{l} = [Buf_{1,BLK}^{0,l} \cdots Buf_{1,BLK}^{N_{FFT}-1,l}]$$

For $0 \leq l < N_{BLK}(\mathrm{k}) - 1$:

$$Buf_{0,BLK}^{w,l} = Buf_{0,BLK}^{w,l+1} + \sum_{c=0}^{N_{\mathrm{in}}-1} \dot{Y}_{\mathrm{c}}^{w,i,k} \breve{\mathbf{H}}_{\mathbf{0},\mathbf{m}_{\mathrm{conv}}(c)}^{\boldsymbol{w},\boldsymbol{k},\boldsymbol{l}}$$

$$Buf_{1,BLK}^{w,l} = Buf_{1,BLK}^{w,l+1} + \sum_{c=0}^{N_{\text{in}}-1} \grave{\mathbf{Y}}_{\text{c}}^{w,i,k} \breve{\mathbf{H}}_{\mathbf{1},\mathbf{m}_{\text{conv}}(c)}^{\boldsymbol{w},\boldsymbol{k},\boldsymbol{l}}$$

Then, the fast convoluted pseudo-FFT domain audio frame $[\grave{\mathbf{Z}}_{0,\text{conv}}^{w,i,k}\ \grave{\mathbf{Z}}_{1,\text{conv}}^{w,i,k}]$ is defined as:

$$\grave{\text{Z}}_{0,\text{conv}}^{w,i,k} = Buf_{0,BLK}^{w,0}$$

$$\grave{\text{Z}}_{1,\text{conv}}^{w,i,k} = Buf_{1,BLK}^{w,0}$$

The fast convoluted pseudo-FFT domain audio frame $[\grave{\mathbf{Z}}_{0,\text{conv}}^{w,i,k}\ \grave{\mathbf{Z}}_{1,\text{conv}}^{w,i,k}]$ is then inverse transformed to the QMF domain and forms the fast convoluted QMF domain audio frame $[\ \grave{\mathbf{z}}_{0,\text{conv}}^{i,k}\ \grave{\mathbf{z}}_{1,\text{conv}}^{i,k}]$.

$$\grave{\mathbf{z}}_{0,\text{conv}}^{i,k} = IFFT\left\{\left[\grave{\text{Z}}_{0,\text{conv}}^{0,i,k} \cdots \grave{\text{Z}}_{0,\text{conv}}^{N_{FFT}-1,i,k}\right]\right\} = [\grave{\text{z}}_{0,\text{conv}}^{0,i,k} \cdots \grave{\text{z}}_{0,\text{conv}}^{N_{FFT}-1,i,k}]$$

$$\grave{\mathbf{z}}_{1,\text{conv}}^{i,k} = IFFT\{[\grave{\text{Z}}_{1,\text{conv}}^{0,i,k} \cdots \grave{\text{Z}}_{1,\text{conv}}^{N_{FFT}-1,i,k}]\} = [\grave{\text{z}}_{1,\text{conv}}^{0,i,k} \cdots \grave{\text{z}}_{1,\text{conv}}^{N_{FFT}-1,i,k}]$$

Finally, the fast convoluted QMF domain audio frame $[\ \grave{\mathbf{z}}_{0,\text{conv}}^{i,k}\ \grave{\mathbf{z}}_{1,\text{conv}}^{i,k}]$ is overlapped and saved.

For $0 \le i < N_{FRM}(\text{k}) - 1$ do:

$$\breve{z}_{0,\text{conv}}^{\frac{N_{FFT}}{2}i+q,k} = \grave{\text{z}}_{0,\text{conv}}^{q,i,k}$$

$$\breve{z}_{1,\text{conv}}^{\frac{N_{FFT}}{2}i+q,k} = \grave{\text{z}}_{1,\text{conv}}^{q,i,k}$$

For $i = N_{FRM}(\text{k}) - 1$ do:

For $0 \le q < N_{FFT}(\text{k})/2 - 1$ do:

$$\breve{z}_{0,\text{conv}}^{\frac{N_{FFT}}{2}i+q,k} = \grave{\text{z}}_{0,\text{conv}}^{q,i,k}$$

$$\breve{z}_{1,\text{conv}}^{\frac{N_{FFT}}{2}i+q,k} = \grave{\text{z}}_{1,\text{conv}}^{q,i,k}$$

For $N_{FFT}(\text{k})/2 - 1 \le q < N_{FFT}(\text{k}) - 1$ do:

$$Buf_{0,FRM}^{q} = \grave{\text{z}}_{0,\text{conv}}^{q,i,k}$$

$$Buf_{1,BLK}^{q} = \grave{\text{z}}_{1,\text{conv}}^{q,i,k}$$ ”

The convolution results from each audio input channel with each BRIR pair are summed up in each QMF frequency band k with $0 \le k < K_{\text{conv}}$ resulting in an intermediate 2-channel K_{conv}-band pseudo-FFT domain signal.

$\breve{\mathbf{z}}_{\text{conv}}^{n,k} = [\breve{z}_{0,\text{conv}}^{n,k}, \breve{z}_{1,\text{conv}}^{n,k}]$ in QMF domain frequency band k.

Next, a bandwise FFT synthesis is carried out to transform the convolution result back to the QMF domain resulting in an intermediate 2-channel K_{conv}-band QMF domain signal with $\max(N_{\text{FFT}}[k])$ time slots.

$$\tilde{\mathbf{z}}_{conv}^{n,k} = \left[\tilde{z}_{0,conv}^{n,k}, \tilde{z}_{1,conv}^{n,k}\right] \text{ with } 0 \le n < \text{L}_N \text{ and } 0 \le k < K_{\text{conv}}.$$

For each QMF domain input signal frame with $L_n = 64$ timeslots a convolution result signal frame with $L_n = 64$ timeslots is returned. The remaining timeslots are stored ($[Buf_{0,FRM}, Buf_{1,FRM}]$) and an overlap-add processing is carried out in the following frame(s).

13.2.4.2.4 Generation of late reverberation

As a second intermediate signal a QMF domain reverberation signal called $\hat{\mathbf{z}}_{\text{rev}}^{n,k} = \left[\hat{z}_{0,\text{rev}}^{n,k}, \hat{z}_{1,\text{rev}}^{n,k}\right]$ is generated up to band $K_{\text{conv}} - 1$ by a reverberator module.

The reverberator takes as input:

1) a QMF domain stereo downmix of one frame of the input signal;
2) the frequency-dependent RT60 reverberation time and the late reverberation energy values from the BRIR metadata information;

and returns a 2-channel QMF domain late reverberation tail.

The RT60 reverberation time and late reverberation energy values are kept as read from the metadata file independent on the audio sampling rate; they are just remapped to different centre frequencies if the audio sampling rate differs from the BRIR sampling rate that was used during parameterization. Therefore, the center-frequencies of the late reverberation analysis bands from the BRIR metadata information are recalculated in a first step, if the audio sampling rate differs from the BRIR sampling rate that was used during parameterization.

$$f_c[k] = \frac{f_c[k]}{f_{s,\text{BRIR}}} \cdot f_s \text{ with } 0 \le k < K_{ana}.$$

Next, the number of required analysis bands K_{LR} is determined. This is determined from:

$$f_c[k] \le f_{c,max} \text{ with } f_{c,max} = 48.5 \cdot \left(\frac{f_s}{2} / 64\right).$$

Then, a QMF domain stereo downmix of one frame of the input signal $\hat{\mathbf{y}}^{n,k}$ is carried out to form the input of the reverberator by weighted summation of the input signal channels. The weighting gains are contained in the downmix matrix M_{DMX}. These gains are real-valued and non-negative and the downmix matrix is of dimension $N_{\text{out}} \times N_{\text{in}}$. It contains a non-zero value where a channel of the input signal is mapped to one of the two output channels.

The channels that represent loudspeaker positions on the left hemisphere are mapped to the left output channel and the channels that represent loudspeakers located on the right hemisphere are mapped to the right output channel. The signals of these channels are weighted by a coefficient of 1. The channels that represent loudspeakers in the median plane are mapped to both output channels of the binaural signal. The input signals of these channels are weighted by a coefficient

$$a = 0.7071 \approx \frac{1}{\sqrt{2}}$$

In addition, an energy equalization step is performed in the downmix. It adapts the bandwise energy of one downmix channel to be equal to the sum of the bandwise energy of the input signal channels that are contained in this downmix channel. This energy equalization is conducted by a bandwise multiplication with a real-valued coefficient

$$c_{\mathrm{eq},k} = \sqrt{\frac{\mathrm{P}_{\mathrm{in}}^{k}}{\mathrm{P}_{\mathrm{out}}^{k} + \varepsilon}}$$

The factor $c_{\mathrm{eq,k}}$ is limited to an interval of [0.5, 2].The numerical constant ε is introduced to avoid a division by zero. The downmix is also bandlimited to the band K_{conv}; the values in all higher frequency bands are set to zero.

The downmix and the K_{LR} values of RT60 and energy are fed to the reverberator. In the reverberator a mono downmix of the stereo input is calculated. This is performed by incoherently applying a 90° phase shift on one of the stereo input channels. This mono signal is then fed to a feedback delay loop in each frequency band k, which creates a decaying sequence of impulses. It is followed by parallel FIR decorrelators that distribute the wanted signal energy in a decaying manner into the intervals between the impulses and create incoherence between the output channels. A decaying filter tap density is applied to create the energy decay defined by the RT60 time. The filter tap phase operations are restricted to four options to implement a sparse and multiplier-free decorrelator. After the calculation of the reverberation an inter-channel coherence (ICC) correction step is included in the reverberator module for every QMF frequency band. In the ICC correction step frequency-dependent direct gains and crossmix gains are used to adapt the ICC. The processing in the reverberator is shown in detail in Figure 81.

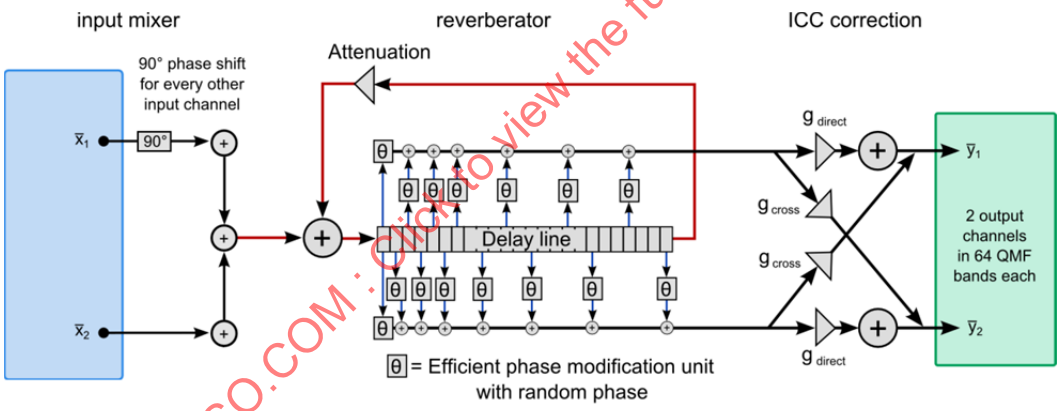


Figure 81 — Processing in the reverberator

The amount of energy and the reverberation times for the different frequency bands are defined by the input parameter set. The RT60 reverberation time and energy values are given at a number of K_{LR} frequency points which are internally mapped to the $K = 64$ QMF frequency bands.

The reverberation is adaptively scaled according to a correlation measure of the input signal frame. The scaling factor is defined as a value in the interval of $\left[\sqrt{N_{\mathrm{DMX,act}}}, N_{\mathrm{DMX,act}}\right]$ linearly depending on a correlation coefficient c_{corr} between 0 and 1. The coefficient c_{corr} is set to the value 0.5.

$N_{\mathrm{DMX,act}}$ is the number of channels that are active (carry signal energy) in the current audio frame and are downmixed to one output channel (e.g.the intersection of all values of one line of $\mathrm{M}_{\mathrm{DMX}}$ and the currently active audio channels).

The scaling factors are:

$$\mathbf{c}_{\text{scale}} = \left[c_{\text{scale},0}, c_{\text{scale},1}\right] = \begin{bmatrix} \sqrt{N_{\text{DMX,act},0}} + c_{\text{corr}} \cdot \left(N_{\text{DMX,act},0} - \sqrt{N_{\text{DMX,act},0}}\right) \\ \sqrt{N_{\text{DMX,act},1}} + c_{\text{corr}} \cdot \left(N_{\text{DMX,act},1} - \sqrt{N_{\text{DMX,act},1}}\right) \end{bmatrix}$$

The number of active downmix channels $N_{\text{DMX,act}}$ may change over time.

The scaling factors are smoothed over audio signal frames by a 1st order low pass filter resulting in smoothed scaling factors $\tilde{\mathbf{c}}_{\text{scale}} = \left[\tilde{c}_{\text{scale},0}, \tilde{c}_{\text{scale},1}\right]$. The scaling factors are initialized in the first audio input data frame.

13.2.4.2.5 QTDL processing

In the bands K_{conv} to K_{max} –1, a QMF domain tapped delay line (QTDL) replaces both the VOFF processing and the sparse Frequency Domain Reverberator.

The QTDL processing is achieved by performing one complex multiplication and delay.

$$\check{z}^{n,k}_{\text{i,QTDL}} = \sum_{m=0}^{N_{\text{in}}} g^{k}_{i,\mathbf{m}_{\text{conv}}[m]}\, \hat{y}^{\nu - d^{k}_{i,\mathbf{m}_{\text{conv}}[m]},k}_{\text{m}} \quad i \in \{0,1\}$$

$$\check{\mathbf{z}}^{n,k}_{\text{QTDL}} = \left[\check{z}^{n,k}_{0,\text{QTDL}}, \check{z}^{n,k}_{1,\text{QTDL}}\right]$$

For each QMF domain input signal with $L_n = 64$ timeslots, 64 QTDL processed output signals $\check{\mathbf{z}}^{n,k}_{\text{QTDL}}$ are returned corresponding to $L_n = 64$ timeslots. The remaining timeslots (corresponding to the last $d^{k}_{i,\mathbf{m}_{\text{conv}}[m]}$ samples) are stored and an overlap-add processing is carried out in the following frame(s).

13.2.4.2.6 Mixing and combination of binaural processed outputs

Next, the VOFF processed output $\tilde{\mathbf{z}}^{n,k}_{conv} = \left[\tilde{z}^{n,k}_{0,conv}, \tilde{z}^{n,k}_{1,conv}\right]$, the reverberator output $\hat{\mathbf{z}}^{n,k}_{\text{rev}} = \left[\hat{z}^{n,k}_{0,\text{rev}}, \hat{z}^{n,k}_{1,\text{rev}}\right]$ and the QTDL processed output $\check{\mathbf{z}}^{n,k}_{\text{QTDL}} = \left[\check{z}^{n,k}_{0,\text{QTDL}}, \check{z}^{n,k}_{1,\text{QTDL}}\right]$ for one QMF domain audio input frame are combined by a mixing process that adds the signals on a band-by-band basis.

The late reverberation output is delayed by d time slots as part of the mixing process. The delay d takes into account the frequency-dependent transition time from early reflections to late reflections in the BRIRs and an initial delay of the reverberator of 20 QMF time slots, as well as an analysis delay of 0.5 QMF time slots for the QMF analysis of the BRIRs to ensure the insertion of the late reverberation at an appropriate time slot.

The combined signal $\hat{\mathbf{z}}^{n,k}$ at one time slot n is calculated by $\hat{\mathbf{z}}^{n,k}_{\text{conv}} + \hat{\mathbf{z}}^{n-d,k}_{\text{rev}}$ in the bands up to K_{conv} and is directly connected by $\check{\mathbf{z}}^{n,k}_{\text{QTDL}}$ in the bands between K_{conv} and K_{max}. For clarity, it should be noted that the bands beyond K_{max} are not processed in the binaural rendering and thus no signal is combined to the output signal in this band.

If *flag_HRIR* == 0

$$\hat{\mathbf{z}}^{n,k} = \begin{cases} \hat{\mathbf{z}}_{\text{conv}}^{n,k} + \hat{\mathbf{z}}_{\text{rev}}^{n-d,k} & 0 \le k < K_{\text{conv}} \\ \check{\mathbf{z}}_{\text{QTDL}}^{n,k} & K_{\text{conv}} \le k < K_{\text{max}} \\ 0 & K_{\text{max}} \le k < 64 \end{cases}$$

else

$$\hat{\mathbf{z}}^{n,k} = \begin{cases} \hat{\mathbf{z}}_{\text{conv}}^{n,k} & 0 \le k < K_{\text{conv}} \\ \check{\mathbf{z}}_{\text{QTDL}}^{n,k} & K_{\text{conv}} \le k < K_{\text{max}} \\ 0 & K_{\text{max}} \le k < 64 \end{cases}$$

13.2.4.2.7 QMF synthesis of binaural QMF domain signal

One 2-channel frame of $L_n = 32$ time slots of the QMF domain output signal $\hat{\mathbf{z}}^{n,k}$ is transformed to a 2-channel time domain signal frame with length L by the QMF synthesis according to ISO/IEC 14496-3:2009, 4.6.18.4.2 yielding the final time domain output signal $\tilde{\mathbf{z}}^{\nu} = [\tilde{z}_0^{\nu}, \tilde{z}_1^{\nu}]$.

13.3 Time-domain binaural renderer

13.3.1 General

The time-domain binaural renderer may be used for generating 3D audio headphone signals for all types of input content (channel and/or object and/or HOA).

In the general case (channel and/or object and/or HOA input), the time-domain binaural renderer operates using the virtual loudspeaker binaural (VLB) approach, which converts a loudspeaker signal $\mathbf{W}_{\text{SPEAKER}}$ to the stereo output signal $\mathbf{W}_{\text{LR}}$ using time-domain virtual loudspeaker binaural parameters for each loudspeaker position.

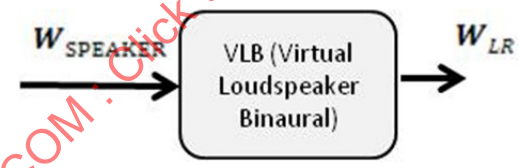


Figure 82 — VLB approach for all types of input content (channel and/or object and/or HOA)

However, in the specific case of HOA-only signals, the HOA to binaural (H2B) approach may also be used, which directly converts the HOA signal to the stereo output signal using HOA-based binaural parameters.

The H2B approach may be used for HOA-only signals by applying a binaural filtering directly to the HOA components **C** to obtain the binaural signals $\mathbf{W}_{\text{LR}}$

Figure 83 — H2B approach for HOA-only signals

When using time-domain binauralization for HOA-only input signals, both VLB and H2B approaches shall be supported in the decoder. In this case, the approach used in the decoder for a given audio bitstream and given filters is selected depending on the availability of the filter format at the decoder (VLB, H2B, or both). H2B filters can be obtained from VLB filters, but not conversely.

In the case where H2B filters are available, the H2B methodology is to be used, otherwise VLB methodology shall be used.

Whatever the approach selected by the decoder (VLB or H2B), the binauralization is based on the filtering (with summation for each ear) of the input signals. In the VLB case, the inputs to the filters are the virtual loudspeakers signals, and in the H2B case, the inputs are the HOA components.

For each ear, this filtering process consists of:

— applying a specific direct filter block, ideally modelling the direct path and early reflections (in effect also modelling the head and torso of the listener), to each input signal;

— applying a common diffuse filter B_{mean} (possibly decomposed into M blocks), ideally modelling the diffuse part of the room response (including late reflections and reverb), to a weighted sum (taking into account the difference of energy between BRIRs) of the input signals;

— limiting the bandwidth of each block (to save computational complexity) according to the cut-off frequency provided (by the filter design, but also possibly by the decoder, e.g. limitation of bandwidth of the content).

The direct and diffuse filter coefficients, diffuse weights, and cut-off frequencies are provided by an adaptive filter parameterization technique.

Further information about delay and complexity of time domain binauralization can be found in Annex H.

13.3.2 Definitions

13.3.2.1 General variables

In the following subclauses, the following variable conventions are used.

n	time index
k	ear index (0=left, 1=right)
u	frequency index
L	number of input channels (virtual loudspeakers or HOA components). It is equal to the number of BRIRs (each BRIR containing a pair of impulse responses).
$l \in [1; L]$	channel index
$I(l)$	input signal for channel l
O^k	output signal for the left ($k = 0$) or right ($k = 1$) ear
$h_{k,l}$	impulse response of the left ($k = 0$) or right ($k = 1$) BRIR l

$h_{k,l}^{[n_0;n_1]}$	tapsn_0 to n_1 of impulse response of the left ($k = 0$) or right ($k = 1$) BRIR l

13.3.2.2 Filter parameterization variables

$bsDelay$	initial delay of a BRIR set in time domain samples (propagation time)
N_d	size in samples of the direct part of BRIRs
$A^k(l) \in \mathbb{R}^{N_d}$	FIR coefficients for the direct block of the left ($k = 0$) or right ($k = 1$) BRIR l
M	number of diffuse blocks
$m \in [1; M]$	block index
$\boldsymbol{B}_l^k \in \mathbb{R}^{N_d \cdot M}$	FIR coefficients for the left ($k = 0$) or right ($k = 1$) diffuse filter of BRIR l
$\boldsymbol{B}_{mean}^k \in \mathbb{R}^{N_d \cdot M}$	FIR coefficients for the left ($k = 0$) or right ($k = 1$) average diffuse filter
$B_{mean}^k(m) \in \mathbb{R}^{N_d}$	FIR coefficients for diffuse block m of the left ($k = 0$) or right ($k = 1$) average diffuse filter
${f_a}^k(l)$	cut off frequency for direct part of the left ($k = 0$) or right ($k = 1$) BRIR l
${f_b}^k(m)$	cut off frequency for the m^{th} diffuse block of the left ($k = 0$) or right ($k = 1$) average diffuse filter
$Dw(l)$	gain to apply to the input channel l before filtering with average diffuse filters
$z^{-N_d.m}$	delay line operator for a delay of m frequency blocks (i.e., $N_d m$ time samples)
$*_{[0\,;\,\ldots\,;\,f]}$	frequency-limited convolution operator (i.e., term-by-term multiplication in the frequency domain applied to a limited number of frequency bins, from 0 to f)

13.3.2.3 Function definitions

The energy of a time-domain single-channel signal $x(n)$ is defined by:

$$E(x) = \sum_{n=0}^{+\infty} x^2(n)$$

The cumulative energy is defined by:

$$e(x,n) = \sum_{i=0}^{n} x^2(i)$$

The normalized cumulative energy is defined by:

$$e_{norm}(x,n) = \frac{e(x,n)}{E(x)}$$

The frequency-domain cumulative energy is defined by:

$$e_{freq}(x,u) = \frac{\sum_{i=0}^{u} X^2(i)}{\sum_{i=0}^{Nyquist} X^2(i)}$$

where $X(u)$ is the discrete Fourier transform of the N-point signal $x(n)$:

$$X(u) = \sum_{n=0}^{N-1} x(n) e^{-i2\pi \frac{u}{N} n}$$

and $Nyquist$ is the index of the Nyquist frequency, i.e., $\frac{N}{2}$ when N is even.

13.3.3 Parameterization of binaural room impulse responses

13.3.3.1 General

For each ear k and channel l, the BRIR impulse response $h_{k,l}$ is parameterized as:

— an initial propagation delay $bsDelay$ (common for all BRIRs). This is shown in Figure 84 as the initial 'quiet samples';

— a direct filter block $A^k(l) \in \mathbb{R}^{N_d}$ (N_d samples in the time domain). This is shown in Figure 84 as the block following bsDelay;

— M diffuse filter blocks $B^k_{mean}(m) \in \mathbb{R}^{N_d}$, for $m \in [1; M]$ (common for all BRIRs). This is shown in Figure 84 as the blocks following the Direct block;

— a diffuse gain Dw(l) (common for left and right impulse responses).

The parameterization is composed of four stages that are described in details in the next subclauses:

1) propagation time calculation;
2) direct/diffuse automatic segmentation;
3) diffuse filters computation;
4) cut-off frequencies computation.

As a special case, if BRIRs are shorter than 1 024 samples in the time domain, then the three first stages of parameterization are ignored and :

— $bsDelay$ is set to 0;

— N_d is set to the power of two above or equal to the BRIRs size;

— $A^k(l) = h_{k,l}$ (plus additional zeros at the end to complete to N_d samples);

— M is set to 0 (i.e., no diffuse filter).

13.3.3.2 Propagation time calculation

Starting from a set of original BRIRs, the offset $bsDelay$ (in sample) is first computed to remove propagation time and samples with near-zero energy at the beginning of all BRIRs. For each ear k and each channel l, the integer time index $d_{k,l}$ at which BRIR cumulative energy reaches $thres1 = 10^{-5}\, E_{max}$ is computed:

$$d_{k,l} = \min\left[\arg_n\left(\frac{e(h_{k,l}, n)}{E_{max}} > thres1\right)\right]$$

where E_{max} is the energy of the BRIR impulse response with maximum energy (among all filters):

$$E_{max} = \max_{k,l}\left[\mathrm{E}(h_{k,l})\right]$$

The minimum of all time indexes is then selected as *bsDelay*:

$$bsDelay = \max_{k,l}\left[d_{k,l}\right]$$

All samples before *bsDelay* are set to zero in the BRIRs.

13.3.3.3 Direct/diffuse automatic segmentation

Each BRIR is then automatically decomposed into a first block (direct and first reflections part) and M diffuse blocks of the same size. This segmentation is illustrated on Figure 84.

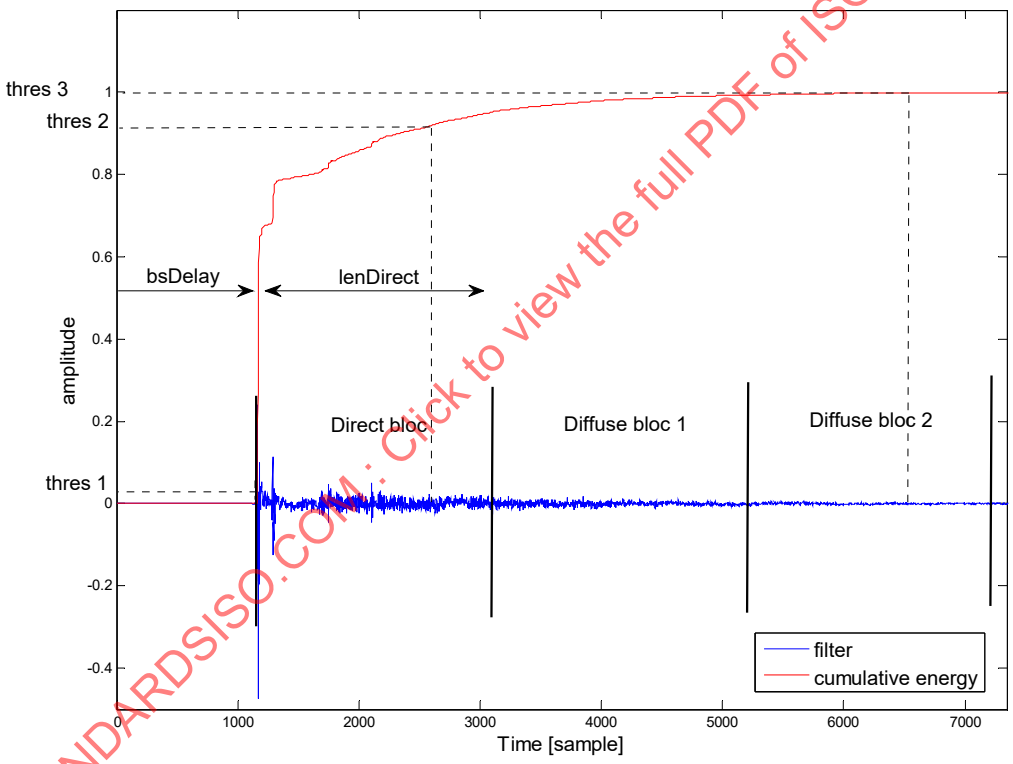


Figure 84 — Automatic direct/diffuse segmentation for one BRIR

The direct block length is computed as follows. For each ear k and each channel l, the integer time index $d_{k,l}$ at which the BRIR normalized cumulative energy reaches $thres2 = 1 - 10^{-15/10}$ is computed:

$$d_{k,l} = \min\left[\arg_n\left(e_{norm}(h_{k,l}, n) > thres2\right)\right]$$

The maximum time index is selected and it is increased to the next power of two to get the direct block length:

$N_d = \text{nextpow2}(\max_{k,l}[d_{k,l}])$

Finally, the direct block length shall be limited to be not more that 8192 samples long:

$N_d = min(8192, N_d)$

For each ear k and channel l, the direct filer $A^k(l)$ is then given by:

$A^k(l) = h_{k,l}^{[bsDelay;bsDelay+N_d-1]}$

The total number of diffuse blocks M is computed as follows. For each ear k and each channel l, the integer time index $d_{k,l}$ at which BRIR normalized cumulative energy reaches *thres3*=1-10$^{-19/10}$ is computed:

$$d_{k,l} = \min\left[\arg_n\left(\frac{e_{norm}(h_{k,l},n)}{E_{max}} > thres3\right)\right]$$

Then M is given by :

$$M = \text{ceil}\left(\frac{\max_{k,l}[d_{k,l}] - (bsDelay + N_d - 1)}{N_d}\right)$$

where ceil is the round up function. If M is zero, then there is no diffuse filter and the last steps of parameterization shall be skipped. This is the case when *thres3* is reached before the end of the direct block (i.e., time index $bsDelay + N_d - 1$).

Otherwise the M-block long diffuse filter is computed for ear k and channel l by:

$\boldsymbol{B}_l^k = h_{k,l}^{[bsDelay+N_d;bsDelay+(M+1).N_d-1]}$

A smoothing window $w(n)$ is applied to fade out the last $N_{fade} = 512$ points of $\boldsymbol{B}_l^k$. Specifically we compute $w(n)$ by:

$w(n) = \frac{cos(\pi.n/N_{fade})+1}{2}$ for $n = [0; N-1]$

BRIR samples beyond $\boldsymbol{B}_l^k$ (i.e., time indexes superior to $bsDelay + (M+1).N_d - 1$) are to be ignored. If the end of $\boldsymbol{B}_l^k$ goes beyond the end of the BRIRs, then $\boldsymbol{B}_l^k$ is completed with zeros at the end (zero-padding) to reach $(M.N_d)$ samples. This is performed before applying the smoothing window.

13.3.3.4 Diffuse filters computation

To reduce computational cost, a single average diffuse filter $\boldsymbol{B}_{mean}^k$ is computed for ear k by taking the mean of all normalized contributions per channel:

$$\boldsymbol{B}_{mean}^k = \frac{1}{L}\sum_{l=1}^{L}\frac{\boldsymbol{B}_l^k}{E(\boldsymbol{B}_l^k)}$$

where $\boldsymbol{B}_l^k$ is the M-block long diffuse filter for ear k and channel l, and $E(\boldsymbol{B}_l^k)$ its energy. $\boldsymbol{B}_{mean}^k$ is then cut into M contiguous blocks $B_{mean}^k(m)$ of N_d samples each for $m \in [1; M]$.

A compensation gain $Dw(l)$ is computed per channel l by:

$$Dw(l) = \frac{2.G}{W^0(l) + W^1(l)}$$

where G is a global attenuation gain of 6dB ($G = 10^{-6/20}$) and $W^k(l)$ is a weight computed for ear k and channel l by:

$$W^k(l) = \frac{\sqrt{E(\boldsymbol{B}^k_{mean})}}{\sqrt{E(\boldsymbol{B}^k_l)}}$$

13.3.3.5 Cut-off frequencies computation

In typical BRIRs, high-frequency energy decays faster than low-frequency energy. Cut-off frequencies are extracted for direct and diffuse block filters, to avoid multiplying frequency bins with low energy during processing.

Figure 85 shows an example of the decomposition of one BRIR (left and right) into 1 block of direct part, and 2 blocks of diffuse part, on a time frequency representation. Each block is N_d = 2048 samples long. The samples after $3N_d$ are not used. The frequencies above ${f_a}^l(l)$ and ${f_a}^r(l)$ in the direct block are not used. The frequencies above ${f_b}^l(1)$ and ${f_b}^r(1)$ in the first diffuse block are not used. The frequencies above ${f_b}^l(2)$ and ${f_b}^r(2)$ in the second diffuse block are not used. During processing, unprocessed bins above the corresponding cut-off frequencies are equivalent to a multiplication by 0 (but not from a complexity point of view) which can lead to some artifacts due to circular convolution inside a block. These artifacts can generally be kept low enough in energy to avoid audible artifacts after the whole process.

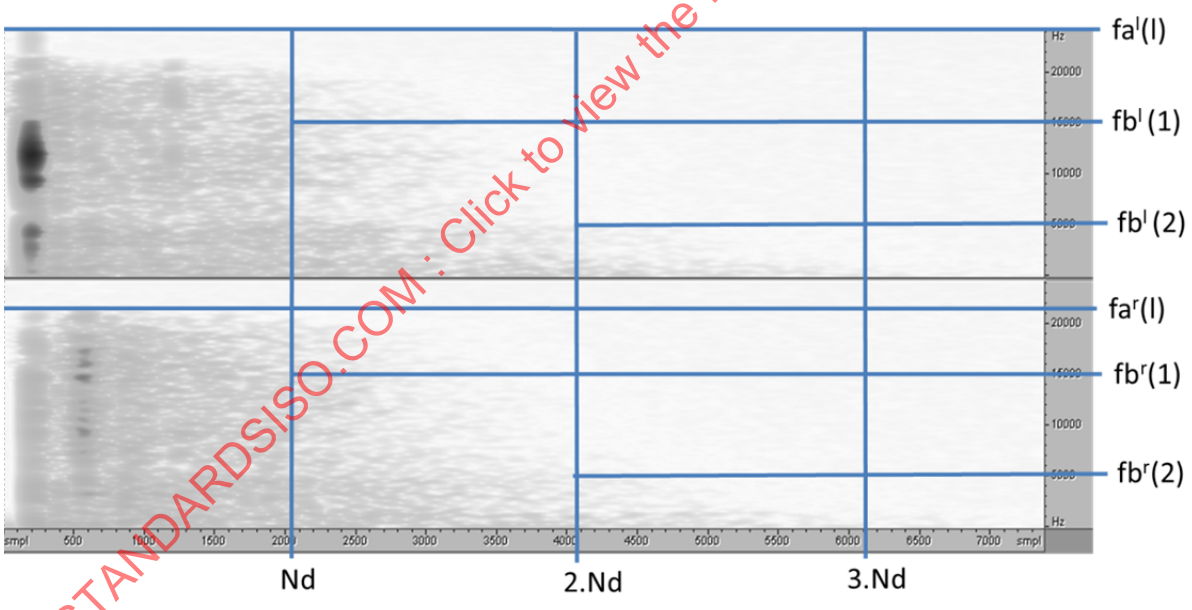


Figure 85 — Time frequency representation of a pair of BRIR

This phenomenon is exploited in the binaural renderer. For ear k and channel l, the frequency index $\hat{d}_{k,l}$ at which the frequency-domain cumulative energy of the direct block reaches *thres4*=0.99 is computed:

$$\hat{d}_{k,l} = \min\left[\arg_u\left(e_{freq}\left(A^k(l), u\right) > thres4\right)\right]$$

Then the normalized cutoff frequency ${f_a}^k(l)$ is defined by:

$$ {f_a}^k(l) = \frac{\hat{d}_{k,l}}{Nyquist}$$

In a final "quantization" stage, ${f_a}^k(l)$ is set to its closest value in vector [1/6 1/3 1/2 2/3 5/6 1].

For each channel l, each ear k and each diffuse bloc m, the frequency index $\hat{d}_{l,k,m}$ at which the frequency-domain cumulative energy of the diffuse block m reaches *thres4* is computed:

$\hat{d}_{l,k,m} = \min\left[\arg_u\left(e_{freq}\left(B_l^k(m), u\right) > thres4\right)\right]$

Then the normalized cutoff frequency $f_b{}^k(m)$ is defined by:

$$f_b{}^k(m) = \frac{\max_l \hat{d}_{l,k,m}}{Nyquist}$$

In a final "quantization" stage, $f_b{}^k(m)$ is set to its closest value in vector [1/6 1/3 1/2 2/3 5/6 1].

13.3.4 Time-domain binaural processing

13.3.4.1 General

The left/right output is given by the following formula:

$$O^k = \sum_{l=1}^{L}\left(A^k(l) *_{\left[0;\ldots;\, f_a{}^k(l)\right]} I(l)\right) + \sum_{m=1}^{M}\left(B^k_{mean}(m) *_{\left[0;\ldots;\, f_b{}^k(m)\right]}\left(z^{-N_d.m}.\sum_{l=1}^{L} D_w(l).I(l)\right)\right)$$

The processing is achieved in the DFT (discrete Fourier transform) domain based on a block-wise approach. The following constraints are associated to the algorithm.

— The number of samples N_d of the direct part shall be equal to a power of 2.

— The size of each of the M diffuse block is equal to N_d.

— The frame size of the binauralization process is equal to N_d.

— The size of the delay line is a multiple of Nd (delay of 1 frequency block).

— The size of the DFT is 2. N_d

Figure 86 shows the implementation with L input signals and M diffuse blocks:

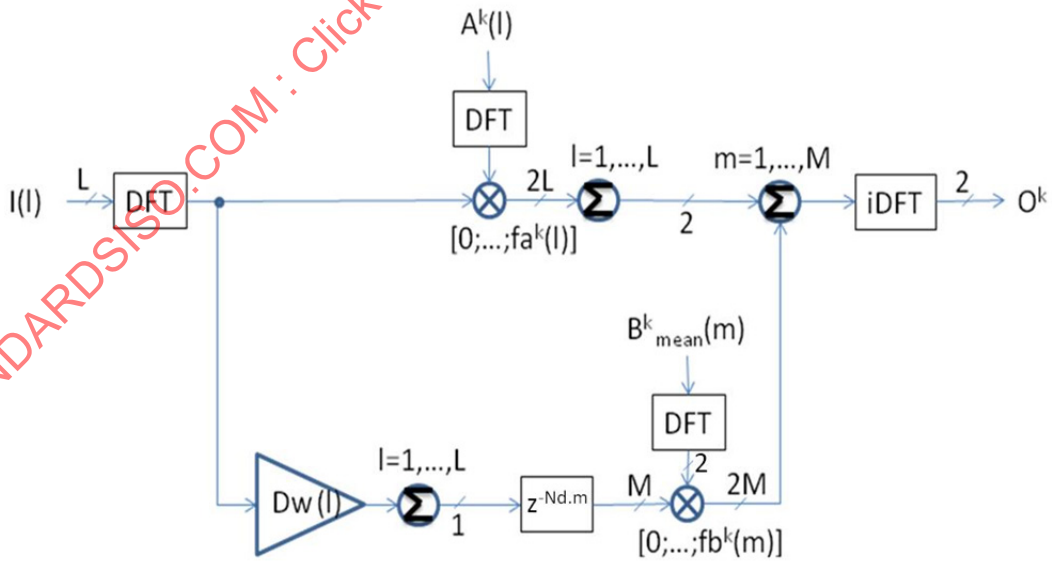


Figure 86 — Description of the binauralization algorithm

13.3.4.2 Initialization

The L filters of the direct block $A^k(l)$ and the M filters of the diffuse blocks $B^k_{mean}(m)$ are provided as time domain blocks of size N_d. Each time domain filter is converted to the frequency domain in an

initialization step. $2L$ buffers of $2N_d$ complex data for Direct part, and $2M$ buffers of $2N_d$ complex data for diffuse blocks are provided.

13.3.4.3 Processing

The L input signals (channel or HOA) are temporarily stored in buffers of size N_d. Using an overlap-add/overlap-save technique, buffers of size $2N_d$ are created and then converted into the DFT domain.

For the direct block and for each ear, these L frequency buffers are multiplied term-by-term with the associated L direct filters from 0 up to the frequency ${f_a}^k(l)$ and accumulated in a frequency output buffer previously initialized to zero.

These L frequency buffers are also multiply-accumulated by the diffuse weights $Dw(l)$ for each ear to provide a weighted downmix buffer dedicated to feed each of the diffuse blocks.

For each of the M diffuse blocks and for each ear, the appropriately delayed weighted downmix buffer is multiplied term-by-term with the associated diffuse filters from 0 up to the frequency ${f_b}^k(l)$ and accumulated in the frequency output buffer.

14 MPEG-H 3D audio stream (MHAS)

14.1 Overview

This clause defines a self-contained stream format to transport MPEG-H 3D audio data. The transport mechanism uses a packetized approach. Both, configuration data as well as coded audio payload data is embedded into separate packets. Synchronization and length information is added to enable a self-synchronizing syntax.

This stream format is intended to be used for the transmission over channels where no frame synchronization is available and it may be used for the transmission over channels with fixed frame synchronization.

14.2 Syntax

14.2.1 Main MHAS syntax elements

Table 218 — Syntax of mpeghAudioStream()

Syntax	No. of bits	Mnemonic

mpeghAudioStream()
{
  while (bitsAvailable() != 0) {
  mpeghAudioStreamPacket();
  }
}		

Table 219 — Syntax of mpeghAudioStreamPacket()

Syntax	No. of bits	Mnemonic
mpeghAudioStreamPacket()		
{		
MHASPacketType = escapedValue(3,8,8);	**3,11,19**	**uimsbf**
MHASPacketLabel = escapedValue(2,8,32);	**2,10,42**	**uimsbf**
MHASPacketLength = escapedValue(11,24,24);	**11,35,59**	**uimsbf**
MHASPacketPayload(MHASPacketType);		
}		
NOTE With the given bit allocation, MHASPacketPayload() is always byte-aligned.		

Table 220 — Syntax of MHASPacketPayload()

Syntax	No. of bits	Mnemonic
MHASPacketPayload(MHASPacketType)		
{		
switch (MHASPacketType) {		
case PACTYP_SYNC:		
0xA5; /* syncword*/	**8**	**uimsbf**
break;		
case PACTYP_MPEGH3DACFG:		
mpegh3daConfig();		
break;		
case PACTYP_MPEGH3DAFRAME:		
mpegh3daFrame();		
break;		
case PACTYP_AUDIOSCENEINFO:		
mae_AudioSceneInfo();		
break;		
case PACTYP_FILLDATA:		
for (i=0; i< MHASPacketLength; i++) {		
mhas_fill_data_byte(i);	**8**	**bslbf**
}		
break;		
case PACTYP_SYNCGAP:		
syncSpacingLength = escapedValue(16,24,24);	**16,40,64**	**uimsbf**
break;		
case PACTYP_MARKER:		
for (i=0; i< MHASPacketLength; i++) {		
marker_byte(i);	**8**	**bslbf**
}		
break;		
case PACTYP_CRC16:		
mhasParity16Data;	**16**	**bslbf**
break;		
case PACTYP_CRC32:		
mhasParity32Data;	**32**	**bslbf**
break;		
case PACTYP_GLOBAL_CRC16:		
global_CRC_type;	**2**	**bslbf**
numProtectedPackets;	**6**	**bslbf**
mhasParity16Data;	**16**	**bslbf**
break;		
case PACTYP_ GLOBAL_CRC32:		
global_CRC_type;	**2**	**bslbf**

Syntax	No. of bits	Mnemonic
numProtectedPackets;	**6**	**bslbf**
mhasParity32Data;	**32**	**bslbf**
break;		
case PACTYP_DESCRIPTOR:		
for (i=0; i< MHASPacketLength; i++) {		
mhas_descriptor_data_byte(i);	**8**	**bslbf**
}		
break;		
case PACTYP_USERINTERACTION:		
mpegh3daElementInteraction();		
break;		
case PACTYP_LOUDNESS_DRC:		
mpegh3daLoudnessDrcInterface();		
break;		
case PACTYP_BUFFERINFO:		
mhas_buffer_fullness_present	**1**	**uimsbf**
if (mhas_buffer_fullness_present)		
mhas_buffer_fullness = escapedValue(15,24,32);	**15,39,71**	**uimsbf**
}		
break;		
case PACTYP_AUDIOTRUNCATION:		
audioTruncationInfo();		
break;		
case PACTYP_GENDATA:		
GenDataPayload();		
break;		
}		
ByteAlign();		
}		

14.2.2 Subsidiary MHAS syntax elements

Table 221 — Syntax of mpegh3daLoudnessDrcInterface()

Syntax	No. of bits	Mnemonic
mpegh3daLoudnessDrcInterface()		
{		
uniDrcInterface(); /* as defined in ISO/IEC 23003-4 */		
}		

Table 222 — Syntax of audioTruncationInfo()

Syntax	No. of bits	Mnemonic
audioTruncationInfo()		
{		
isActive;	**1**	**bool**
ati_reserved;	**1**	**bool**
truncFromBegin;	**1**	**bool**
nTruncSamples;	**13**	**uimsbf**
}		

14.3 Semantics

14.3.1 mpeghAudioStreamPacket()

MHASPacketType This element specifies the payload type in the actual packet. The meaning of MHASPacketType is defined in Table 223. A decoder which does not support a certain MHASPacketType shall skip this packet and continue with the next package.

Table 223 — Value of MHASPacketType

MHASPacketType	Value
PACTYP_FILLDATA	0
PACTYP_MPEGH3DACFG	1
PACTYP_MPEGH3DAFRAME	2
PACTYP_AUDIOSCENEINFO	3
/ reserved for ISO use */*	4-5
PACTYP_SYNC	6
PACTYP_SYNCGAP	7
PACTYP_MARKER	8
PACTYP_CRC16	9
PACTYP_CRC32	10
PACTYP_DESCRIPTOR	11
PACTYP_USERINTERACTION	12
PACTYP_LOUDNESS_DRC	13
PACTYP_BUFFERINFO	14
PACTYP_GLOBAL_CRC16	15
PACTYP_GLOBAL_CRC32	16
PACTYP_AUDIOTRUNCATION	17
PACTYP_GENDATA	18
/ reserved for ISO use */*	19-127
/ reserved for use outside of ISO scope */*	128-261
/ reserved for ISO use */*	262-389
/ reserved for use outside of ISO scope */*	390-517
Application-specific MHASPacketType values are mandated to be in the space reserved for use outside of ISO scope. These shall be skipped by a decoder as a minimum of structure is required by the decoder to skip these extensions.	

MHASPacketLabel This element provides an indication of which packets belong together. For example, with using different labels, different MPEG-H 3D audio configuration structures may be assigned to particular sequences of MPEG-H 3D audio access units.
If this field is set to '0', the packet is of general interest and is related to the complete stream.

MHASPacketLength This element indicates the length of the MHASPacketPayload() in Bytes.

MHASPacketPayload() The payload for the actual MHASPacket.

14.3.2 MHASPacketPayload()

mpegh3daConfig() An MPEG-H 3D audio configuration structure as defined in subclause 5.2.2.1.

mpegh3daFrame() An MPEG-H 3D audio payload as defined in subclause 5.2.3.1.

mae_AudioSceneInfo() An MPEG-H 3D audio scene information structure as defined in subclause 15.2.

ByteAlign() Up to 7 fill bits to achieve byte alignment with respect to the beginning of the syntactic element in which ByteAlign() occurs.

syncSpacingLength the length in Bytes between the last two **MHASPacketType** PACTYP_SYNC.

mhasParity16Data a 16-bit field that contains the CRC value that yields a zero output of the 16 registers in the decoder with the polynomial:

$$x^{16} + x^{15} + x^5 + 1$$

and the initial state of the shift register of 0xFFFF.

mhasParity32Data a 32-bit field that contains the CRC value that yields a zero output of the 32 registers in the decoder with the polynomial:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

and the initial state of the shift register of 0xFFFFFFFF.

mhas_fill_data_byte 8-bit data elements, no restrictions apply

mpegh3daElementInteraction() MPEG-H 3D audio element interaction structure as defined in subclause 17.7.4

mpegh3daLoudnessDrcInterface() An MPEG-H 3D Loudness and DRC interface as defined in Table 221.

mhas_buffer_fullness_present a bit signalling the presence of mhas_buffer_fullness

mhas_buffer_fullness data element indicating the state of the bit reservoir in the course of encoding the access unit. It is transmitted as the number of available bytes in the bit reservoir. The state of the bit reservoir is derived according to subclause 5.6.2.

global_CRC_type This element provides an indication on the packet types allowed within the numProtectedPackets protected packets with the global CRC specified in MHASPacketType PACTYP_GLOBAL_CRC32 or PACTYP_GLOBAL_CRC16, according to Table 224.

Table 224 — Value of global_CRC_type

Value	Indication on the packet types allowed within the numprotectedpackets
0	multiple packets of any packet type.
1	multiple packets of any packet type, of which only one packet of type PACTYP_MPEGH3DAFRAME.
2-3	reserved for ISO use

numProtectedPackets a 6-bit field that indicates the number of MHAS packets protected by the CRC check defined in the MHASPacketType PACTYP_GLOBAL_CRC16 and PACTYP_GLOBAL_CRC32.

14.3.3 Subsidiary MHAS packets

isActive If 1 the truncation message is active, if 0 the decoder should ignore the message.

ati_reserved reserved bit shall be zero.

truncFromBegin if 0 truncate samples from the end, if 1 truncate samples from the beginning.

nTruncSamples number of samples to truncate.

14.4 Description of MHASPacketTypes

14.4.1 PACTYP_FILLDATA

The MHASPacketType PACTYP_FILLDATA provides the possibility to add fill data to adjust the instantaneous bit-rate. This may be desirable in certain real-time applications where data is transmitted over a constant rate channel.

MHASPacketLabel shall be 0, because packets of this type do not relate to certain payload data.

It is expected that decoders will ignore the data transmitted in packets of type PACTYP_FILLDATA. Furthermore, intermediate tools that are processing an MHAS streams are allowed to remove such packets from the stream.

It is allowed to set MHASPacketLength to 0. This yields in a minimum packet size of 2 bytes.

14.4.2 PACTYP_MPEGH3DACFG

MHAS Packets of the MHASPacketType PACTYP_MPEG3DACFG embed an MPEG-H 3D audio configuration structure, mpegh3daConfig(), in the MHASPacketPayload().

MHASPacketLabel indicates the ID used for this configuration. This configuration shall be used for packets of type PACTYP_MPEGH3DAFRAME that have the same ID, so that a unique link between configuration data and payloads is possible.

If MHASPacketType of type PACTYP_MPEGH3DACFG is not present, the mpegh3daConfig() should be conveyed through out-band means, such as session announcement/description/control protocols.

14.4.3 PACTYP_MPEGH3DAFRAME

MHAS Packets of the MHASPacketType PACTYP_MPEG3DAFRAME embed a frame of MPEG-H 3D audio, mpegh3daFrame(), in the MHASPacketPayload().

MHASPacketLabel indicates the ID used for this payload. Packets of type PACTYP_MPEGH3DACFG that have the same ID carry the corresponding configuration, so that a unique link between configuration data and payloads is possible.

14.4.4 PACTYP_SYNC

The syncword payload for MPEG-H audio streams is '1010 0101'.

For this packet type, MHASPacketLabel has no meaning and shall be set to 0.

NOTE: The complete syncword is not only determined by the syncword payload, but by the complete mpeghAudioStreamPacket() with the MHASPacketType of PACTYP_SYNC, including all its fields. Therefore, a packet of type PACTYP_SYNC is 0xC001A5.

14.4.5 PACTYP_SYNCGAP

The MHASPacketType PACTYP_SYNCGAP may be used to improve synchronization to a stream. It directly follows a MHASPacketType PACTYP_SYNC.

For this packet type, MHASPacketLabel has no meaning and shall be set to 0.

14.4.6 PACTYP_MARKER

14.4.6.1 General

The MHASPacketType PACTYP_MARKER indicates a marker event in the stream. The event type is signalled in the packet payload.

Table 225 — Meaning of marker_byte

Value of first marker_byte	Meaning
0x01	Configuration change marker
0x02	Random access/Immediate playout marker
0x03	Program boundary marker
0x04 ... 0xDF	*Reserved for ISO use*
0xE0 ... 0xFF	*Reserved for non-ISO use*

14.4.6.2 Configuration change marker

When the first marker_byte of the packet payload is "0x01", a configuration change occurred and evaluating the configuration structure mpegh3daConfig() in the following MHASPacketType PACTYP_MPEG3DACFG is mandatory.

For packets of PACTYP_MPEG3DACFG without a preceding configuration change marker packet, evaluation of the configuration structure is not required.

14.4.6.3 Random access/Immediate playout marker

When the first marker_byte of the packet payload is "0x02", the following packet of type PACTYP_MPEG3DAFRAME with identical MHASPacketLabel is encoded following the rules given in subclause 5.7.

14.4.6.4 Program boundary marker

When the first marker_byte of the packet payload is "0x03", a program boundary occurs at this point in time and all following packets belong to a new program.

14.4.7 PACTYP_CRC16 and PACTYP_CRC32

The MHASPacketType PACTYP_CRC16 and PACTYP_CRC32 may be used for detection of errors in the subsequent MHAS packet with MHASPacketLabel set to the same value. It shall be directly followed by the MHAS packet which its CRC value refers to. This may be beneficial when an MHAS stream is conveyed over an error prone channel.

The error detection method uses one of the generator polynomial and associated shift register states as defined for mhasParity16Data or mhasParity32Data respectively.

All bits of the MHASPacketPayload() (including the ByteAlign()) of the protected MHAS packet are included into the CRC-check.

In the case where there are no errors, each of the outputs of the shift register shall be zero. At the CRC encoder the **mhasParity16Data/mhasParity32Data** field is encoded with a value such that this is ensured.

14.4.8 PACTYP_DESCRIPTOR

The PACTYP_DESCRIPTOR may be used to embed MPEG-2 TS/PS descriptors in MHAS streams. Data conveyed as **mhas_descriptor_data_byte** shall be in accordance with the syntax and semantics as defined for descriptor() as defined in ISO/IEC 13818-1.

For this packet type and for descriptors transmitted in the first descriptor loop in the TS_program_map_section(), MHASPacketLabel shall be 0. TS_program_map_section() is defined in ISO/IEC 13818-1.

For this packet type and for descriptors assigned to one elementary stream (i.e. the second descriptor loop in the TS_program_map_section()), MHASPacketLabel have the same value as in the PACTYP_CONFIG from the associated elementary stream.

14.4.9 PACTYP_USERINTERACTION

The MHASPacketType PACTYP_USERINTERACTION may be used to feed element interaction data in the form of the mpegh3daElementInteraction() structure to the decoder.

For this packet type, MHASPacketLabel shall have the same value as the packets of MHASPacketType PACTYP_MPEGH3DACFG and PACTYP_AUDIOSCENEINFO (if present), which the user interaction data refers to. If its value is different, the user interaction specified in mpegh3daElementInteraction() structure shall not be applied.

14.4.10 PACTYP_LOUDNESS_DRC

The MHASPacketType PACTYP_LOUDNESS_DRC may be used to feed Loudness and DRC control data to the decoder.

For this packet type, MHASPacketLabel has the same value as the packet of MHASPacketType PACTYP_MPEGH3DACFG, which the control data refers to.

14.4.11 PACTYP_BUFFERINFO

The MHASPacketType PACTYP_BUFFERINFO may be used to indicate the buffer fullness of the encoded stream or sub-stream with MHASPacketLabel set to the same value.

If MHASPacketLabel is set to 0, the information in the packet relates to the overall stream including all available sub-streams.

If present, this packet shall be placed into the stream at that point of time it provides valid information; i.e. when present after one packet of MHASPacketType PACTYP_MPEGH3DAFRAME, it shall signal the buffer state after encoding of that access unit.

14.4.12 PACTYP_GLOBAL_CRC16 and PACTYP_ GLOBAL_CRC32

The MHASPacketType PACTYP_GLOBAL_CRC16 and PACTYP_ GLOBAL_CRC32 may be used for detection of errors in the subsequent numProtectedPackets MHAS packets. For this packet type, MHASPacketLabel has no meaning and shall be set to 0. This may be beneficial when an MHAS stream is conveyed over an error prone channel.

The error detection method uses one of the generator polynomials and associated shift register states as defined for mhasParity16Data or mhasParity32Data respectively.

The CRC-check includes:

— first all bits positioned before the mhasParity32Data/mhasParity16Data in the MHAS packet of type PACTYP_GLOBAL_CRC32 or PACTYP_GLOBAL_CRC16, corresponding to the fields: MHASPacketType, MHASPacketLabel, MHASPacketLength, global_CRC_type and numProtectedPackets;

— and afterwards all bits of the MHASAudioStreamPacket() of the subsequent numProtectedPackets MHAS packets.

In the case where there are no errors, each of the outputs of the shift register shall be zero. At the CRC encoder the **mhasParity16Data/mhasParity32Data** field is encoded with a value such that this is ensured.

14.4.13 PACTYP_AUDIOTRUNCATION

The MHAS package of type PACTYP_AUDIOTRUNCATION indicates a potential truncation. Truncation in this context means the removal of audio samples from the decoded PCM samples. Audio samples are removed either before or after a truncation point as signalled in the truncation packet.

The packet contains a flag, **isActive**, which indicates whether the truncation shall actually be applied. If this flag is 0 the truncation package shall be ignored.

If the process of truncation is applied after the mixing stage, i.e. after the signal has passed all core decoder and rendering stages, but before the post-processing (DRC-2, binauralization etc.) and end-of-chain (DRC-3 etc.) then the truncation point will occur at a point in time delayed by the core decoding and rendering delay.

If the process of truncation occurs at a later stage, after the mixing stage (e.g., after end-of-chain), the truncation point shall be delayed by a period corresponding to a delay of the additional processing blocks (e.g., DRC-2, binauralization) and the truncation shall be carried out at this later truncation stage (e.g. after end-of-chain). The decoded samples are either truncated from the beginning (if **truncFromBegin**==1) or from the end (if **truncFromBegin**==0). In the case of a truncation from the end, the decoder shall discard all samples after the truncation point during the truncation process. In the case of a truncation from the beginning, the decoder shall discard all samples up to the truncation point during the truncation process.

Metadata shall be applied to the truncated audio samples such that the resulting audio samples are identical to those that would have resulted from applying the metadata to the non-truncated audio sample frame.

For correct application of the truncation the following additional rules apply:

— If **truncFromBegin** == 1

  — The MHAS stream shall contain an additional MHAS packet of type PACTYP_MPEGH3DACFG;

  — The MHAS truncation packet shall occur after the PACTYP_MPEGH3DACFG and before the corresponding PACTYP_MPEGH3DAFRAME that contains the AU to truncate.

— If **truncFromBegin** == 0

  — The MHAS truncation packet shall occur before the corresponding PACTYP_MPEGH3DAFRAME that contains the AU to truncate;

  — If **isActive**==1 the decoder shall perform a decoder re-initialization as if a change of decoder configuration had occurred.

Truncation messages shall be processed jointly with the AU of the following PACTYP_MPEGH3DAFRAME packet with identical MHASPacketLabel.

14.4.14 PACTYP_AUDIOSCENEINFO

MHAS Packets of the MHASPacketType PACTYP_AUDIOSCENEINFO embed an MPEG-H 3D audio scene information structure, mae_AudioSceneInfo(), in the MHASPacketPayload().

MHASPacketLabel indicates the ID used for this audio scene information. This audio scene information shall be used for packets of type PACTYP_MPEGH3DAFRAME that have the same ID and packets of type PACTYP_MPEGH3DACFG, so that a unique link between audio scene information, configuration data and payloads is possible.

If present, the MHASPacketType PACTYP_AUDIOSCENEINFO shall occur immediately after each MHAS packet of type PACTYP_MPEGH3DACFG. In this case the mpegh3daConfig() within the MHAS packet of type PACTYP_MPEGH3DACFG shall not contain a mae_AudioSceneInfo() structure.

If MHAS packets of types PACTYP_MPEGH3DACFG and PACTYP_AUDIOSCENEINFO are both not present, the mae_AudioSceneInfo() can be conveyed through out-of-band means, such as session announcement/description/control protocols, either within the mpegh3daConfig() or separately.

14.5 Application examples

14.5.1 Light-weighted broadcast

Certain applications require a minimum overhead in terms of bitrate, while synchronization time is less critical. (e.g. internet radio).

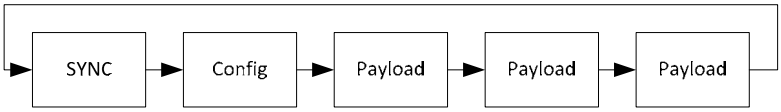


Figure 87 — Example 1

14.5.2 MPEG-2 transport stream

When embedding MPEG-H 3D audio streams into MPEG-2 transport streams, fast synchronization to a stream at random access points is most important, while bitrate overhead is usually less critical. Therefore, the configuration structure is sent on a regular basis, typically twice a second.

To improve synchronization to the stream, packets of type PACTYP_SYNC may be embedded more frequently and in addition the PACTYP_SYNCGAP type may also be embedded.

The figures below indicate some possible sequences of packet types when the MPEG-H audio stream is intended to be embedded into the MPEG-2 transport stream.

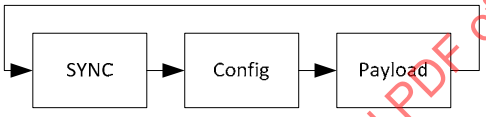


Figure 88 — Example 2

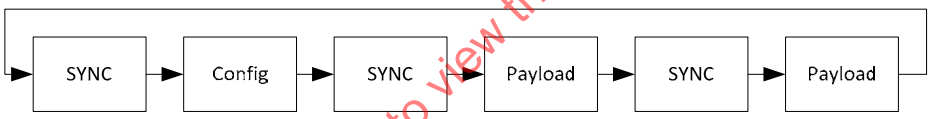


Figure 89 — Example 3

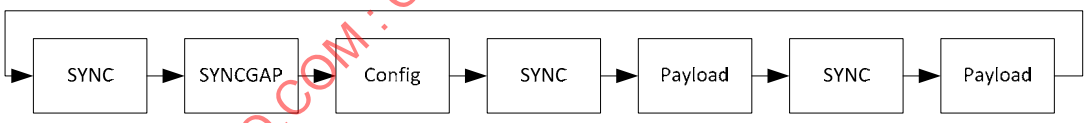


Figure 90 — Example 4

14.5.3 CRC error detection

Certain applications require additional error protection, for example if MHAS is used on common serial interfaces (e.g. AES/EBU, S/PDIF), while a minimizing the additional bitrate overhead. Figure 91 and Figure 92 illustrate two examples on how the CRC and global CRC packets can be used for error detection.

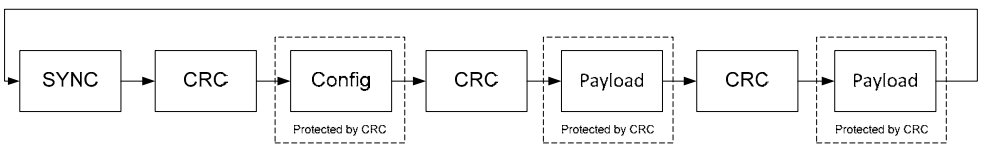


Figure 91 — Example 5

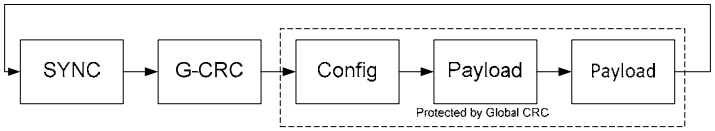


Figure 92 — Example 6

14.5.4 Audio sample truncation

Some applications may require that at least some audio frames need to be truncated to a number of audio samples which is less than the audio frame size. One example of such an application is sample accurate alignment of an audio stream to a video stream that has a different frame rate. Furthermore, this truncation may have to be inserted on the fly, e.g., for stream splicing. Such an example of stream splicing using the MHAS packets of type PACTYP_AUDIOTRUNCATION for switching between two MHAS streams, coming from different sources, is illustrated in Figure 93.

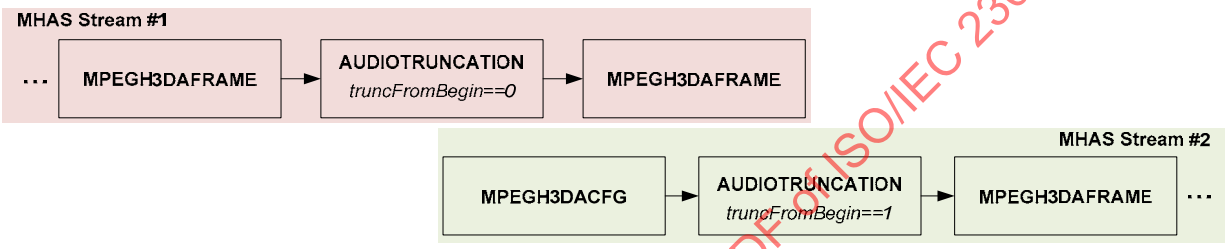


Figure 93 — Example 7

In the first stream, the MHAS packet containing the truncation message with the number of samples to be truncated at the end of the AU of the following PACTYP_MPEGH3DAFRAME indicates also the end of the stream.

14.6 Multi-stream delivery and interface

This subclause addresses the situation in which multiple incoming input streams form one program. These incoming streams may be merged into one MHAS stream prior to decoding. By assigning each of the incoming streams (i.e. packets of type PACTYP_MPEGH3DAFRAME) and their related configuration structures (i.e. packets of type PACTYP_MPEGH3DACFG) a unique **MHASPacketLabel**, sub-streams within one MHAS stream are formed. As the sub-streams are generated by the same encoder, it is presumed that various incoming streams, despite being potentially delivered over different transmission channels, are completely aligned and have no phase offset. Compensation for different transmission delays shall be accomplished by the systems layer and is not covered by this specification.

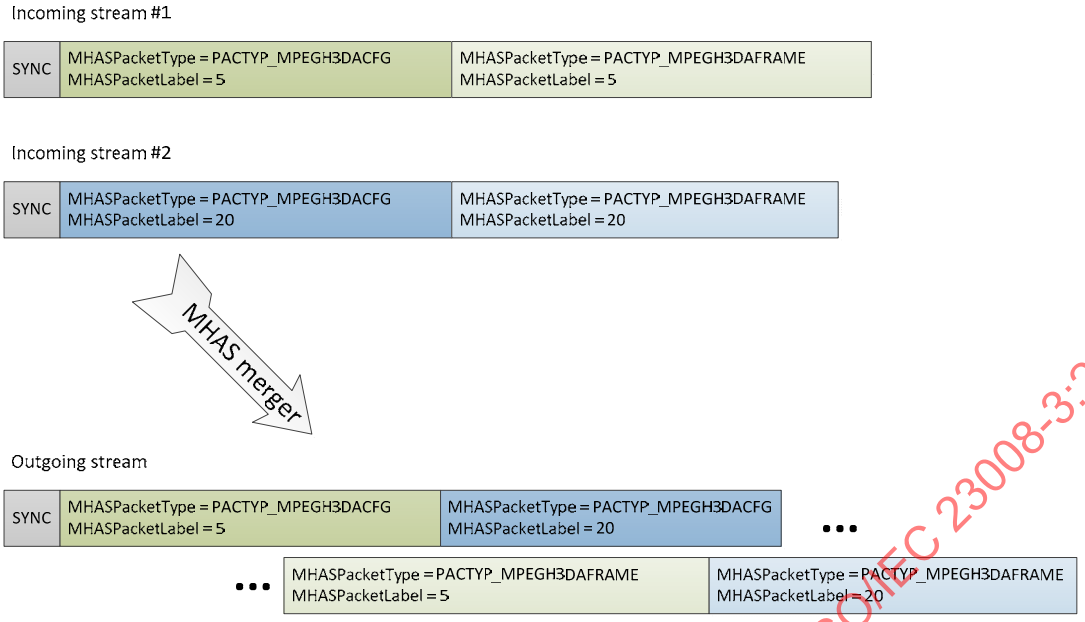


Figure 94 — Merging two MHAS streams

The MHAS packets of the incoming streams shall be merged into the outgoing steam such that all MHAS packets that belong to the same time instance and thus form one access unit are consecutively written into the outgoing stream.

In general, no specific order of MHAS sub stream packets needs to be maintained if the sub stream packets belong to one access unit. However it is recommended to order the sub-streams in ascending order of the packet label number. Through this procedure, each received new packet with a higher label belongs to the same time slot as the previous packets.

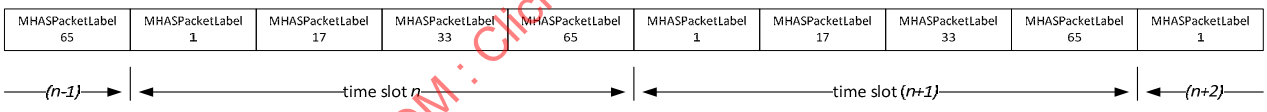


Figure 95 — Time slots with multiple MHAS packets

Though it may be desirable, it is not required that the sub-stream with **mae_isMainStream** set to '1' has the lowest **MHASPacketLabel** number.

As specified in subclause 14.3.1 incoming packets with **MHASPacketLabel** set to '0' are related to the complete stream, at least packets of type PACTYP_MPEGH3DAFRAME shall not be labeled with '0' when more than one sub-stream is present in the MHAS stream.

The multi-stream enabled decoder shall be capable of handling those streams with sub-streams labeled with different **MHASPacketLabels**. By utilizing the field **mae_bsMetaDataElementIDoffset** in mae_AudioSceneInfo() (see 15.3), the decoder shall be capable of arranging the sub-streams in the correct order before decoding.

To enable identification of several configurations inside one stream in the case of multi-stream delivery, the MHAS packet label shall be used as follows: The label values "1" to "16" (0x01-0x10) shall be used for the main stream, the label values "17" to "32" (0x11-0x20) shall be used for the first sub-stream, the next 16 values (0x21-0x30) for the second sub-stream, and so on.

Table 226 — Meaning of MHASPacketLabel in multi-stream environments

Value of MHASPacketLabel	Meaning
0x01-0x10	Main stream
0x11-0x20	First sub-stream
0x21-0x30	Second sub-stream
...	...

EXAMPLE The receiver has several incoming streams, from which he chooses the main stream containing the channel bed, additional effect channels and the main dialog in language 1 and the stream #N containing the audio description in language 1. These two MHAS streams are merged by an MHAS merger into one stream containing two sub-streams.

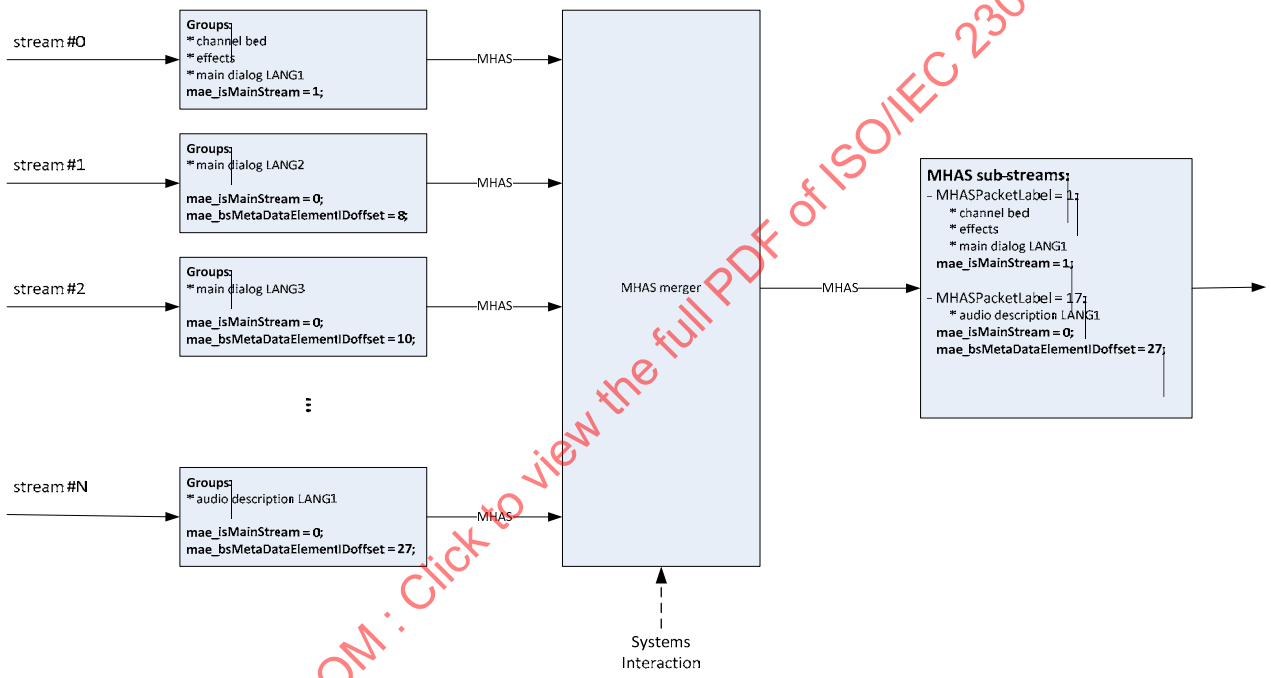


Figure 96 — Example of switching and merging multiple incoming streams

The combined stream has the following structure.

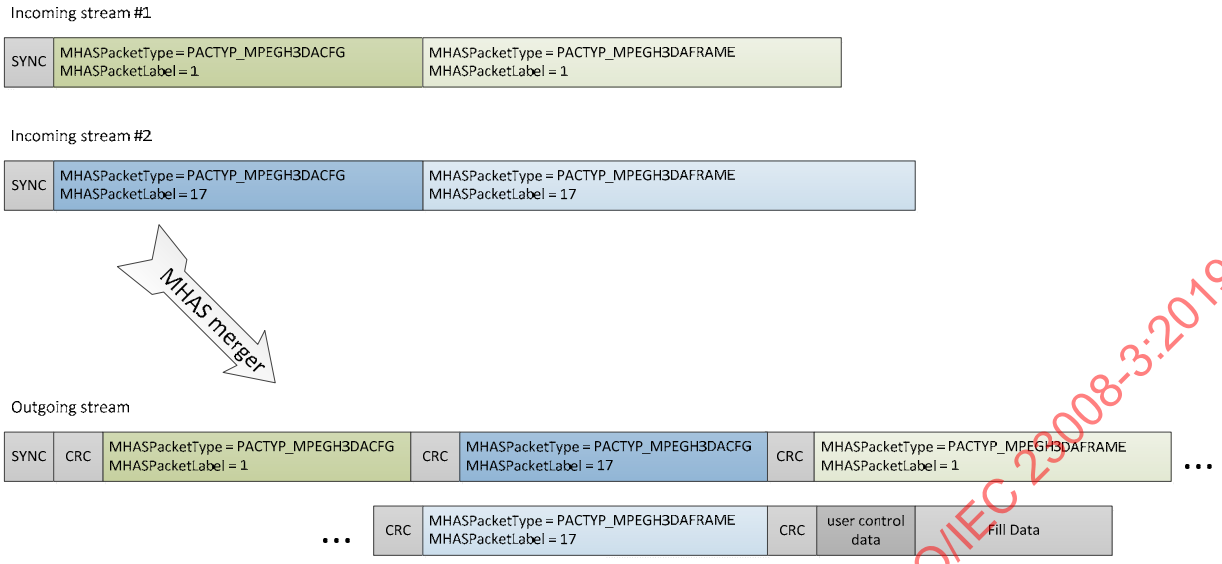


Figure 97 — Example output bitstream from MHAS merger

Additionally, the MHAS merger may also include or discard additional packages such as PACTYP_CRC32 as required by the subsequent transmission line.

14.7 Carriage of generic data

The generic data is encapsulated in an MHAS Packet as part of an MHAS audio data stream.

14.7.1 Syntax

For the MHAS packet with MHASPacketType of **PACTYP_GENDATA,** GenDataPayload is defined as shown in Table 227.

If a packet with MHASPacketType of **PACTYP_GENDATA** is present in the MPEG-H 3D audio MHAS stream, MHAS packets of type **PACTYP_MPEGH3DAFRAME** shall also be present in the stream. The total bitrate associated with all packets of type **PACTYP_GENDATA** shall not be greater than 1 % of the audio bitrate, averaged over a 5 s time window.

For example, if the nominal audio bitrate is 512 kb/s and one **PACTYP_GENDATA** MHAS packet of length 256 bytes is transmitted each second, the bitrate associated with these packets is 0.4 % of the audio bitrate.

Table 227 — Syntax of GenDataPayload

Syntax	No. of bits	Mnemonic
GenDataPayload()		
{		
dataType	**8**	**uimsbf**
if (dataType == 0) {		
t35Code;	**16…40**	**uimsbf**
}		
else if (dataType == 0xFF) {		
uuid;	**128**	**uimsbf**
shortUuid;	**8**	**uimsbf**
}		
genData;	**var*8**	**uimsbf**
}		

14.7.2 Semantics

dataType This element indicates corresponding reference of the generic data as shown in Table 228. This field takes either a dynamically assigned value (see below under shortUUID) or one of two reserved values.

Table 228 — Value of datatype

Value	Corresponding references
0	ITU T.35 ID follows
1-254	**dataType** is UUID short form (no additional ID follows) See **uuid** and **shortUuid** field semantics below
255	Full UUID and its short form follow

t35Code For **dataType** value 0, this field shall be the "country code" field (1 or 2 bytes) followed by the "manufacturer code" field (1 or more bytes) as defined by ITU T.35 [12]. The length is variable, but cannot be less than 2 bytes.

uuid A binary encoding of UUID. The value of **uuid** shall be as defined in IETF RFC 4122.

shortUuid For **dataType** values 1-254, the **dataType** value is the short form value of the UUID corresponding to the previous field, **uuid**. The long form identifier of the system type, **uuid**, is dynamically mapped to this short form identifier, **shortUuid**, and this short form identifier can occur later in the same stream to refer to the associated UUID. This technique saves identifier overhead. The **dataType** value 255 both provides **genData** associated with a UUID, and establishes a mapping between a **shortUuid** and the full **uuid**, such that the **dataType** field in subsequent packets may be equal to **shortUuid** to indicate that the **genData** in those packets is associated with this previously mapped **uuid**. The value 0 or 255 shall not be used for **shortUuid**.

Note that the system for which the generic metadata applies is responsible for ensuring that the frequency of establishment, and the expiry period (if any), of mappings between short form and long form identifiers are appropriate to the application.

genData The variable length generic data payload. Its length is the bytes that remain from the MHAS Packet length after the preceding fields in GenDataPayload().

14.7.3 Processing at the MPEG-H 3D audio decoder

MPEG-H 3D audio decoder should extract the MHAS packet with **PACTYP_GENDATA** and, if **dataType**, **t35Code**, **uuid** and **shortUuid** that occur in the GenDataPayload() are understood, deliver the complete packet (including the MHAS header) in binary to a generic data engine known to handle such a **dataType**. If the **dataType**, **t35Code**, **uuid** and **shortUuid** fields in the GenDataPayload() are not understood, the audio decoder should discard the packet. Both the length and value of the **uuid** and **t35Code** fields of interest to the decoder are pre-known to the decoder. For **t35Code,** there is the possibility of a length overrun (e.g. checking for a length of 4 bytes when it is only 3 bytes long). Decoders should ensure the comparison length does not exceed the MHAS packet length. Any overrun on the comparison length will spill harmlessly into the **genData** and the comparison will always fail in such conditions due to the progressive nature of the t35Code encoding. That is, if all the bytes match in **t35Code** then it is a match regardless of the **genData** contents.

15 Metadata audio elements (MAE)

15.1 General

The set of metadata consists of:

— Descriptive metadata: Information about the existence of objects inside the bitstream and high-level properties of objects;

— Restrictive metadata: Information of how interaction is possible or enabled by the content creator;

— Positional metadata and ability to render to specific loudspeakers and to signal channel content as objects;

— Structural metadata: Grouping and combination of objects.

The metadata is organized as groups or switch groups of elements. All static metadata is organized in the mae_AudioSceneInfo() in the mpegh3daConfigExtension(). The Main Audio Scene describes the full scene in terms of the static metadata of all elements.

Groups of elements

Related elements are associated with a group. These elements belong together and can only be manipulated together. Elements of a group can no longer be interactively altered on their own (e.g. if a group is switched off, all its child elements are switched off) but altered as a unit. Examples are channel-based recordings, e.g. an AB recording where the two recorded signals belong together and should only be manipulated as a pair. This grouping allows for signaling of stems and submixes by gathering the dedicated elements in groups.

Switch groups of elements

It should be possible to define special groups, where just one or a selected number of grouped elements can be switched on at a given time. This group is called a switch group. This concept permits an audience to enable just one element from a group of related elements when required. An example is the possibility to produce and transmit a multitude of language tracks.

Group presets

Group presets define a combination of groups in an audio scene. A group preset contains a list of groups or switch groups, each referenced by their unique ID and an associated on/off-status for each of the referenced structures (presets' conditions) which are not modifiable by the user. With group presets a content creator or an application can provide a restricted number of meaningful rendering options to the user.

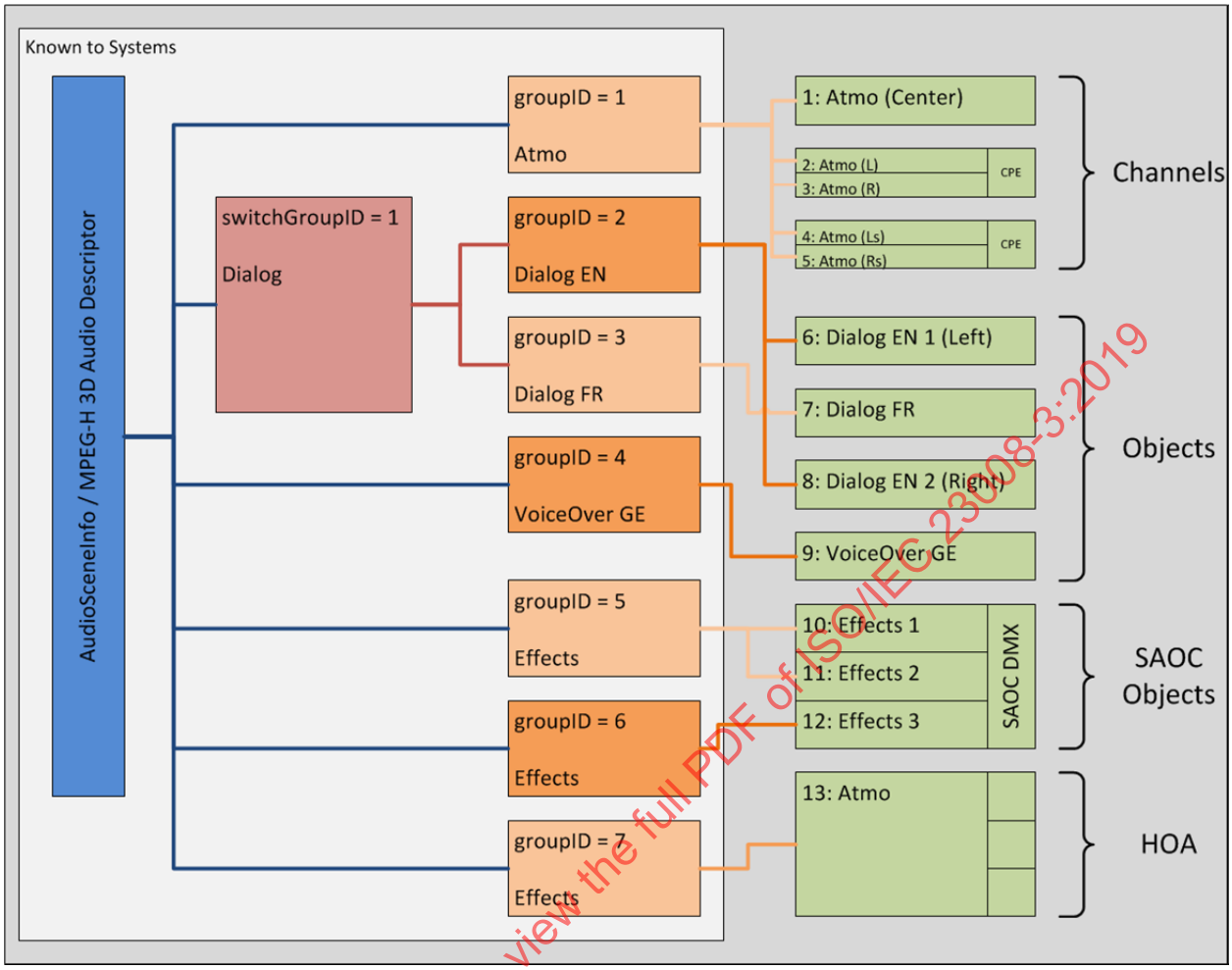


Figure 98 — Example structure of metadata elements

15.2 Syntax

Table 229 — Syntax of mae_AudioSceneInfo()

Syntax	No. of bits	Mnemonic
mae_AudioSceneInfo() {		
mae_isMainStream;	**1**	**bslbf**
if (mae_isMainStream) {		
mae_audioSceneInfoIDPresent;	**1**	**bslbf**
if (mae_audioSceneInfoIDPresent) {		
mae_audioSceneInfoID;	**8**	**uimsbf**
}		
mae_numGroups;	**7**	**uimsbf**
mae_GroupDefinition(mae_numGroups);		
mae_numSwitchGroups;	**5**	**uimsbf**
mae_SwitchGroupDefinition(mae_numSwitchGroups);		
mae_numGroupPresets;	**5**	**uimsbf**
mae_GroupPresetDefinition(mae_numGroupPresets);		
mae_Data();		
mae_metaDataElementIDoffset = 0;		
mae_metaDataElementIDmaxAvail;	**7**	**uimsbf**

Syntax	No. of bits	Mnemonic
}		
else {		
mae_bsMetaDataElementIDoffset;	**7**	**uimsbf**
mae_metaDataElementIDoffset = mae_bsMetaDataElementIDoffset + 1;		
mae_metaDataElementIDmaxAvail;	**7**	**uimsbf**
}		
}		

Table 230 — Syntax of mae_Data()

Syntax	No. of bits	Mnemonic
mae_Data()		
{		
mae_numDataSets;	**4**	**uimsbf**
for (dscr = 0; dscr < mae_numDataSets; dscr ++) {		
mae_dataType[dscr]**;**	**4**	**uimsbf**
mae_dataLength[dscr]**;**	**16**	**uimsbf**
switch (mae_dataType[dscr]) {		
case ID_MAE_GROUP_DESCRIPTION:		
mae_Description(ID_MAE_GROUP_DESCRIPTION);		
break;		
case ID_ MAE_SWITCHGROUP_DESCRIPTION:		
mae_Description(ID_MAE_SWITCHGROUP_DESCRIPTION);		
break;		
case ID_ MAE_GROUP_PRESET_DESCRIPTION:		
mae_Description(ID_MAE_GROUP_PRESET_DESCRIPTION);		
break;		
case ID_ MAE_GROUP_CONTENT:		
mae_ContentData();		
break;		
case ID_MAE_GROUP_COMPOSITE:		
mae_CompositePair();		
break;		
case ID_MAE_SCREEN_SIZE:		
mae_ProductionScreenSizeData();		
break;		
case ID_MAE_DRC_UI_INFO:		
mae_DrcUserInterfaceInfo();		
break;		
case ID_MAE_SCREEN_SIZE_EXTENSION:		
mae_ProductionScreenSizeDataExtension();		
break;		
case ID_MAE_GROUP_PRESET_EXTENSION:		
mae_GroupPresetDefinitionExtension();		
break;		
case ID_MAE_LOUDNESS_COMPENSATION:		
mae_LoudnessCompensationData(mae_numGroups, mae_numGroupPresets);		
break;		
default:		
while (mae_dataLength[dscr]--) {		
tmp;	**8**	**uimsbf**
}		

Syntax	No. of bits	Mnemonic
break;		
}		
}		
}		

Table 231 — Syntax of mae_GroupDefinition()

Syntax	No. of bits	Mnemonic
mae_GroupDefinition(numGroups)		
{		
for (grp = 0; grp < numGroups; grp++) {		
mae_groupID[grp];	7	uimsbf
mae_allowOnOff[grp];	1	bslbf
mae_defaultOnOff[grp];	1	bslbf
mae_allowPositionInteractivity[grp];	1	bslbf
if (mae_allowPositionInteractivity[grp]) {		
mae_interactivityMinAzOffset[grp];	7	uimsbf
mae_interactivityMaxAzOffset[grp];	7	uimsbf
mae_interactivityMinElOffset[grp];	5	uimsbf
mae_interactivityMaxElOffset[grp];	5	uimsbf
mae_interactivityMinDistFactor[grp];	4	uimsbf
mae_interactivityMaxDistFactor[grp];	4	uimsbf
}		
mae_allowGainInteractivity[grp];	1	bslbf
if (mae_allowGainInteractivity[grp]) {		
mae_interactivityMinGain[grp];	6	uimsbf
mae_interactivityMaxGain[grp];	5	uimsbf
}		
mae_bsGroupNumMembers[grp];	7	uimsbf
mae_hasConjunctMembers[grp];	1	bslbf
if (mae_hasConjunctMembers[grp]) {		
mae_startID[grp];	7	uimsbf
}		
else {		
for (obj = 0; obj < mae_bsGroupNumMembers[grp] + 1; obj++) {		
mae_metaDataElementID[grp][obj];	7	uimsbf
}		
}		
}		
}		

Table 232 — Syntax of mae_SwitchGroupDefinition()

Syntax	No. of bits	Mnemonic
mae_SwitchGroupDefinition(numSwitchGroups)		
{		
for (swgrp = 0; swgrp < numSwitchGroups; swgrp ++) {		
mae_switchGroupID[swgrp];	5	uimsbf
mae_switchGroupAllowOnOff[swgrp];	1	bslbf
if (mae_switchGroupAllowOnOff[swgrp]) {		
mae_switchGroupDefaultOnOff[swgrp];	1	bslbf

Syntax	No. of bits	Mnemonic
}		
mae_bsSwitchGroupNumMembers[swgrp];	**5**	**uimsbf**
for (grp = 0; grp < mae_bsSwitchGroupNumMembers[swgrp] + 1; grp++) {		
mae_switchGroupMemberID[swgrp][grp];	**7**	**uimsbf**
}		
mae_switchGroupDefaultGroupID[swgrp];	**7**	**uimsbf**
}		
}		

Table 233 — Syntax of mae_Description()

Syntax	No. of bits	Mnemonic
mae_Description(type)		
{		
mae_bsNumDescriptionBlocks;	**7**	**uimsbf**
for (n = 0; n < mae_bsNumDescriptionBlocks + 1; n++) {		
if (type == ID_MAE_GROUP_DESCRIPTION) {		
mae_descriptionGroupID[n];	**7**	**uimsbf**
}		
else if (type == ID_MAE_SWITCHGROUP_DESCRIPTION) {		
mae_descriptionSwitchGroupID[n];	**5**	**uimsbf**
}		
else if (type == ID_MAE_GROUPPRESET_DESCRIPTION) {		
mae_descriptionGroupPresetID[n];	**5**	**uimsbf**
}		
mae_bsNumDescLanguages[n];	**4**	**uimsbf**
for (i=0; i<mae_bsNumDescLanguages[n] + 1; i++) {		
mae_bsDescriptionLanguage[n][i];	**24**	**uimsbf**
mae_bsDescriptionDataLength[n][i];	**8**	**uimsbf**
for (c = 0; c < mae_bsDescriptionDataLength [n][i] + 1; c++) {		
mae_descriptionData[n][i][c];	**8**	**uimsbf**
}		
}		
}		
}		

Table 234 — Syntax of mae_ContentData()

Syntax	No. of bits	Mnemonic
mae_ContentData()		
{		
mae_bsNumContentDataBlocks;	**7**	**uimsbf**
for (n = 0; n < mae_bsNumContentDataBlocks + 1; n++) {		
mae_ContentDataGroupID[n];	**7**	**uimsbf**
mae_contentKind[n];	**4**	**uimsbf**
mae_hasContentLanguage[n];	**1**	**bslbf**
if (mae_hasContentLanguage[n]) {		
mae_contentLanguage[n];	**24**	**uimsbf**
}		
}		
}		

Table 235 — Syntax of mae_CompositePair()

Syntax	No. of bits	Mnemonic
mae_CompositePair()		
{		
mae_bsNumCompositePairs;	**7**	**uimsbf**
for (sa=0; sa<mae_bsNumCompositePairs + 1; sa++) {		
mae_CompositeElementID [sa][0]**;**	**7**	**uimsbf**
mae_CompositeElementID [sa][1]**;**	**7**	**uimsbf**
}		
}		

Table 236 — Syntax of mae_GroupPresetDefinition()

Syntax	No. of bits	Mnemonic
mae_GroupPresetDefinition(numGroupPresets)		
{		
for (gp = 0; gp < numGroupPresets; gp++) {		
mae_groupPresetID[gp];	**5**	**uimsbf**
mae_groupPresetKind[gp];	**5**	**uimsbf**
mae_bsGroupPresetNumConditions[gp][0]**;**	**4**	**uimsbf**
for (cnd = 0; cnd < mae_bsGroupPresetNumConditions[gp][0] + 1; cnd++) {		
mae_groupPresetReferenceID[gp][cnd]**;**	**7**	**uimsbf**
mae_groupPresetConditionOnOff[gp][0][cnd]**;**	**1**	**bslbf**
if (mae_groupPresetConditionOnOff[gp][0][cnd]) {		
mae_groupPresetDisableGainInteractivity[gp][0][cnd];	**1**	**bslbf**
mae_groupPresetGainFlag[gp][0][cnd]**;**	**1**	**bslbf**
if (mae_groupPresetGainFlag[gp][0][cnd]) {		
mae_groupPresetGain[gp][0][cnd];	**8**	**uimsbf**
}		
mae_groupPresetDisablePositionInteractivity[gp][0][cnd];	**1**	**bslbf**
mae_groupPresetPositionFlag[gp][0][cnd]**;**	**1**	**bslbf**
if (mae_groupPresetPositionFlag [gp][0][cnd]) {		
mae_groupPresetAzOffset[gp][0][cnd];	**8**	**uimsbf**
mae_groupPresetElOffset[gp][0][cnd];	**6**	**uimsbf**
mae_groupPresetDistFactor[gp][0][cnd];	**4**	**uimsbf**
}		
}		
}		
}		
}		

Table 237 — Syntax of mae_ProductionScreenSizeData()

Syntax	No. of bits	Mnemonic
mae_ProductionScreenSizeData()		
{		
hasNonStandardScreenSize;	**1**	**bslbf**
if (hasNonStandardScreenSize) {		
bsScreenSizeAz;	**9**	**uimsbf**
bsScreenSizeTopEl;	**9**	**uimsbf**
bsScreenSizeBottomEl;	**9**	**uimsbf**
}		
}		

Table 238 — Syntax of mae_LoudnessCompensationData()

Syntax	No. of bits	Mnemonic
mae_LoudnessCompensationData(numGroups,numGroupPresets)		
{		
mae_loudnessCompGroupLoudnessPresent;	**1**	**bslbf**
if (mae_loudnessCompGroupLoudnessPresent == 1) {		
for(grp=0; grp<numGroups; grp++) {		
mae_bsLoudnessCompGroupLoudness[grp];	**8**	**uimsbf**
}		
} else {		
/* not present or provided by mpegh3daLoudnessInfoSet() */		
}		
mae_loudnessCompDefaultParamsPresent;	**1**	**bslbf**
if (mae_loudnessCompDefaultParamsPresent == 1) {		
for(grp=0; grp<numGroups; grp++) {		
groupID = mae_groupID[grp];		
mae_loudnessCompDefaultIncludeGroup[grp];	**1**	**bslbf**
}		
mae_loudnessCompDefaultMinMaxGainPresent;	**1**	**bslbf**
if (mae_loudnessCompDefaultMinMaxGainPresent == 1) {		
mae_bsLoudnessCompDefaultMinGain;	**4**	**uimsbf**
mae_bsLoudnessCompDefaultMaxGain;	**4**	**uimsbf**
}		
}		
for (gp=0; gp<numGroupPresets; gp++) {		
groupPresetID = mae_groupPresetID[gp];		
mae_loudnessCompPresetParamsPresent[gp];	**1**	**bslbf**
if (mae_loudnessCompPresetParamsPresent[gp] == 1) {		
for (grp=0; grp<numGroups; grp++) {		
groupID = mae_groupID[grp];		
mae_loudnessCompPresetIncludeGroup[gp][grp];	**1**	**bslbf**
}		
mae_loudnessCompPresetMinMaxGainPresent[gp];	**1**	**bslbf**
if (mae_loudnessCompPresetMinMaxGainPresent[gp]) {		
mae_bsLoudnessCompPresetMinGain[gp];	**4**	**uimsbf**
mae_bsLoudnessCompPresetMaxGain[gp];	**4**	**uimsbf**
}		
}		
}		
}		

Table 239 — Syntax of mae_ProductionScreenSizeDataExtension()

Syntax	No. of bits	Mnemonic
mae_ProductionScreenSizeDataExtension()		
{		
mae_overwriteProductionScreenSizeData;	**1**	**bslbf**
if (mae_overwriteProductionScreenSizeData) {		
/* NON-CENTERED DEFAULT PRODUCTION SCREEN */		
bsScreenSizeLeftAz;	**10**	**uimsbf**

Syntax	No. of bits	Mnemonic
bsScreenSizeRightAz;	**10**	**uimsbf**
}		
mae_NumPresetProductionScreens;	**5**	**uimsbf**
for (n = 0; n < mae_NumPresetProductionScreens; n++) {		
mae_productionScreenGroupPresetID[n];	**5**	**uimsbf**
mae_hasNonStandardScreenSize[n]**;**	**1**	**bslbf**
if (mae_hasNonStandardScreenSize[n]) {		
isCenteredInAzimuth[n];	**1**	**bslbf**
if (isCenteredInAzimuth[n]) {		
bsScreenSizeAz[n];	**9**	**uimsbf**
} else {		
bsScreenSizeLeftAz[n]**;**	**10**	**uimsbf**
bsScreenSizeRightAz[n]**;**	**10**	**uimsbf**
}		
bsScreenSizeTopEl[n]**;**	**9**	**uimsbf**
bsScreenSizeBottomEl[n]**;**	**9**	**uimsbf**
}		
}		
}		

Table 240 — Syntax of mae_GroupPresetDefinitionExtension()

Syntax	No. of bits	Mnemonic
mae_GroupPresetDefinitionExtension()		
{		
for (gp = 0; gp < mae_numGroupPresets; gp++) {		
mae_hasSwitchGroupConditions[gp];	**1**	**bslbf**
if (mae_hasSwitchGroupConditions[gp]) {		
temp = mae_bsGroupPresetNumConditions[gp][0] +1;		
for (cnd = 0; cnd < temp; cnd++) {		
mae_isSwitchGroupCondition[gp][0][cnd];	**1**	**bslbf**
}		
}		
mae_hasDownmixIdGroupPresetExtensions[gp];	**1**	**bslbf**
if (mae_hasDownmixIdGroupPresetExtensions[gp]) {		
mae_numDownmixIdGroupPresetExtensions[gp];	**5**	**uimsbf**
for (egp = 1; egp < mae_numDownmixIdGroupPresetExtensions[gp] +1; egp++) {		
mae_groupPresetDownmixId[gp][egp];	**7**	**uimsbf**
mae_bsGroupPresetNumConditions[gp][egp]**;**	**4**	**uimsbf**
for (cnd = 0; cnd < mae_bsGroupPresetNumConditions[gp][egp] + 1; cnd++) {		
mae_isSwitchGroupCondition[gp][egp][cnd];	**1**	**bslbf**
if (mae_isSwitchGroupCondition[gp][egp][cnd]) {		
mae_groupPresetSwitchGroupID[gp][egp][cnd];	**5**	**uimsbf**
} else {		
mae_groupPresetGroupID[gp][egp][cnd];	**7**	**uimsbf**
}		
mae_groupPresetConditionOnOff[gp][egp][cnd];	**1**	**bslbf**
if (mae_groupPresetConditionOnOff[gp][egp][cnd] {		
mae_groupPresetDisableGainInteractivity[gp][egp][cnd];	**1**	**bslbf**
mae_groupPresetGainFlag[gp][egp][cnd];	**1**	**bslbf**
if (mae_groupPresetGainFlag[gp][egp][cnd]) {		
mae_groupPresetGain[gp][egp][cnd];	**8**	**uimsbf**
}		

Syntax	No. of bits	Mnemonic
mae_groupPresetDisablePositionInteractivity[gp][egp][cnd];	**1**	**bslbf**
mae_groupPresetPositionFlag[gp][egp][cnd];	**1**	**bslbf**
if (mae_groupPresetPositionFlag [gp][egp][cnd]) {		
mae_groupPresetAzOffset[gp][egp][cnd];	**8**	**uimsbf**
mae_groupPresetElOffset[gp][egp][cnd];	**6**	**uimsbf**
mae_groupPresetDistFactor[gp][egp][cnd];	**4**	**uimsbf**
}		
}		
}		
}		
}		
}		
}		

Table 241 — Syntax of mae_DrcUserInterfaceInfo()

Syntax	No. of bits	Mnemonic
mae_DrcUserInterfaceInfo()		
{		
version;	**2**	**uimsbf**
if (version==0) {		
bsNumTargetLoudnessConditions;	**3**	**uimsbf**
targetLoudnessValueLower[0] = -63;		
for(c=0; c<numTargetLoudnessConditions; c++) {		
bsTargetLoudnessValueUpper[c];	**6**	**uimsbf**
drcSetEffectAvailable[c];	**16**	**bslbf**
targetLoudnessValueLower[c+1] = targetLoudnessValueUpper[c];		
}		
} else {		
/* discard remaining bits signalled by mae_dataLength */		
}		
}		

15.3 Semantics

mae_isMainStream This flag signals if the actual MPEG-H stream is the main stream.

mae_audioSceneInfoIDPresent This flag defines if an ID for the current audio scene is present in the bitstream.

mae_audioSceneInfoID This field defines the ID for the current audio scene. This field can take values between 0 and 255. A value of 0 means that the ID shall not be evaluated.

mae_numGroups This field signals the number of groups in the overall audio scene (complete number of groups in the main stream plus all possible additional streams). This field shall take values between 0 and 127, a maximum number of 127 groups are assumed.

mae_numSwitchGroups This field signals the number of switch groups in the overall scene (complete number of switch groups in the main stream plus all possible additional streams). This field can take values between 0 and 31, resulting in a maximum number of 31 switch groups.

mae_numGroupPresets Number of defined group presets. This field can take values between 0 and 31, resulting in a maximum number of 31 group presets. A group preset is a subset of all groups of an overall scene where the on/off status of each of these groups is bound to a condition. With group presets it is possible to define a controlled behaviour in dependence of the on/off status of some of the groups and/or switch groups in an overall audio scene. A group preset is valid if all its associated conditions are true (logical AND of the conditions yields 1), i.e. if the on/off status of all associated groups and the on/off status of each associated switch group conformas to the defined conditions. A condition associated with a switch group with value 1 is true if one member of the switch group is switched on. A condition associated with a switch group with value 0 is true if all members of the switch group are switched off. Switch group conditions with value 0 are only applicable for switch groups whose mae_switchGroupAllowOnOff flag is equal to 1.

Any evaluation of the valid/selected preset (and therefore the application of preset-dependent values) shall only happen in the defined 'basic interaction mode' (see subclause 17.7.3). The chosen/selected preset is indicated by the presetID that is received via the mpegh3daElementInteraction() interface.

mae_bsMetaDataElementIDoffset This field defines the offset for the first metadata element of the current MPEG-H data stream. It is zero if the stream is the main stream.

mae_numDataSets This field defines the number of data sets that are following in the bitstream.

mae_dataType For each data element this field defines the type of description that follows in the bitstream.

Table 242 — Value of mae_dataType

mae_dataType	value	meaning
ID_MAE_GROUP_DESCRIPTION	0	Group description follows in the bitstream
ID_MAE_SWITCHGROUP_DESCRIPTION	1	Switch group description follows in the bitstream
ID_ MAE_GROUP_CONTENT	2	Group content information follows in the bitstream
ID_MAE_GROUP_COMPOSITE	3	Composite pair information follows in the bitstream
ID_MAE_SCREEN_SIZE	4	Information about the local screen size follows in the bitstream
ID_MAE_GROUP_PRESET_DESCRIPTION	5	Group preset description follows in the bitstream
ID_MAE_DRC_UI_INFO	6	Extension metadata with DRC user interface information
ID_MAE_SCREEN_SIZE_EXTENSION	7	Extension metadata of the screen size information follows in the bitstream
ID_MAE_GROUP_PRESET_EXTENSION	8	Extension metadata of the group preset definition follows in the bitstream
ID_MAE_LOUDNESS_COMPENSATION	9	Loudness compensation information follows in the bitstream
reserved	10 - 15	n/a

mae_dataLength This field defines the length in bytes of the data element that follows in the bitstream.

mae_groupID This field uniquely defines an ID for a group of metadata elements. This field can take values between 0 and 126.

mae_allowOnOff This flag defines if the audience is allowed to switch a metadata element group on and off (enable/disable playback).

mae_defaultOnOff This field defines the default status of a metadata element group, i.e. if this group is switched on (=1) or off (=0) by default.

mae_allowPositionInteractivity This flag defines if the audience is allowed to change the position of the elements of a metadata element group.

mae_interactivityMinAzOffset This field defines the minimum azimuth offset for changing the position of the members of a metadata element group, e.g. changing the original azimuth by a minimum offset of -30°. This field can take values between MinAzOffset = -180° and MinAzOffset = 0°:

MinAzOffset = –1.5 · mae_interactivityMinAzOffset

mae_interactivityMaxAzOffset This field defines the maximum azimuth offset for changing the position of the members of a metadata element group, e.g. changing the original azimuth by a maximum offset of +30°. This field can take values between MaxElOffset = 0° and MaxElOffset =180°:

MaxAzOffset = +1.5 · mae_interactivityMaxAzOffset

mae_interactivityMinElOffset This field defines the minimum elevation offset for changing the position of the members of a metadata element group, e.g. changing the original elevation by a minimum offset of -30°. This field can take values between MinElOffset = -90° and MinElOffset = 0°:

MinElOffset = –3 · mae_interactivityMinElOffset

mae_interactivityMaxElOffset This field defines the maximum elevation offset for changing the position of the members of a metadata element group, e.g. changing the original elevation by a maximum offset of +30°. This field can take values between MaxElOffset = 0° and MaxElOffset = 90°:

MaxElOffset = +3 · mae_interactivityMaxElOffset

mae_interactivityMinDistFactor This field defines the minimum distance change factor for interactively changing the position of the members of a metadata element group. The field describes a multiplication factor by which the original distance is changed, e.g. multiplied by a minimum factor of 0.5. This field can take values between 0 and 15 resulting in MinDistFactor between 0,000 25 and 8:

$$\text{MinDistFactor} = 2^{(\text{mae\_interactivityMinDistFactor} - 12)}$$

mae_interactivitiyMaxDistFactor This field defines the maximum distance change factor for interactively changing the position of the members of a metadata element group. The field describes a multiplication factor by which the original distance is changed, e.g. multiplied by a maximum

factor of 4. This field can take values between 0 and 15 resulting in MaxDistFactor between 0,000 25 and 8:

$$MaxDistFactor = 2^{(mae\_interactivityMaxDistFactor - 12)}$$

mae_allowGainInteractivity This flag defines if the audience is allowed to change the gain of a metadata element group.

mae_interactivityMinGain This field defines a minimum gain of the members of a metadata element group. The field can take values between
MinGain = –63 dB
and
MinGain = 0 dB
in 1 dB steps, with

MinGain in dB = **mae_interactivityMinGain** – 63

If **mae_interactivityMinGain** is set to 0, MinGain shall be set to minus infinity dB.

mae_interactivityMaxGain This field defines a maximum gain of the members of a metadata element group. The field can take values between
MaxGain = 0 dB
and
MaxGain =31 dB
in 1 dB steps, with

MaxGain in dB = **mae_interactivityMaxGain**

The value of MinGain and MaxGain define the interval in dB of allowed interactivity gain changes relative to the OAM gain if OAM data is present for the current group. For example, with a MinGain of - 2 dB and a MaxGain of 6 dB, the gain of the elements in the group shall be interactively altered only in the interval between -2 dB and +6 dB relative to their current OAM gain in dB. For groups without OAM data, the MinGain and MaxGain values define the interval in dB of allowed interactivity gain changes relative to the current audio gain (amplitude of the audio samples).

mae_bsGroupNumMembers This field signals the number of members of a group of metadata elements. The field can take values between 0 and 127 resulting in a maximum number of 128 members. mae_groups that are part of switch groups shall consist of one or multiple complete signalGroups.

mae_hasConjunctMembers This flag defines if all members of the metadata element group are coded consecutively in the bitstream.

mae_startID If the members of the element group are coded consecutively this field defines the offset for the first metadata element of this group.

mae_metaDataElementID	This field uniquely defines an ID for a metadata element (i.e. channels, dynamic objects, SAOC channels, SAOC objects, SignalGroupTypeSAOC signals, HOA). This field can take values between 0 and 127, resulting in a maximum of 128 metadata elements. One group shall only contain elements of the same signal type (static objects (channel-based signals without accompanying OAM data), objects (audio tracks with accompanying OAM data), SAOC channels, SAOC object, SignalGroupTypeSAOC signals, HOA). If a bitstream contains an mae_AudioSceneInfo() syntax element, each mae_metaDataElementID shall be referred to in exactly one mae_GroupDefinition() bitstream element.
mae_switchGroupID	This field uniquely defines an ID for a switch group of metadata elements groups.
mae_switchGroupAllowOnOff	This flag defines if the audience is allowed to completely disable the playback of the switch group. If the flag is set to zero, then one member of the group is always played back, if the flag is set to one, then either none or one member of the group is played back.
mae_switchGroupDefaultOnOff	This flag defines if the switch group is enabled or disabled for playback by default. If the flag is enabled by default, then the default member of the group is played back in the initial setting.
mae_bsSwitchGroupNumMembers	This field signals the number of members of a switch group. It can take values between 0 and 31, resulting in a maximum number of 32 members. If a group is a member of a switch group, the mae_allowOnOff field should be ignored during processing and playback.
mae_switchGroupMemberID	This field specifes the groupIDs of the members of the switch group.
mae_switchGroupDefaultGroupID	This field signals default member of the switch group (i.e. if a switch group is selected for playback by the audience, the default member is played back until the audience switches to another member).
mae_bsNumDescriptionBlocks	This field specifies the number of description blocks.
mae_descriptionGroupID	This field specifies the mae_groupID of the group to which the description block applies.
mae_descriptionSwitchGroupID	This field specifies the mae_switchGroupID of the switch group to which the description block applies.
mae_descriptionGroupPresetID	This field specifies the mae_groupPresetID of the group preset to which the description block applies.

mae_bsNumDescLanguages This field specifies the number of available languages for the description text.

mae_bsDescriptionLanguage This 24-bit field identifies the language of the description text of a metadata element group. It contains a 3-character code as specified by ISO 639-2. Both ISO 639-2/B and ISO 639-2/T may be used. Each character is coded into 8 bits according to ISO/IEC 8859-1 and inserted in order into the 24-bit field. EXAMPLE: French has 3-character code "fre", which is coded as: "0110 0110 0111 0010 0110 0101".

mae_bsDescriptionDataLength This field defines the length of the following group description in the bitstream.

mae_descriptionData This field contains a description of a metadata element group or a switch group, i.e. a string describing the content by a high-level description. The format shall follow UTF-8 according to ISO/IEC 10646.

mae_bsNumContentDataBlocks This field specifies the number of ContentData blocks.

mae_contentDataGroupID This field specifies the mae_groupID of the group to which the ContentData block applies.

mae_contentKind This field defines the kind of content of a metadata element group,

Table 243 — Value of mae_contentKind

mae_contentKind	Description
0	undefined
1	complete main
2	dialogue
3	music
4	effect
5	mixed
6	LFE
7	voiceover
8	spokensubtitle
9	audiodescription/visually impaired
10	commentary
11	hearing impaired
12	emergency
13-15	reserved

mae_hasContentLanguage This field defines if the actual metadata element group has a language assigned to its content.

mae_contentLanguage This 24-bit field identifies the language of a metadata element group. It contains a 3-character code as specified by ISO 639-2. Both ISO 639-2/B and ISO 639-2/T may be used. Each character is coded into 8 bits according to ISO/IEC 8859-1 and inserted in order into the 24-bit field. EXAMPLE: French has 3-character code “fre”, which is coded as: “0110 0110 0111 0010 0110 0101”.

mae_bsNumCompositePairs This is the number of CompositePairs.

mae_CompositeElementID This field uniquely defines an ID for a metadata element. This field can take values between 0 and 127, resulting in a maximum of 128 metadata elements. The metadata element in position 0, is an *independent* object and the metadata element in position 1 is a *dependent* object. These two objects form together the CompositePair. This dependent object will be given the metadata of the independent object. The dependent object and the independent object represent a composite object pair combining the dependent object and the independent object in time or frequency domain. Each element may be either a discrete object or an SAOC object.

mae_groupPresetID This field uniquely defines an ID for a group preset. This field can take values between 0 and 31. The group preset the lowest mae_groupPresetID value shall be selected as default.

mae_groupPresetKind This field defines the kind of content of a group preset.

Table 244 — Value of mae_groupPresetKind

mae_groupPresetKind	Description
0	undefined
1	integrated TV loudspeaker
2	high quality loudspeaker
3	mobile loudspeakers
4	mobile headphones
5	hearing impaired (light)
6	hearing impaired (heavy)
7	visually impaired / audio description
8	spoken subtitles
9	loudness/DRC
10-25	/* reserved for ISO use */
26-30	/* reserved for use outside of ISO scope */
31	other

mae_bsGroupPresetNumConditions

This field defines the number of group conditions that are associated with a group preset or a group preset extension. The field takes values between 0 and 15; a minimum of 1 condition and a maximum of 16 conditions are assumed. A condition is a combination of a mae_groupID or an mae_switchGroupID (if mae_isSwitchGroupCondition is equal to 1) and an on/off status. Use the following signaling to define a special preset or a group preset extension with no groupPresetCondition for full user interactivity:

bsGroupPresetNumConditions = 1

group preset: mae_groupPresetReferenceID = 127

group preset extension: mae_groupPresetGroupID = 127

mae_groupPresetConditionOnOff = 0

mae_groupPresetReferenceID

This field specifies the groups or switch groups associated with a group preset. By default, this reference is interpreted as a groupID. The reference can be defined to be interpreted as a switchGroupID by extension metadata.

mae_groupPresetGroupID

This field specifies the groups associated with a group preset extension.

mae_groupPresetConditionOnOff

This flag describes the required on/off status of a group or a switch group associated with a group preset or a group preset extension. If the flag is 1 and the referenced ID is defined to be interpreted as a groupID, the associated group has to be switched on to validate the group preset or group preset extension.

If the referenced ID is defined to be interpreted as a switchGroupID, a groupPresetConditionOnOff with value 1 means that one member of the switch group has to be switched on to validate the group preset If the flag is 0, the associated group or switch group has to be switched off.

mae_groupPresetDisableGainInteractivity

This field defines whether the gain interactivity of the currently referenced group or the members of the referenced switch group shall be disabled (flag is equal to 1) or shall stay enabled (flag is equal to 0) if the preset or preset extension is chosen/valid.

mae_groupPresetGainFlag

This field defines whether the corresponding preset or preset extension specifies an initial gain of the members of a metadata element group or of the element members of the group members of the referenced switch group. It shall only be 1 if the flag mae_allowGainInteractivity of the corresponding group or of all the members of the referenced switch group is set to 1.

mae_groupPresetGain

The field defines the initial gain of the members of a metadata element group or of the element members of the group members of the referenced switch group when the corresponding preset or preset extension is selected.

groupPresetGain in dB = 0.5 · (**mae_groupPresetGain** – 255) + 32

mae_groupPresetDisablePositionInteractivity

This field defines whether the position interactivity of the currently referenced group or of the members of the referenced switch group shall be disabled (flag is equal to 1) or shall stay enabled (flag is equal to 0) if the preset or preset extension is chosen/valid.

mae_groupPresetPositionFlag

This field defines whether initial position interactivity data (azimuth offset, elevation offset and distance factor) is present that shall be applied to the members of a metadata element group or to the element members of the group members of the referenced switch group. It shall only be 1 if the flag mae_allowPositionInteractivity of the corresponding group or of all the members of the referenced switch group is set to 1.

mae_groupPresetAzOffset

This field defines the additional azimuth offset that shall be applied to the currently referenced group or the members of the referenced switch group if the preset or preset extension is chosen/valid. This field can take values between PresetAdditionalAzOffset = -180° and AzOffset = +180°:

PresetAdditionalAzOffset =1.5 · (**mae_groupPresetAzOffset** – 127)

mae_groupPresetElOffset

This field defines the additional elevation offset that shall be applied to the currently referenced group or the members of the referenced switch group if the preset or preset extension is chosen/valid. This field can take values between PresetAdditionalElOffset = -90° and ElOffset = +90°:

PresetAdditionalElOffset = 3 · (**mae_groupPresetElOffset** – 31)

mae_groupPresetDistFactor

This field defines the additional distance change factor that shall be applied to the currently referenced group or the members of the referenced switch group if the preset or preset extension is chosen/valid. This field can take values between 0 and 15 resulting in PresetAdditionalDistFactor between 0.000 25 and 8:

$$PresetAdditionalDistFactor = 2^{(mae\_groupPresetDistFactor-12)}$$

hasNonStandardScreenSize

This flag specifies whether a nominal screen size is defined that is different than the standard screen size. The definition is done via viewing angles corresponding to the screen edges. In case hasNonStandardScreenSize is zero, the following values are used as default (assuming a 4k display and an optimal viewing distance):

$\varphi_{left}^{nominal} = 29.0°$

$\varphi_{right}^{nominal} = -29.0°$

$\theta_{top}^{nominal} = 17.5°$

$\theta_{bottom}^{nominal} = -17.5°$

bsScreenSizeAz

This field defines the azimuth corresponding to the left and right screen edge:

$$\varphi_{left}^{nominal} = 0.5 \cdot bsScreenSizeAz$$

$$\varphi_{left}^{nominal} = \min(\max(\varphi_{left}^{nominal}, 0), 180)$$

$$\varphi_{right}^{nominal} = -0.5 \cdot bsScreenSizeAz$$

$$\varphi_{right}^{nominal} = \min(\max(\varphi_{right}^{nominal}, -180), 0)$$

bsScreenSizeTopEl

This field defines the elevation corresponding to the top screen edge:

$$\theta_{top}^{nominal} = 0.5 \cdot (bsScreenSizeTopEl - 255)$$

$$\theta_{top}^{nominal} = \min(\max(\theta_{top}^{nominal}, -90), 90)$$

bsScreenSizeBottomEl

This field defines the elevation corresponding to the bottom screen edge:

$$\theta_{bottom}^{nominal} = 0.5 \cdot (bsScreenSizeBottomEl - 255)$$

$$\theta_{bottom}^{nominal} = \min(\max(\theta_{bottom}^{nominal}, -90), 90)$$

mae_overwriteProductionScreenSizeData

This field defines if the bitstream contains azimuth values for a non-centered default production screen. If this flag is set to 1, the following azimuth values shall be used in the processing instead of the values from mae_ProductionScreenSizeData().

bsScreenSizeLeftAz

This field defines the azimuth corresponding to the left screen edge:

$\varphi_{left}^{nominal} = 0.5 \cdot (bsScreenSizeLeftAz - 511)$

$\varphi_{left}^{nominal} = \min(\max(\varphi_{left}^{nominal}, -180), 180)$

bsScreenSizeRightAz

This field defines the azimuth corresponding to the right screen edge:

$\varphi_{right}^{nominal} = 0.5 \cdot (bsScreenSizeRightAz - 511)$

$\varphi_{right}^{nominal} = \min(\max(\varphi_{right}^{nominal}, -180), 180)$

mae_NumPresetProductionScreens

This field defines the number of preset-associated production screens.

mae_productionScreenGroupPresetID

This field defines the presetID the current production screen is associated with.

mae_hasNonStandardScreenSize

This field defines if the bitstream contains a non-standard preset-associated production screen size. If the flag is one, the non-standard production screen size information follows in the bitstream.

isCenteredInAzimuth

This flag defines whether the production screen is centered in azimuth (absolute values of the azimuth angles of the left and right screen edge are identical) or not.

mae_hasSwitchGroupCondition

This flag defines whether a group preset has switch group conditions (flag is equal to 1).

mae_isSwitchGroupCondition

This field defines whether the condition from original a preset definition or a group preset extension references a groupID (mae_isSwitchGroupCondition is equal to 0) or if the referenced ID shall be interpreted as a switchGroupID (mae_isSwitchGroupCondition is equal to 1).

mae_hasDownmixIdGroupPresetExtensions

This flag defines whether a group preset has layout-dependent extensions (flag is equal to 1). Group presets can be extended by group presets extensions, which are applicable for a specific downmixId. These extensions can overwrite the preset conditions and other preset characteristics, e.g. the group preset gain. If a preset is selected and there is a current downmixId, the conditions and characteristics of the appropriate group preset extension shall replace the corresponding values from the chosen/valid group preset in the processing and rendering.

mae_groupPresetDownmixId	This field references a downmixId, for which the current group preset extension is applicable**.**
mae_groupPresetSwitchGroupID	This field specifies the switch groups associated with a group preset extension.
mae_loudnessCompGroupLoudnessPresent	A field that indicates whether group loudness values for loudness compensation follow in the bitstream.
mae_bsLoudnessCompGroupLoudness	A field that signals a loudness value for the current metadata element group (groupID). loudnessCompGroupLoudness in dB = 0.25 · **mae_bsLoudnessCompGroupLoudness** - 57.75
mae_loudnessCompDefaultParamsPresent	A field that indicates whether loudness compensation parameters for the default scene follow in the bitstream. If not present, all metadata element groups shall be incorporated in the computation of the loudness compensation gain.
mae_loudnessCompDefaultIncludeGroup	A field that signals whether the current metadata element group (groupID) shall be incorporated in the computation of the loudness compensation gain of the default scene.
mae_loudnessCompDefaultMinMaxGainPresent	A field that indicates whether min/max values for loudness compensation gain of the default scene follow in the bitstream.
mae_bsLoudnessCompDefaultMinGain	A field that signals a minimum value for the loudness compensation gain of the default scene. It can take values between 0 and minus 42 dB in 3 dB steps. A value of minus infinity can be signalled by the largest number of mae_bsLoudnessCompDefaultMinGain (15). If not present the default value is minus infinity. loudnessCompDefaultMinGain in dB = -3 · **mae_bsLoudnessCompDefaultMinGain**
mae_bsLoudnessCompDefaultMaxGain	A field that signals a maximum value for the loudness compensation gain of the default scene. It can take values between 0 and 45 dB in 3 dB steps. If not present the default value is plus 21 dB. loudnessCompDefaultMaxGain in dB = 3 · **mae_bsLoudnessCompDefaultMaxGain**
mae_loudnessCompPresetParamsPresent	A field that indicates whether loudness compensation parameters for the current preset (groupPresetID) follow in the bitstream. If not present, all metadata element groups shall be incorporated in the computation of the loudness compensation gain.

mae_loudnessCompPresetIncludeGroup	A field that signals whether the current metadata element group (groupID) shall be incorporated in the computation of the loudness compensation gain of the current preset (groupPresetID).
mae_loudnessCompPresetMinMaxGainPresent	A field that indicates whether min/max values for loudness compensation gain of the current preset (groupPresetID) follow in the bitstream.
mae_loudnessCompPresetMinGain	A field that signals a minimum value for the loudness compensation gain of the current preset (groupPresetID). It can take values between 0 and minus 42 dB in 3 dB steps. A value of minus infinity can be signalled by the largest number of mae_bsLoudnessCompPresetMinGain (15). If not present the default value is minus infinity. loudnessCompPresetMinGain in dB = -3 · **mae_bsLoudnessCompPresetMinGain**
mae_loudnessCompPresetMaxGain	A field that signals a maximum value for the loudness compensation gain of the current preset (groupPresetID). It can take values between 0 and 45 dB in 3 dB steps. If not present the default value is plus 21 dB. loudnessCompPresetMaxGain in dB = 3 · **mae_bsLoudnessCompPresetMaxGain**
mae_metaDataElementIDmaxAvail	This field signals the maximum available mae_metaDataElementID in a Main Stream or Sub-Stream.
version	A version field that shall be set to zero.
bsNumTargetLoudnessConditions	A field that signals the number of target loudness conditions. The field can take values between 0 and 7. numTargetLoudnessConditions = **bsNumTargetLoudnessConditions** + 1
bsTargetLoudnessValueUpper	A field that signals the upper limit of the target loudness range defined by targetLoudnessValueUpper/-Lower. The field can take values between -63 and 0 dB. A range check shall include the upper boundary value and exclude the lower boundary value. If the requested decoder target loudness (targetLoudness) includes any of the signalled ranges, the corresponding drcSetEffectAvailable field can, e.g. be used for guidance of a user interface if the content of the DRC bitstream is not available. targetLoudnessValueUpper in dB = **bsTargetLoudnessValueUpper** – 63

drcSetEffectAvailable	This field signals the available DRC set effects in the audio bitstream for a certain target loudness range. Each bit represents a requestable DRC set effect type, where ISO/IEC 23003-4:2015, Table 11 defines bit positions by index (0 corresponds to LSB) and the corresponding semantics. If not present, all bits shall be set to one. Note that the DRC set selection process according to Clause 6.4.4 can handle any request independent of availability in the bitstream.

15.4 Definition of mae_metaDataElementIDs

The definition of metadata groups (mae_GroupDefinition) references signals via the identifiers mae_metaDataElementID. These identifiers are calculated using the following pseudo code.

```
mae_metaDataElementID = mae_metaDataElementIDoffset;
for ( grp = 0; grp < bsNumSignalGroups + 1; grp++) {
    if ( SignalGroupType[grp] == SignalGroupTypeChannels ) {
        for ( id = 0; id < bsNumberOfSignals[grp] + 1; id++ )  {
            mae_metaDataElementID++;
        }
   }
   else if ( SignalGroupType[grp] == SignalGroupTypeObject ) {
        for ( id = 0; id < bsNumberOfSignals[grp] + 1; id++ )  {
            mae_metaDataElementID++;
        }
   }
    else if ( SignalGroupType[grp] == SignalGroupTypeSAOC ) {
      for ( id = 0; id < numSpeakers + bsNumSaocObjects; id++ )  {
         mae_metaDataElementID++;
      }
      if(saocDmxLayoutPresent == 1) {
         mae_metaDataElementID++;
      }
    }
    else if ( SignalGroupType[grp]  == SignalGroupTypeHOA ) {
        mae_metaDataElementID++;
    }
}
```

with `numSpeakers` as used when evaluating SAOC3DgetNumChannels(saocChannelLayout).

If saocDmxLayoutPresent == 1 the MAE information shall contain one additional group with exactly one data element. The additional group shall be located after the last group containing elements of signal type SAOC channels or SAOC objects. The data element in the additional group shall be associated with the complete group of SAOC Transport Channels. Additionally, if saocDmxLayoutPresent == 1, the value of the variable baseChannelCount, defined in subclause 6.3.3, shall be increased by 1, such that an individual DRC gain sequence can be assigned to the complete group of SAOC transport channels.

15.5 Loudness compensation after gain interactivity

An important feature of MPEG-H 3D audio is the support of user interaction at the decoder: The user can, e.g. adjust the volume of metadata element groups or even switch them on and off. Changing the level of groups also implies that the overall loudness of the rendered audio scene is changed compared to the unmodified case.

The loudness normalization feature as specified in subclause 6.4.7, applies to unmodified reference scenes dependent on the selected preset, target layout or DRC configuration. The loudness compensation tool specified in this sub-clause operates in addition to loudness normalization according to subclause 6.4.7 and compensates for any loudness change relative to the unmodified reference scene due to user interaction with the reference scene.

If the structure mae_LoudnessCompensationData() is present in the metadata bitstream (mae_dataType == ID_MAE_LOUDNESS_COMPENSATION), the loudness compensation tool shall be enabled. If not present, the loudness compensation tool shall be disabled by default.

If the loudness compensation tool is enabled, a loudness compensation gain shall be computed after any gain interaction or preset selection according to Table 245. The computed compensation gain shall be applied to each audio element. The input parameters for the computation of the loudness compensation gain are extracted as specified in Table 246.

Group loudness values can be either transmitted within mae_LoudnessCompensationData() by using mae_loudnessCompGroupLoudnessPresent==1 (see subclause 15.2) or within mpegh3aLoudnessInfoSet() by using loudnessInfoType==1 (see subclause 6.3). If group loudness values for one or more metadata element groups are missing within mpegh3aLoudnessInfoSet(), the loudness compensation gain shall be computed as listed in Table 245 for groupLoudnessValueMissingFlag==1.

Table 245 — Pseudo code for computation of loudness compensation gain

```
computeLoudnessCompensationGain( numGroups,
                                 includeGroup[],
                                 groupLoudnessValueMissingFlag,
                                 groupLoudness[],
                                 groupGainDefaultDb[],
                                 groupGainInteractivityDb[],
                                 groupStateDefault[],
                                 groupStateInteractivity[],
                                 minGainDb,
                                 maxGainDb)
{
     /* init */
     loudnessReference = 0;
     loudnessAfterInteract = 0;

     /* compute components of loudness compensation gain */
     for (n=0; n<numGroups; n++) {
       if (groupLoudnessValueMissingFlag == 0) {
           tmp1 = pow(10, (groupGainDefaultDb[n] + groupLoudness[n]) / 10.0);
           tmp2 = pow(10, (groupGainInteractivityDb[n] + groupLoudness[n]) / 10.0);
       } else { /* group loudness value missing for one or more groups */
           tmp1 = pow(10, groupGainDefaultDb[n] / 10.0);
           tmp2 = pow(10, groupGainInteractivityDb[n] / 10.0);
       }
       loudnessReference += includeGroup[n]*groupStateDefault[n] * tmp1;
```

```
        loudnessAfterInteract += includeGroup[n]*groupStateInteractivity[n] * tmp2;
    }

    /* loudness compensation gain in dB */
    loudnessCompensationGainDb = 10 * log10(
                                    loudnessReference / loudnessAfterInteract);

    /* clip loudness compensation gain to min/max gain */
    if (loudnessCompensationGainDb < minGainDb) {
        loudnessCompensationGainDb = minGainDb;
    }
    if (loudnessCompensationGainDb > maxGainDb) {
        loudnessCompensationGainDb = maxGainDb;
    }

    return loudnessCompensationGainDb;
}
```

Table 246 — Input parameters for computation of loudness compensation gain

Input parameters	**Default scene** (no preset selected)	**Preset scene** (groupPresetID selected, gp is the index of the selected preset)
numGroups	mae_numGroups	mae_numGroups
includeGroup[]	mae_loudnessCompDefault-IncludeGroup[]	mae_loudnessCompPreset-IncludeGroup[gp][]
groupLoudness[]	loudnessCompGroupLoudness[] if present; alternatively, extracted from mpegh3daLoudnessInfoSet().	loudnessCompGroupLoudness[] if present; alternatively, extracted from mpegh3daLoudnessInfoSet().
groupGainDefaultDb[]	0 dB for all groups.	If present, groupPresetGain[gp][] and otherwise 0 dB for all groups.
groupGainInteractivityDb[]	Current interactivity gain in dB.	Current interactivity gain in dB.
groupStateDefault[]	“1” for groups that are switched on in the default scene, “0” for groups that are switched off.	“1” for groups that are switched on in the preset definition, “0” for groups that are switched off. For groups that are not explicitly referenced in the preset definition, the respective state of the default scene applies.
groupStateInteractivity[]	“1” for groups that are switched on after interactivity, “0” for groups that are switched off.	“1” for groups that are switched on after interactivity, “0” for groups that are switched off.
minGain	loudnessCompDefaultMinGain	loudnessCompPresetMinGain[gp]
maxGain	loudnessCompDefaultMaxGain	loudnessCompPresetMaxGain[gp]

16 Loudspeaker distance compensation

If no external loudspeaker compensation is conducted then loudspeaker distance compensation processing shall be applied as part of the loudspeaker feed processing chain. Loudspeaker distance compensation processing performs distance compensation if the distances of the local loudspeakers from the listening area ("sweet spot") differ from one another. The processing applies trim parameters (compensation gains as well as compensation delays) to counter the effects of non-uniform loudspeaker distances.

The trim parameters:

$T_{d,A}$ trim delay in samples for each output channel A;

$T_{g,A}$ trim gain (linear gain value) for each output channel A;

shall be computed as follows as a function of the loudspeaker distances in $trim_A$, the distances given in metres:

$$T_{d,A} = \text{round}\left(\frac{\max_n trim_n - trim_A}{340 / f_s} \right) \text{ with output sampling rate } f_s \text{ in Hz}$$

$$T_{g,A} = \frac{\sqrt{trim_A}}{\sqrt{\max_n trim_n}}$$

Note that $\max_n trim_n$ denotes the distance of the loudspeaker located farest away from the sweet spot. Thus, all trim delays are non-negative and no trim gain exceeds the neutral gain value 1.0.

If the distances of the loudspeakers of the reproduction setup are non-uniform, the derived trim parameters shall be applied accordingly: The loudspeaker signal of output channel A shall be delayed by $T_{d,A}$ time domain samples and the signal shall also be multiplied by the linear gain $T_{g,A}$.

Besides the distance compensation, the loudspeaker calibration gains shall be applied here.

17 Interfaces to the MPEG-H 3D audio decoder

17.1 General

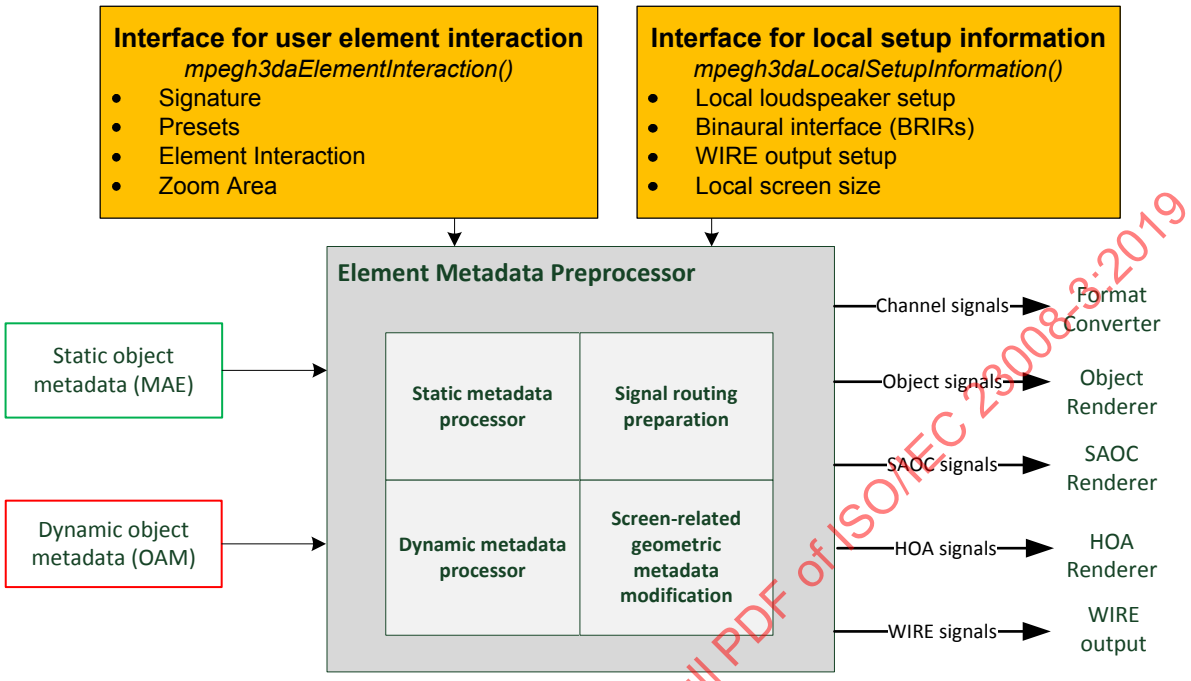


Figure 99 — Interfaces to MPEG-H 3D audio decoder

17.2 Interface for local setup information

17.2.1 General

The mpegh3daLocalSetupInformation() syntax element defines the formatting of local setup information for the rendering. It defines the rendering type and the screen information if available. Two different rendering types are defined: loudspeaker rendering and binaural rendering.

17.2.2 WIRE output

In addition to the signaling of the target layout or the binaural configuration, it is also possible to signal WIRE outputs. WIRE outputs are arbitrary output devices (e.g. headphone outputs or auxiliary loudspeakers) that exist in addition to the regular playback device. The output after the decoder shall first contain the regular loudspeaker signals followed by the WIRE outputs. A WIRE output can be single channel or multichannel.

The element groups in a WIRE output should be processed with the same loudness/DRC processing as if they are regular loudspeaker signals. The application of a peak limiter at the end of the processing chain is highly recommended.

Only channels or objects can be routed to WIRE outputs.

17.2.3 Syntax for local setup information

Table 247 — Syntax for mpegh3daLocalSetupInformation()

Syntax	No. of bits	Mnemonic
mpegh3daLocalSetupInformation()		
{		
bsRenderingType;	**1**	**uimsbf**
switch (bsRenderingType) {		
case 0:		
LoudspeakerRendering();		
case 1:		
BinauralRendering();		
}		
bsNumWIREoutputs;	**16**	**uimsbf**
if (bsNumWIREoutputs > 0) {		
for (n = 0; n < bsNumWIREoutputs; n++) {		
WireID[n];	**16**	**uimsbf**
}		
}		
hasLocalScreenSizeInformation;	**1**	**bslbf**
if (hasLocalScreenSizeInformation) {		
LocalScreenSizeInformation();		
}		
}		

17.2.4 Semantics for local setup information

bsRenderingType Defines the type of rendering (loudspeaker rendering or binaural rendering).

bsNumWIREoutputs This field defines the number of WIRE outputs. A maximum of 65535 WIRE outputs is assumed; therefore the field can take values between 0 and 65535.

WireID This field contains an identifier for a WIRE output.

hasLocalScreenSizeInformation Flag that defines if information about the local screen size is available. If yes, the local screen size information follows afterwards.

17.3 Interface for local loudspeaker setup and rendering

17.3.1 General

The LoudspeakerRendering() syntax element defines the signalling of the local loudspeaker setup and target layout.

The target layout for loudspeaker reproduction is signalled using the syntax element SpeakerConfig3D() (see subclause 5.2.2.2). The target layout can be given by either

— a CICPspeakerLayoutIdx (speakerLayoutType = 0), or

— a list of CICPspeakerIdx (speakerLayoutType = 1), or

— a mixed list of loudspeaker positions in terms of azimuth and elevation and CICPspeakerIdx (speakerLayoutType = 2).

In case speakerLayoutType=0 or speakerLayoutType=1 is used, the exact position may be signalled in addition for each loudspeaker. In this way it is possible to signal standard loudspeaker layouts with individual adjustments to the precise loudspeaker positions (loudspeaker misplacement). In this case, the decoder is able to interpret and make use of the implicit semantic information about the setup.

The format converter provides optimized behaviour if a standard setup is provided along with additional offset values. The format converter accepts offset values from the nominal loudspeaker positions as an input parameter (see subclause 10.3.4). In case speakerLayoutType<=1 is used and the exact position information is transmitted, then the exact position information shall be used to calculate the offsets from the nominal loudspeaker positions. These offsets for azimuth and elevation angles are defined as the difference between signalled position and nominal position. The nominal positions are defined in ISO/IEC 23001-8.

If speakerLayoutType=0 (CICPspeakerLayoutIdx) is used, the order of channels is implicitly given and the order of output channels then follows the order of channels in Table 163. If the channel order of a standard setup is permuted (with respect to the order given in Table 163), then speakerLayoutType=1 shall be used for signalling, with a different order of CICPspeakerIdx than in Table 163. Signalling of speakerLayoutType=1 shall also be used for setups that are not standard layouts but consist of standard loudspeakers. Completely arbitrary setups shall be signalled by speakerLayoutType=2.

If speakerLayoutType=1 or speakerLayoutType=2 is used, the output of the decoder shall follow the order given in the LoudspeakerRendering() syntax element.

In addition to the layout information, a distance to the reference point ('sweet spot') is defined for each loudspeaker which shall be applied by the renderer in the loudspeaker distance compensation processing (see Clause 16).

Further, a loudspeaker calibration gain value is also defined for each loudspeaker which shall be applied by the renderer in the loudspeaker compensation processing (see Clause 16).

17.3.2 Syntax for local loudspeaker signalling

Table 248 — Syntax for LoudspeakerRendering()

Syntax	No. of bits	Mnemonic
LoudspeakerRendering()		
{		
bsNumLoudspeakers;	**16**	**uimsbf**
targetLayout = SpeakerConfig3d();		
hasLoudspeakerDistance;	**1**	**bslbf**
hasLoudspeakerCalibrationGain;	**1**	**bslbf**
useTrackingMode;	**1**	**bslbf**
for (n = 0; n < bsNumLoudspeakers; n++) {		
if (speakerLayoutType <= 1) {		
hasKnownPosition[n];	**1**	**bslbf**
if (hasKnownPosition[n]) {		
loudspeakerAzimuth[n];	**9**	**uimsbf**
loudspeakerElevation[n];	**8**	**uimsbf**
}		
}		

Syntax	No. of bits	Mnemonic
if (hasLoudspeakerDistance) {		
loudspeakerDistance[n];	**10**	**uimsbf**
}		
if (hasLoudspeakerCalibrationGain) {		
loudspeakerCalibrationGain[n];	**7**	**uimsbf**
}		
}		
externalDistanceCompensation;	**1**	**bslbf**
}		

17.3.3 Semantics for local loudspeaker signalling

bsNumLoudspeakers Number of loudspeakers in the local reproduction setup.

hasLoudspeakerDistance This field defines if a loudspeaker distance is given in the bitstream.

HasLoudspeakerCalibrationGain This field defines if a loudspeaker calibration gain is given in the bitstream.

useTrackingMode This field defines if a processing of scene displacement values sent via the mpegh3daSceneDisplacementData() interface shall happen or not.

hasKnownPosition This flag defines if an explicit signaling of the known position of the loudspeaker is following in the bitstream.

loudspeakerAzimuth This field defines the azimuth angle of a loudspeaker. It can take values between -180° and 180° in 1° steps.

Azimuth = (loudspeakerAzimuth – 256)

Azimuth = min (max (Azimuth, -180), 180)

loudspeakerElevation This field defines the elevation angle of a loudspeaker. It can take values between -90° and 90° in 1° steps.

Elevation = (loudspeakerElevation – 128)

Elevation= min (max (Elevation, -90), 90)

loudspeakerDistance This field defines the distance of a loudspeaker to the reference point centered in the loudspeaker setup in cm. The field can take values between 1 and 1 023, corresponding to 1 to 1 023 cm in 1 cm steps.

loudspeakerCalibrationGain This field defines a loudspeaker calibration gain in dB. The field can take values between 0 and 127, corresponding to dB values between Gain = -32 dB and Gain = 31,5 dB in 0.5 dB steps:

Gain [dB] = 0,5 · (loudspeakerGain – 64)

externalDistanceCompensation This flag defines whether loudspeaker compensation shall be a applied to the decoder output signals. If this flag is 1, the loudspeaker compensation shall be omitted and loudspeakerDistance and loudspeakerCalibrationGain shall not be applied in the decoder.

17.4 Interface for binaural room impulse responses (BRIRs)

17.4.1 General

The BRIR/HRTF data for the binaural rendering may be provided to the decoder by using the syntax element BinauralRendering(). It supports the description of:

— raw FIR filters (for both virtual loudspeakers and HOA-to-binaural approaches);

— low-level parameters associated to the time-domain binaural renderer (for both virtual loudspeakers and HOA-to-binaural approaches) and the frequency-domain binaural renderer.

In case the impulse responses are described as virtual loudspeaker filters, the measurement setup of the impulse responses is defined, i.e. the positions of loudspeakers during the measurement procedure or sound sources in a room simulation.

The measurement setup of the selected impulse response set is signalled using the bitstream syntax element SpeakerConfig3d(). In case a CICPspeakerLayoutIdx is used for signalling the measurement setup, the order of the BRIR pairs has to follow the order of channels in Table **163**; otherwise speakerLayoutType=1 or speakerLayoutType=2 shall be used to signal a different BRIR order.

The binaural data should contain at least one representation that is supported by the binaural renderer, otherwise no output is provided.

Within the BinauralRendering() structure no more than one of the signalled sets of impulse responses shall be applicable to each of the three binaural rendering modes. When performing binauralization the choice of binaural rendering mode determines the BRIR set used.

17.4.2 Syntax of binaural renderer interface

Table 249 — Syntax of BinauralRendering()

Syntax	No. of bits	Mnemonic
BinauralRendering()		
{		
bsFileSignature;	**32**	**bslbf**
bsFileVersion;	**8**	**uimsbf**
bsNumCharName;	**8**	**uimsbf**
for (i=0; i<bsNumCharName; i++) {		
bsName[i];	**8**	**bslbf**
}		
useTrackingMode;	**1**	**bslbf**
bsNumBinauralDataRepresentation;	**4**	**uimsbf**
for (r = 0; r < bsNumBinauralDataRepresentation; r++) {		
brirSamplingFrequencyIndex;	**5**	**uimsbf**
if (brirSamplingFrequencyIndex == 0x1f) {		
brirSamplingFrequency;	**24**	**uimsbf**
}		
isHoaData;	**1**	**bslbf**

Syntax	No. of bits	Mnemonic
if (isHoaData) {		
hoaOrderBinaural = escapedValue(3,5,0);	**3,8**	**uimsbf**
nBrirPairs = (hoaOrderBinaural+1)^2;		
} else {		
MeasurementSetup = SpeakerConfig3d();		
if (speakerLayoutType == 0) { /* See SpeakerConfig3d() */		
nBrirPairs = escapedValue(5,8,0)+1;	**5,13**	**uimsbf**
} else {		
nBrirPairs = numSpeakers; /* See SpeakerConfig3d() */		
}		
}		
bsBinauralDataFormatID;	**2**	**uimsbf**
ByteAlign();		
switch (bsBinauralDataFormatID) {		
case 0:		
BinauralFIRData();		
break;		
case 1:		
FdBinauralRendererParam();		
break;		
case 2:		
TdBinauralRendererParam();		
break;		
}		
}		
}		

Table 250 — Syntax of BinauralFirData()

Syntax	No. of bits	Mnemonic
BinauralFirData()		
{		
bsNumCoefs;	**24**	**uimsbf**
for (pos = 0; pos < nBrirPairs; pos++) {		
for (i = 0; i < bsNumCoefs; i++) {		
bsFirCoefLeft[pos][i];	**32**	**bslbf**
bsFirCoefRight[pos][i];	**32**	**bslbf**
}		
}		
bsAllCutFreq;	**32**	**bslbf**
if (bsAllCutFreq == 0) {		
for (pos = 0; pos < nBrirPairs; pos++) {		
bsCutFreqLeft[pos];	**32**	**bslbf**
bsCutFreqRight[pos];	**32**	**bslbf**
}		
} else {		
for (pos = 0; pos < nBrirPairs; pos++) {		
bsCutFreqLeft [pos] = bsAllCutFreq;		
bsCutFreqRight [pos] = bsAllCutFreq;		
}		
}		
}		

Table 251 — Syntax of FdBinauralRendererParam()

Syntax	No. of bits	Mnemonic
FdBinauralRendererParam()		
{		
flagHrir;	**1**	**bslbf**
dInit;	**10**	**uimsbf**
kMax;	**6**	**uimsbf**
kConv;	**6**	**uimsbf**
kAna;	**6**	**uimsbf**
VoffBrirParam();		
if (flagHrir==0) {		
SfrBrirParam();		
}		
QtdlBrirParam();		
}		

Table 252 — Syntax of VoffBrirParam()

Syntax	No. of bits	Mnemonic
VoffBrirParam()		
{		
nBitNFilter;	**4**	**uimsbf**
nBitNFft;	**3**	**uimsbf**
nBitNBlk;	**3**	**uimsbf**
for (k=0; k<kMax ; k++) {		
nFilter[k];	**nBitNFilter**	**uimsbf**
nFft[k];	**nBitNFft**	**uimsbf**
fftLength = pow(2, nFft[k]);		
nBlk[k];	**nBitNBlk**	**uimsbf**
for (b=0; b<nBlk[k]; b++) {		
for (nr=0; nr< nBrirPairs; nr++) {		
for (v=0; v<fftLength; v++) {		
VoffCoeffLeftReal[k][b][nr][v];	**32**	**bslbf**
VoffCoeffLeftImag[k][b][nr][v];	**32**	**bslbf**
VoffCoeffRightReal[k][b][nr][v];	**32**	**bslbf**
VoffCoeffRightImag[k][b][nr][v];	**32**	**bslbf**
}		
}		
}		
}		
}		

Table 253 — Syntax of SfrBrirParameter()

Syntax	No. of bits	Mnemonic
SfrBrirParam()		
{		
for (k=0; k<kAna ; k++) {		
fcAna[k];	**32**	**bslbf**
rt60[k];	**32**	**bslbf**
nrgLr[k];	**32**	**bslbf**
}		
}		

Table 254 — Syntax of QtdlBrirParameter()

Syntax	No. of bits	Mnemonic
QtdlBrirParam()		
{		
for (k=0; k<kMax-kConv ; k++) {		
nBitQtdlLag[k]**;**	**4**	**uimsbf**
for (nr=0; nr< nBrirPairs; nr++) {		
QtdlGainLeftReal[k][nr]**;**	**32**	**bslbf**
QtdlGainLeftImag[k][nr]**;**	**32**	**bslbf**
QtdlGainRightReal[k][nr]**;**	**32**	**bslbf**
QtdlGainRightImag[k][nr]**;**	**32**	**bslbf**
QtdlLagLeft[k][nr]**;**	**nBitQtdlLag**[k]	**uimsbf**
QtdlLagRight[k][nr]**;**	**nBitQtdlLag**[k]	**uimsbf**
}		
}		
}		

Table 255 — Syntax of TdBinauralRendererParam()

Syntax	No. of bits	Mnemonic
TdBinauralRendererParam()		
{		
bsDelay;	**16**	**uimsbf**
bsDirectLen;	**16**	**uimsbf**
bsNbDiffuseBlocks;	**8**	**uimsbf**
for (chan=0; chan< nBrirPairs; chan++) {		
bsFmaxDirectLeft[chan];	**32**	**bslbf**
bsFmaxDirectRight[chan];	**32**	**bslbf**
}		
for (block=0; block< bsNbDiffuseBlocks; block++) {		
bsFmaxDiffuseLeft[block];	**32**	**bslbf**
bsFmaxDiffuseRight[block];	**32**	**bslbf**
}		
for (chan=0; chan< nBrirPairs; chan++) {		
bsWeights[chan];	**32**	**bslbf**
}		
for (chan=0; chan< nBrirPairs; chan++) {		
for (tap=0; tap< bsDirectLen; tap++) {		
bsFIRDirectLeft[chan][tap];	**32**	**bslbf**
bsFIRDirectRight[chan][tap];	**32**	**bslbf**
}		
}		
for (bloc=0; bloc< bsNbDiffuseBlocks; bloc++) {		
for (tap=0; tap< bsDirectLen; tap++) {		
bsFIRDiffuseLeft[bloc][tap];	**32**	**bslbf**
bsFIRDiffuseRight[bloc][tap];	**32**	**bslbf**
}		
}		
}		

17.4.3 Semantics

bsFileSignature A string of 4 ASCII characters that reads “BRIR”.

bsFileVersion File version indication. This document describes version 1 of the file format.

bsNumCharName Number of ASCII characters in the BRIR name.

bsName BRIR name.

useTrackingMode This flag defines if a tracker device is connected and the binaural rendering shall be processed in a special headtracking mode, meaning a processing of scene displacement values sent via the mpegh3daSceneDisplacementData() interface shall happen.

bsNumBinauralDataRepresentation Total number of representations of the filters.

brirSamplingFrequencyIndex This index determines the sampling frequency of the BRIRs. The value of brirSamplingFrequencyIndex and their associated sampling frequencies are specified in Table 256.

Table 256 — Value and meaning of brirSamplingFrequencyIndex

brirSamplingFrequencyIndex	Sampling frequency
0x00	96 000
0x01	88 200
0x02	64 000
0x03	48 000
0x04	44 100
0x05	32 000
0x06	24 000
0x07	22 050
0x08	16 000
0x09	12 000
0x0a	11 025
0x0b	8 000
0x0c	7 350
0x0d	reserved
0x0e	reserved
0x0f	57 600
0x10	51 200
0x11	40 000
0x12	38 400
0x13	34 150
0x14	28 800
0x15	25 600
0x16	20 000
0x17	19 200
0x18	17 075
0x19	14 400
0x1a	12 800
0x1b	9 600
0x1c	reserved

brirSamplingFrequencyIndex	Sampling frequency
0x1d	reserved
0x1e	reserved
0x1f	escape value
NOTE The values of **brirSamplingFrequencyIndex** 0x00 up to 0x0e are identical to those of the **samplingFrequencyIndex** 0x0 up to 0xe contained in the AudioSpecificConfig() specified in ISO/IEC 14496-3:2009.	

brirSamplingFrequency BRIR sampling frequency as unsigned integer value in case brirSamplingFrequencyIndex equals zero.

isHoaData Set to 1, this element indicates that corresponding BRIR coefficients are represented in the spherical harmonics (HOA) domain. Otherwise corresponding BRIR coefficients are represented in the virtual loudspeaker domain.

hoaOrderBinaural This element determines the HOA order of the filter dataset. The ordering of components shall be in accordance with the "Single Index Designation (SID)" as defined in ISO/IEC 14496-11.

nBrirPairs The number of HRTF pairs that are transmitted, i.e. virtual loudspeaker positions or number of channels or number of HOA components to be filtered by a pair of binaural filters.

bsBinauralDataFormatID Indicates the representation type of the BRIR set (FIR, FD parameterized or TD parameterized).

bsNumCoefs Indicates the number of coefficients of a FIR filter.

pos BRIR index ranged from 0 to nBrirPairs-1.

i Sample index ranged from 0 to bsNumCoefs-1.

bsFirCoefLeft FIR coefficient for left ear (interpreted as float in IEEE 754 format).

bsFirCoefRight FIR coefficient for right ear (interpreted as float in IEEE 754 format).

bsAllCutFreq if non-zero, the frequency above which left and right filters values can be neglected are the same for all filters and this value is bsAllCutFreq.

bsCutFreqLeft Frequency above which left filter value can be neglected.

bsCutFreqRight Frequency above which right filter value can be neglected.

nr BRIR order index ranged from 0 to nBrirPairs-1.

flagHrir Indicates whether inputted impulse response is HRIR or BRIR.

Table 257 — Value of flagHrir

flagHrir	Meaning
0	BRIR, the sparse reverberator is switched on
1	HRIR, the sparse reverberator is switched off

dInit	Value of the propagation time.
kMax	The maximum processing band.
kConv	Number of bands used for convolution.
kAna	Number of analysis bands used in analysis of late reverberation analysis.
nBitNFilter	Number of bits for nFilter.
nBitNFft	Number of bits for nFftr.
nBitNBlk	Number of bits for n_block.
k	Band index.
fftLength	Length of VOFF Coefficient.
b	Block index.
v	VOFF coefficient index.
nFilter[k]	Filter length per band for VOFF.
nFft[k]	The length of the FFT for each band is expressed as a power of 2, where nFft[k] is the exponent. i.e. $2^{nFft[k]}$ = FFT length per band for VOFF.
nBlk[k]	Number of blocks per band for VOFF.
VoffCoeffLeftReal[k][b][nr][v]	Real values of left VOFF coefficients (interpreted as float in IEEE 754 format).
VoffCoeffLeftImag[k][b][nr][v]	Imaginary values of left VOFF coefficients (interpreted as float in IEEE 754 format).
VoffCoeffRightReal[k][b][nr][v]	Real values of right VOFF coefficients (interpreted as float in IEEE 754 format).
VoffCoeffRightImag[k][b][nr][v]	Imaginary values of right VOFF coefficients (interpreted as float in IEEE 754 format).
fcAna[k]	Centre frequencies of the late reverberation analysis frequency bands (interpreted as float in IEEE 754 format).
RT60[k]	Reverberation times RT60 in seconds of the late reverberation in the late reverberation analysis bands (interpreted as float in IEEE 754 format).
NRG[k]	Energy values that represent the energy (amplitude to the power of two) of the late reverberation part of one BRIR in the late reverberation analysis bands (interpreted as float in IEEE 754 format).

nBitQtdlLag[k]	Number of bits used for convolution.
QtdlGainLeftReal[k][nr]	Real values of left QTDL gains(interpreted as float in IEEE 754 format).
QtdlGainLeftImag[k][nr]	Imaginary values of left QTDL gains(interpreted as float in IEEE 754 format).
QtdlGainRightReal[k][nr]	Real values of right QTDL gains(interpreted as float in IEEE 754 format).
QtdlGainRightImag[k][nr]	Imagniary values of right QTDL gains(interpreted as float in IEEE 754 format).
QtdlLagLeft[k][nr][l]	Left lag values in samples for QTDL.
QtdlLagRight[k][nr][l]	Right lag values in samples for QTDL.
chan	index of channel.
block	index of diffuse block.
tap	index of filter tap.
bsDelay	delay in sample to apply at the beginning of output signals (to compensate BRIRs propagation time removed at parameterization).
bsDirectLen	size in samples of the direct part of parameterized BRIRs.
bsNbDiffuseBlocks	number of blocks in diffuse part of parameterized BRIRs.
bsFmaxDirectLeft	cut off frequency of left direct part, given by a number between 0 and 1, where 1 corresponds to the Nyquist frequency (interpreted as float in IEEE 754 format).
bsFmaxDirectRight	cut off frequency of right direct part, given by a number between 0 and 1, where 1 corresponds to the Nyquist frequency (interpreted as float in IEEE 754 format).
bsFmaxDiffuseLeft	cut off frequency of left diffuse part, given by a number between 0 and 1, where 1 corresponds to the Nyquist frequency (interpreted as float in IEEE 754 format).
bsFmaxDiffuseRight	cut off frequency of right diffuse part, given by a number between 0 and 1, where 1 corresponds to the Nyquist frequency (interpreted as float in IEEE 754 format).
bsWeights	gain to apply to the input channels before filtering with diffuse part of the impulse response (interpreted as float in IEEE 754 format).
bsFIRDirectLeft	FIR coefficients of direct part of left parameterized BRIRs (interpreted as float in IEEE 754 format).
bsFIRDirectRight	FIR coefficients of direct part of right parameterized BRIRs (interpreted as float in IEEE 754 format).
bsFIRDiffuseLeft	FIR coefficients of diffuse part of left parameterized BRIRs (interpreted as float in IEEE 754 format).
bsFIRDiffuseRight	FIR coefficients of diffuse part of right parameterized BRIRs (interpreted as float in IEEE 754 format).

17.5 Interface for local screen size information

17.5.1 General

In both rendering cases (binaural rendering or loudspeaker rendering) local screen size information can be signalled in the mpegh3daLocalSetupInformation() syntax element. This data shall be used for screen-related element remapping and zooming in the decoder if screen-related elements are present in the audio scene.

17.5.2 Syntax

Table 258 — Syntax of LocalScreenSizeInformation()

Syntax	No. of bits	Mnemonic
LocalScreenSizeInformation()		
{		
isCenteredInAzimuth;	**1**	**bslbf**
if (isCenteredInAzimuth) {		
bsLocalScreenSizeAz;	**9**	**uimsbf**
} else {		
bsLocalScreenSizeLeftAz;	**10**	**uimsbf**
bsLocalScreenSizeRightAz;	**10**	**uimsbf**
}		
hasLocalScreenElevationInformation;	**1**	**bslbf**
if (hasLocalScreenElevationInformation) {		
bsLocalScreenSizeTopEl;	**9**	**uimsbf**
bsLocalScreenSizeBottomEl;	**9**	**uimsbf**
}		
}		

17.5.3 Semantics

isCenteredInAzimuth This flag defines if the screen is centered in azimuthal direction with respect to the listening position (sweet spot).

bsLocalScreenSizeAz This field defines the azimuth corresponding to the left and right screen edge if the screen is centered in azimuth:

$\varphi_{left} = 0.5 \cdot \text{bsLocalScreenSizeAz}$

$\varphi_{left} = \min(\max(\varphi_{left}, 0), 180)$

$\varphi_{right} = -0.5 \cdot \text{bsLocalScreenSizeAz}$

$\varphi_{right} = \min(\max(\varphi_{right}, -180), 0)$

bsLocalScreenSizeLeftAz This field defines the azimuth corresponding to the left screen edge if the screen is not centered in azimuth:

$\varphi_{left} = 0.5 \cdot (\text{bsLocalScreenSizeLeftAz} - 511)$

$\varphi_{left} = \min(\max(\varphi_{left}, -180), 180)$

bsLocalScreenSizeRightAz This field defines the azimuth corresponding to the right screen edge if the screen is not centered in azimuth:

φ_{right} = 0.5 · (bsLocalScreenSizeRightAz – 511)

φ_{right} = min (max (φ_{right} , -180), 180)

hasLocalScreenElevationInformation
This flag defines if elevation information is available.

bsLocalScreenSizeTopEl This field defines the elevation corresponding to the top screen edge:

θ_{top} = 0,5 · (bsLocalScreenSizeTopEl – 255)

θ_{top} = min (max (θ_{top} , -90), 90)

bsLocalScreenSizeBottomEl This field defines the elevation corresponding to the bottom screen edge:

θ_{bottom} = 0,5 · (bsLocalScreenSizeBottomEl – 255)

θ_{bottom} = min (max (θ_{bottom} , -90), 90)

17.6 Interface for signaling of local zoom area

17.6.1 General

A local zoom area can be signalled to the decoder by the LocalZoomAreaSize() interface.

17.6.2 Syntax

Table 259 — Syntax of LocalZoomAreaSize()

Syntax	No. of bits	Mnemonic
LocalZoomAreaSize()		
{		
bsZoomAzCenter;	**10**	**tcimsbf**
bsZoomAz;	**10**	**uimsbf**
bsZoomElCenter;	**9**	**tcimsbf**
bsZoomEl;	**10**	**uimsbf**
}		

17.6.3 Semantics

bsZoomAzCenter This field defines the azimuth corresponding to the centre position of left and right edges of a zoom area in azimuth. This field can take integer values between -360 and +360.

$\varphi_{center}^{ZoomArea} = 0.5 \cdot \text{bsZoomAzCenter}$

$\varphi_{center}^{ZoomArea} = \min(\max(\varphi_{center}^{ZoomArea}, -180), +180)$

bsZoomAz This field defines the azimuth corresponding to the offset from the centre position to left and right edges of a zoom area in azimuth. This field can take integer values between 0 and +720.

$\varphi_{offset}^{ZoomArea} = 0.125 \cdot \text{bsZoomAz}$

$$\varphi_{offset}^{ZoomArea} = \min(\max(\varphi_{offset}^{ZoomArea}, 0), 90)$$

bsZoomElCenter This field defines the elevation corresponding to the centre position of top and bottom edges of a zoom area in elevation. This field can take integer values between -180 and +180.

$$\theta_{center}^{ZoomArea} = 0.5 \cdot \text{bsZoomElCenter}$$

$$\theta_{center}^{ZoomArea} = \min(\max(\theta_{center}^{ZoomArea}, -90), +90)$$

bsZoomEl This field defines the elevation corresponding to the offset from the centre position to top and bottom edges of a zoom area in elevation. This field can take integer values between 0 and +720.

$$\theta_{offset}^{ZoomArea} = 0.125 \cdot \text{bsZoomEl}$$

$$\theta_{offset}^{ZoomArea} = \min(\max(\theta_{offset}^{ZoomArea}, 0), 90)$$

17.7 Interface for user interaction

17.7.1 General

This subclause describes a syntax for a normative interface for user interactivity or communication with the application.

17.7.2 Definition of user interaction categories

Different user interaction categories are defined according to the metadata audio element syntax. All modifications are defined on the level of a group of elements, because groups gather related elements that shall only be manipulated jointly. The possible modification is defined according to the metadata audio element fields.

— **On/Off interactivity**: A group of elements is switched on or off (playback is enabled or disabled).

— **Position interactivity**: The position in 3D space of all elements of a group is changed (azimuth, elevation and distance are modified). The interaction happens by offset values (azimuth, elevation) or a multiplication factor (distance) to allow for an unchanged relative inter-element relationship.

— **Gain interactivity**: The level/gain of all elements of a group is changed. The interaction happens by an additional decibel value that is added to all elements of a group, such that the relative inter-element relationship is kept unchanged.

— **WIRE interactivity**: The audio content of the elements of a group are routed to a WIRE output (e.g. content for hearing impaired or an additional language track, a WIRE output is a generic output in addition to the connected standard output device (loudspeakers or headphones)). No metadata is sent along the unrendered audio content.

17.7.3 Definition of an interface for user interaction

The mpegh3daElementInteraction() syntax element provides an interface for all permitted forms of user interaction. Two interaction modes are defined in the interface.

The first mode is an advanced interaction mode, where the interaction may be signalled for each element group that is present in the audio scene. This mode enables the user to freely choose which groups to play back and to interact with all of them (within the restrictions of allowances and ranges defined in the metadata and the restrictions of switch group definitions).

The second mode is a basic interaction mode, where the user may choose one of a set group presets that are defined in the metadata audio element syntax. With a group preset, the on/off statuses of the groups and switch groups that are referenced in the conditions of the chosen preset are defined and cannot be changed by the user. The user may only change the on/off status of the other groups and switch groups that are not referenced in the conditions of the chosen preset. The position and gain of all groups may be changed according to the defined restrictions and ranges.

A signature is introduced in the interface. The electronic signature may be used to identify and authenticate a particular person, device or software as the originator of the user interaction data.

If the current group preset or group preset extension contains one or more switch group conditions, the following behaviour shall be verified by the metadata pre-processor.

— If a switch group condition is equal to 1, it shall be verified that one member of the corresponding switch group is enabled (on/off status equal to 1). If no member is enabled by the user (via interaction), the default member has to be enabled by the metadata preprocessor.
— All preset-dependent object characteristics and parameters (disable gain interactivity, group preset gain, disable position interactivity, group preset azimuth offset, group preset elevation offset, group preset distance factor) shall be applied to the active member, independent on which member is currently active.
— If a switch group condition is equal to 0, all members of the corresponding switch group shall be set to inactive/disabled.

Switch group conditions with value 0 are only applicable for switch groups whose mae_switchGroupAllowOnOff flag is equal to 1.

17.7.4 Syntax of interaction interface

Table 260 — Syntax of mpegh3daElementInteraction()

Syntax	No. of bits	Mnemonic
mpegh3daElementInteraction()		
{		
ei_InteractionSignatureDataLength;	**8**	**uimsbf**
if (ei_InteractionSignatureDataLength >0) {		
ei_InteractionSignatureDataType;	**8**	**uimsbf**
for (c = 0; c < ei_InteractionSignatureDataLength ; c++) {		
ei_InteractionSignatureData[c];	**8**	**uimsbf**
}		
}		
ElementInteractionData();		
hasLocalZoomAreaSize;	**1**	**bslbf**
if (hasLocalZoomAreaSize) {		
LocalZoomAreaSize();		
}		
}		

Table 261 — Syntax of ElementInteractionData()

Syntax	No. of bits	Mnemonic
ElementInteractionData()		
{		
ei_interactionMode;	**1**	**bslbf**
ei_numGroups; /* Channel, Object, HOA, SAOC */	**7**	**uimsbf**
if (ei_interactionMode == 0) {		
ei_GroupInteractivityStatus(ei_numGroups);		
} else {		
ei_groupPresetID;	**5**	**uimsbf**
ei_GroupInteractivityStatus(ei_numGroups);		
}		
}		

Table 262 — Syntax of ei_GroupInteractivityStatus()

Syntax	No. of bits	Mnemonic
ei_GroupInteractivityStatus (numGroups)		
{		
for (grp = 0; grp < numGroups; grp++) {		
ei_groupID[grp];	**7**	**uimsbf**
ei_onOff[grp];	**1**	**bslbf**
ei_routeToWIRE[grp];	**1**	**bslbf**
if (ei_routeToWIRE[grp] == 1) {		
routeToWireID[grp];	**16**	**uimsbf**
}		
if (ei_onOff [grp] == 1) {		
ei_changePosition[grp]; /* position change */	**1**	**bslbf**
if (ei_changePosition[grp]) {		
ei_azOffset[grp];	**8**	**uimsbf**
ei_elOffset[grp];	**6**	**uimsbf**
ei_distFact[grp];	**4**	**uimsbf**
}		
ei_changeGain; /* gain change */	**1**	**bslbf**
if (ei_changeGain) {		
ei_gain;	**7**	**uimsbf**
}		
}		
}		
}		

17.7.5 Semantics of interaction interface

ei_InteractionSignatureDataLength This field defines the length of the following interaction signature in byte.

ei_InteractionSignatureDataType This field defines the type of signature. The values in Table 263 are possible.

Table 263 — Value of ei_InteractionSignatureDataType

Value	Meaning
0	Generic string in UTF-8 according to ISO/IEC 10646
1-127	Reserved for ISO use.
128-255	Reserved for use outside of ISO scope.

ei_InteractionSignatureData This field contains a signature defining the originator of the interaction data.

hasLocalZoomAreaSize Flag that defines if information about the local zoom area size is available. If this flag is enabled, object remapping for zooming is applied according to subclause 18.5.

ei_interactionMode Flag that defines if the advanced interaction type or the basic interaction mode is chosen. A value of 0 indicates the advanced interaction mode. A value of 1 indicates the basic interaction mode. In case mae_numGroupPresets equals zero ei_interactionMode shall be set to zero, otherwise ei_interactionMode shall be set to one.

ei_numGroups This field contains the number of groups in the audio scene.

ei_groupPresetID This field contains a mae_groupPresetID that is defined in the audio scene. This ID reflects the user's preset choice.

ei_groupID mae_groupID for the current group for which the interaction is described.

ei_routeToWIRE This field defines if the audio content of the group should be routed to a WIRE output.

ei_routeToWireID ID of the WIRE output where the group should be routed to.

ei_onOff Defines the on/off status of the current group. In case the basic interaction mode (interaction on group presets) is chosen, this value has to be identical to the defined on/off status of the group with ei_groupID if this group is part of the conditions of the chosen group preset with ei_groupPresetID. For basic interaction mode it is not allowed to signal a different on/off status here. For all groups that are not part of the conditions of the chosen group preset, the on/off status may arbitrarily be signalled.

ei_changePosition This flag defines if the position of the group elements has been changed (azimuth and/or elevation and/or distance have been modified by the user).

ei_azOffset The change of azimuth is given as an offset. This field can take values between AzOffset=-180° and AzOffset=180°:

$$AzOffset = 1.5 \cdot (ei\_azOffset - 128)$$

$$AzOffset = \min (\max (AzOffset, -180), 180)$$

ei_elOffset The change of azimuth is given as an offset. This field can take values between ElOffset=-90° and ElOffset=90°:

$$ElOffset = 3 \cdot (ei\_elOffset - 32)$$

$$ElOffset = \min (\max (ElOffset, -90), 90)$$

ei_distFact The distance interactivity is given as a multiplication factor. The field can take values between 0 to 15 resulting in DistFactor between 0.00025 and 8:

$$DistFactor = 2^{(ei\_distFactor-12)}$$

$$DistFactor = \min(\max (DistFactor, 0.00025),\ 8)$$

ei_changeGain This flag defines if the gain/level of the group elements has been changed (the gain has been modified by the user)

ei_gain This field defines an additional gain of the members of the current group. The field can take values between 0 and 127 representing gain values between

Gain = -63 dB and Gain = 31 dB in 1 dB steps, with

$$Gain\ [dB] = ei\_gain - 64$$

$$Gain\ [dB] = \min(\max (Gain, -63),\ 31)$$

If ei_gain is set to 0, Gain shall be set to minus infinity dB.

17.8 Interface for loudness normalization and dynamic range control (DRC)

The interface for loudness normalization and dynamic range control is fully specified in ISO/IEC 23003-4.

17.9 Interface for scene displacement data

17.9.1 General

This subclause describes syntax for the normative interface for scene displacement data received by the decoder. Use-cases are the deployment of a tracking device that tracks the user's head (e.g. a virtual reality (VR) device) resulting in a scene displacement that needs to be processed by the decoder framework or a tracking device that directly tracks the scene displacement (e.g. 3D mouse or data glove).

17.9.2 Definition of an interface for scene-displacement data

The mpegh3daSceneDisplacementData() syntax element provides the interface for the scene displacement, given by three Tait-Bryan angles:

— 'yaw' (α_{yaw}) is a rotation around the z axis;

— 'pitch' (β_{pitch}) is a rotation around the x axis;

— 'roll' (θ_{roll}) is a rotation around the y axis.

The three angles describe the relative orientation between two orthogonal right-handed 3D Cartesian coordinate systems by three rotation angles alongside the three coordinate axes, namely between the so-called 'fixed scene setup coordinate system' and the 'deflected scene coordinate system'.

The coordinate system of the scene-displacement data is a right-handed coordinate system using the following direction of the axes:

— x axis pointing to the right;

— y axis pointing straight ahead;

— z axis pointing straight up.

The fixed scene setup coordinate system is the original coordinate system, in which the positioning of loudspeakers (azimuth, elevation), reproductions screen and objects (azimuth, elevation, and radius) are defined.

The deflected scene coordinate system describes the orientation of the virtual scene after the objects positions have been updated. This coordinate system rotates with the deflection of a tracker device from its idle state. The origin is assumed to be the same as the scene coordinate system, but the coordinate system may be deflected along all three axes, indicated by the values given by the scene displacement interface.

Positive rotations correspond to the direction of rotation about each axis as indicated below. For this definition, clockwise and anti-clockwise is determined by looking directly along the axis of rotation, towards the origin of the coordinate system. The positive rotations are illustrated in Figure 100.

— A **positive yaw value** corresponds to a clockwise rotation about **z axis**.

— A **positive pitch value** corresponds to an anti-clockwise rotation about the **x axis**.

— A **positive roll value** corresponds to an anti-clockwise rotation about the **y axis**.

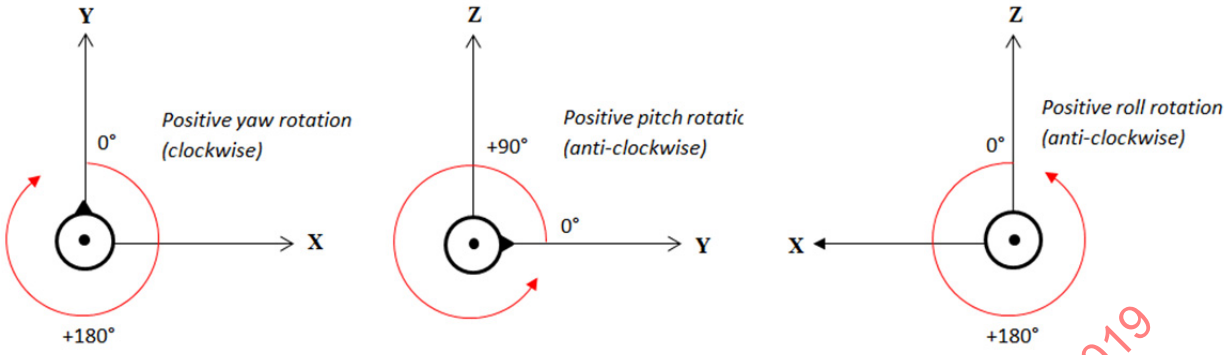


Figure 100 — Directions of positive rotations

17.9.3 Syntax of the scene displacement interface

Table 264 — Syntax of mpegh3daSceneDisplacementData()

Syntax	No. of bits	Mnemonic
mpegh3daSceneDisplacementData() {		
sd_yaw;	**9**	**uimsbf**
sd_pitch;	**9**	**uimsbf**
sd_roll;	**9**	**uimsbf**
}		

17.9.4 Semantics of the scene displacement interface

sd_yaw This field defines the scene displacement angle about the z axis. This field can take values between α_{yaw} == -180° and α_{yaw} == 180°.

$$\alpha_{yaw} = \left(\frac{sd\_yaw}{2^8} - 1 \right) \cdot 180$$

α_{yaw} = min (max (α_{yaw}, -180), 180)

A positive yaw value corresponds to a clockwise rotation about the z axis.

sd_pitch This field defines the scene displacement angle about the x axis. This field can take values between β_{pitch} = -180° and β_{pitch} = 180°.

$$\beta_{pitch} = \left(\frac{sd\_pitch}{2^8} - 1 \right) \cdot 180$$

β_{pitch} = min (max (β_{pitch}, -180), 180)

A positive pitch value corresponds to an anti-clockwise rotation about the x axis.

sd_roll This field defines the scene displacement angle about the y axis. This field can take values between θ_{roll} = -180° and θ_{roll} = 180°.

$$\theta_{roll} = \left(\frac{sd\_roll}{2^8} - 1 \right) \cdot 180$$

θ_{roll} = min (max (θ_{roll}, -180), 180)

A positive roll value corresponds to an anti-clockwise rotation about the y axis.

17.10 Interfaces for channel-based, object-based, and HOA metadata and audio data

17.10.1 General

This subclause describes the output interfaces for the delivery of un-rendered channels, objects, and HOA content and associated metadata. The implementation of these interfaces is optional, however, if implemented, there are specific parts which shall be included.

17.10.2 Expectations on external renderers

If the output interfaces defined in this subclause are implemented and used for connecting to external, non-MPEG-H renderers, then these external renderers should apply and handle the metadata provided in this interface and related audio data in the same manner as if internal rendering is applied. This includes, but is not limited to:

— correct handling of loudness-related metadata in particular with the aim of preserving intended target loudness;

— preserving artistic intent, such as applying transmitted Downmix matrices correctly;

— rendering spatial attributes of objects appropriately (position, spatial extent, etc.);

— correct handling of diffuseness, divergence azimuth range and exclusion sector metadata;

— correct handling of group priorities.

17.10.3 Object-based metadata and audio data (object output interface)

17.10.3.1General

This interface provides the output of metadata and audio data for object-based audio content (distinct objects which have accompanying OAM data, SignalGroupType is equal to SignalGroupTypeObject (given by Signals3d())). It supports the output of metadata and audio data of object-based content after the processing of the element metadata preprocessor module.

The interface is restricted to metadata and audio data for the initial object-based audio elements, that are enabled for playback (switched on), when the interface output data is requested.

The interface does **not support** the output of metadata or audio data of other audio elements.

Note that the output of channel-based elements that are routed to the object renderer during scene-displacement processing (see subclause 18.8) is not supported by this interface.

Content that is received by the object output interface should be processed with the same loudness/DRC processing before output to the interface in the same manner as if they are part of the regular output signals. The application of a peak limiter at the end of the processing chain is highly recommended.

The delay of the received signals should be adjusted before presentation at the interface, such that they have the same delay as the regular output signals. The object renderer output should be muted and not played back in case an output via the object interface is requested.

If an object output interface is provided by an implementation, the following processing shall be applied to the object-based content before the object output interface:

— processing of the object metadata preprocessor module (user interaction, screen-related processing, etc.), except diffuseness processing and divergence processing;

— DRC-1;

— resampling;

— frame truncation (truncation of PCM data);

— DRC-2;

— loudness normalization;

— application of the peak limiter.

Diffuseness processing (i.e. the creation of the diffuse audio part as well as weighting of the direct sound part using the diffuseness-dependent weighting factor as described in subclause 18.11) is bypassed in the situation that an output of object audio data plus metadata is requested via the object output interface. Diffuseness processing should instead be applied by an external renderer.

The object-based audio element replica created during divergence processing (as defined in subclause 18.1) shall not be presented via the object output interface. The weighting of the original objects with the divergence-dependent weighting factor $\rho_{original}$ shall be bypassed in the situation that an output of object audio data plus metadata is requested via the object output interface. Divergence processing should instead be applied by an external renderer.

If an object output interface is provided by an implementation, the following metadata shall be provided via the application specific interface to be interpreted and acted upon by potential external renderers:

— number of output objects;

— information about audio truncation and number of valid PCM frames for the current frame;

— OAM metadata:

  — dynamic object priority (if available);

  — object position (azimuth, elevation, radius);

  — spread;

  — object gain;

— Signal group related metadata:

  — static group priority;

  — “fixed position” flag;

— Enhanced object metadata:

  — diffuseness;

  — divergence and divergence azimuth range;

  — exclusion sector metadata.

The following sub-clauses provide syntax and semantics for a preferred implementation of such an interface. Implementers may choose to provide the information outlined in this preferred implementation according to a different format, e.g. using a different structure, different ordering of data, or different data types for conveying the object metadata.

17.10.3.2Syntax of an interface for object-based metadata

Table 265 — Syntax of mpegh3da_getObjectAudioAndMetadata()

Syntax	No. of bits	Mnemonic
mpegh3da_getObjectAudioAndMetadata()		
{		
/* FRAME CONFIGURATION */		
goa_frameLength;	**6**	**uimsbf**
goa_audioTruncation;	**2**	**bslbf**
if (goa_audioTruncation>0) {		
goa_numSamples;	**13**	**uimsbf**
} else {		
goa_numSamples = goa_frameLength << 6;		
}		
/* OBJECT METADATA */		
goa_numberOfOutputObjects;	**9**	**uimsbf**
for (o = 0; o < goa_numberOfOutputObjects; o++) {		
goa_elementID[o];	**9**	**uimsbf**
goa_hasDynamicObjectPriority[o];	**1**	**bslbf**
goa_hasUniformSpread[o];	**1**	**bslbf**
/* OAM Data */		
goa_numOAMframes[o]	**6**	**uimsbf**
for (nf = 0; nf < goa_numOAMframes[o]; nf++) {		
goa_objectMetadataPresent;	**1**	**bslbf**
if (goa_objectMetadataPresent==1) {		
goa_positionAzimuth[o][nf];	**8**	**uimsbf**
goa_positionElevation[o][nf];	**6**	**uimsbf**
goa_positionRadius[o][nf];	**4**	**uimsbf**
goa_objectGainFactor[o][nf];	**7**	**uimsbf**
if (goa_hasDynamicObjectPriority[o]) {		
goa_dynamicObjectPriority[o][nf];	**3**	**uimsbf**
}		
if (goa_hasUniformSpread[o]) {		
goa_uniformSpread[o][nf];	**7**	**uimsbf**
} else {		
goa_spreadWidth[o][nf];	**7**	**uimsbf**
goa_spreadHeight[o][nf];	**5**	**uimsbf**
goa_spreadDepth[o][nf];	**4**	**uimsbf**
}		
}		
}		
/* Signal group related data */		
goa_fixedPosition[o];	**1**	**bslbf**
goa_groupPriority[o];	**3**	**uimsbf**
/* Enhanced Object Metadata */		
goa_diffuseness[o];	**7**	**uimsbf**
goa_divergence[o];	**7**	**uimsbf**
goa_divergenceAzimuthRange[o];	**6**	**uimsbf**
goa_numExclusionSectors[o];	**4**	**uimsbf**
for (s = 0; s < goa_numExclusionSectors[o]; s++) {		

Syntax	No. of bits	Mnemonic
goa_usePredefinedSector[o][s];	**1**	**bslbf**
if (goa_usePredefinedSector[o][s]) {		
goa_excludeSectorIndex[o][s];	**4**	**uimsbf**
} else {		
goa_excludeSectorMinAzimuth[o][s];	**7**	**uimsbf**
goa_excludeSectorMaxAzimuth[o][s]	**7**	**uimsbf**
goa_excludeSectorMinElevation[o][s];	**5**	**uimsbf**
goa_excludeSectorMaxElevation[o][s]	**5**	**uimsbf**
}		
} */* for (s = 0; s < goa_numExclusionSectors[o]; s++) */*		
} */* for (o = 0; o < goa_numberOfOutputObjects; o++) */*		
}		

17.10.3.3Semantics of the interface for object-based metadata

goa_frameLength

This field is used to determine the length of a nominal audio frame in samples:

nominal audio frame length = goa_frameLength << 6;

goa_audioTruncation

Determines whether the audio output corresponding to this goa frame has been truncated according to Table 266.

Table 266 — truncation direction signalling

Value	Truncation applied
0	No truncation
1	From the left (beginning of buffer)
2	From the right (end of buffer)

goa_numSamples

Valid output PCM samples the decoder produced corresponding to this goa frame. In case of truncation this can be different from the nominal audio frame length. This value should be evaluated by possible external renderers to ensure the application of the received metadata to the correct audio PCM samples.

goa_numberOfOutputObjects

Number of output objects:

$$n_{\text{obj, out}} = \text{goa\_numberOfOutputObjects};$$

goa_elementID

This field gives the mae_metaDataElementID of the current audio object.

goa_hasDynamicObjectPriority

This field defines whether the current object contains a dynamic object priority value.

goa_hasUniformSpread This field defines whether the current object contains uniform or non-uniform spread values.

goa_numOAMframes This field gives the number of OAM frames that are available in the current audio frame for the current object. The minimum is 1 OAM frame per audio frame.

goa_objectMetadataPresent Indicates whether object metadata is present in the current OAM frame.

goa_positionAzimuth This field defines the azimuth position of the current OAM frame of the current object.

azimuth = (goa_positionAzimuth - 128) · 1.5

azimuth = min(max(azimuth, -180), 180)

goa_positionElevation This field defines the elevation position of the current OAM frame of the current object.

elevation = (goa_positionElevation - 32) · 3.0

elevation = min(max(elevation, -90), 90)

goa_positionRadius This field defines the radius of the current OAM frame of the current object.

radius = pow(2.0, (goa_positionRadius/3.0))/2.0

radius = min(max(radius[o], 0.5), 16)

goa_objectGainFactor This field defines the gain of the current OAM frame of the current object.

gain = pow(10.0, (**goa_objectGainFactor** - 96.0)/40.0)

gain = min(max(gain, 0.004), 5.957)

goa_dynamicObjectPriority This field defines the dynamic object priority value of the current OAM frame of the current object. This field can take values between 0 and 7.

goa_uniformSpread This field contains the uniform spread value of the current OAM frame of the current object.

spread = goa_uniformSpread · 1.5

spread = min(max(spread, 0), 180)

goa_spreadWidth This field contains the spread value in width dimension of the current OAM frame of the current object.

spread_width = goa_spreadWidth[o] · 1.5

spread_width = min(max(spread_width, 0), 180)

goa_spreadHeight

This field contains the spread value in height dimension of the current OAM frame of the current object.

spread_height = goa_spreadHeight[o] · 3.0

spread_height = min(max(spread_height, 0), 90)

goa_spreadDepth

This field contains the spread value in depth dimension of the current OAM frame of the current object.

spread_depth = (pow(2.0, (goa_spread_depth/3.0))/2.0) – 0.5

spread_depth = min(max(spread_depth, 0), 16)

goa_fixedPosition

This field defines if the current object's position shall be updated during processing of scene displacement (tracking) data. If the position shall not be updated, the flag is set to 1.

goa_groupPriority

This field defines the priority of the group to which the current object belongs to. It can take integer values between 0 and 7.

goa_diffuseness

This field defines the diffuseness of the group to which the current object belongs to. This field can take values between 0 and 127, corresponding to diffuseness values between 0.0 and 1.0:

diffuseness = (goa_diffuseness/127)

goa_divergence

This field defines the divergence of the objects. The field can take values between 0 and 127, corresponding to divergence values between 0.0 and 1.0:

goa_divergence = (goa_divergence/127)

goa_divergenceAzimuthRange

If the divergence of the object or group is larger than 0.0 (goa_divergence > 0), the goa_divergenceAzimuthRange defines the positioning of the virtual sources. The field can take values between 0 and 63, resulting in azimuth offset angles between 0° and 180°:

$$\varphi_{\text{offset}} = 3.0 \cdot \text{goa\_divergenceAzimuthRange}$$

$$\varphi_{\text{offset}} = \min(\max(\varphi_{\text{offset}}, 0), 180)$$

goa_numExclusionSectors

This field defines the number of exclusion sectors that are defined for the current object.

goa_usePredefinedSector

This field defines if an exclusion sector is signalled by a predefined sector index (flag is equal to 1) or if an arbitrary sector is specified (flag is equal to 0).

goa_excludeSectorIndex

This field defines the sector index of a defined exclusion sector.

goa_excludeSectorMinAzimuth This field defines the minimum excluded azimuth value for a defined exclusion sector.

$$\varphi_{sector,min} = 3.0 \cdot (goa\_excludeSectorMinAzimuth - 63)$$

$$\varphi_{sector,min} = \min(\max(\varphi_{zone,min}, -180), 180)$$

goa_excludeSectorMaxAzimuth This field defines the maximum excluded azimuth value for a defined exclusion sector.

$$\varphi_{sector,max} = 3.0 \cdot (goa\_excludeSectorMaxAzimuth - 63)$$

$$\varphi_{sector,max} = \min(\max(\varphi_{zone,max}, -180), 180)$$

goa_excludeSectorMinElevation This field defines the minimum excluded elevation value for a defined exclusion sector.

$$\theta_{sector,min} = 6.0 \cdot (goa\_excludeSectorMinElevation - 15)$$

$$\theta_{sector,min} = \min(\max(\theta_{zone,min}, -90), 90)$$

goa_excludeSectorMaxElevation This field defines the maximum excluded elevation value for a defined exclusion sector.

$$\theta_{sector,max} = 6.0 \cdot (goa\_excludeSectorMaxElevation - 15)$$

$$\theta_{sector,max} = \min(\max(\theta_{zone,max}, -90), 90)$$

17.10.4 Channel-based metadata and audio data

17.10.4.1General

This interface provides the output of metadata for channel-based audio content (usually one complete signal group with SignalGroupType set to SignalGroupTypeChannels (given by Signals3d())). It supports the output of metadata data of channel-based content after the processing of the element metadata preprocessor module.

The interface is restricted to metadata for the initial channel-based audio elements, that are enabled for playback (switched on) when the interface output data is requested.

The interface does **not support** the output of metadata of other audio elements.

The output of channel-based elements, where the positions shall be modified by the element metadata preprocessor, is not supported. Instead those elements may be treated as object content with positions defined by the indicated loudspeaker positions.

Content that is received by the channel output interface should be processed with the same Loudness/DRC processing before presented at the interface as if they are part of the regular output signals. The application of a peak limiter at the end of the processing chain is highly recommended.

The delay of the received signals should be adjusted before presentation to the interface, such that they have the same delay as the regular output signals. The format converter output should be muted and not played back in case an output via the channel interface is requested.

If a channel output interface is provided by an implementation, the following processing shall be applied to the channel-based content before the channel output interface:

— Processing of the metadata preprocessor module (e.g. user interaction);

— DRC-1;

— Resampling;

— Frame truncation (truncation of PCM data);

— DRC-2;

— Loudness normalization;

— Application of the peak limiter.

If a channel output interface is provided by an implementation, the following metadata shall be provided via the interface to be evaluated by possible external renderers:

— Number of channels;

— Number of valid PCM samples for the current frame;

— elementIDs for the referenced audio channels;

— Channel configuration;

— “fixed position” flag;

— Static group priority;

— Downmix matrix elements, if transmitted and matching the selected Reproduction Layout (according to subclause 10.3.1).

The following subclauses provide syntax and semantics for a preferred implementation of such an interface. Implementers may choose to provide the information outlined in this preferred implementation in a different format, e.g. using a different structure, different ordering of data, or different data types for conveying the object metadata.

17.10.4.2Syntax of an interface for channel-based metadata (informative)

This sub-clause provides syntax and semantics for a preferred implementation of the channel-based metadata interface.

Table 267 — Syntax of mpegh3da_getChannelMetadata()

Syntax	No. of bits	Mnemonic
mpegh3da_getChannelMetadata()		
{		
/* FRAME CONFIGURATION */		
gca_frameLength;	**6**	**uimsbf**
gca_audioTruncation;	**2**	**bslbf**
if (gca_audioTruncation>0) {		
gca_numSamples;	**13**	**uimsbf**
} else {		
gca_numSamples = gca_frameLength << 6;		
}		
/* CHANNEL METADATA */		
gca_numberOfOutputChannelGroups;	**9**	**uimsbf**
for (cGrp = 0; cGrp < gca_numberOfOutputChannelGroups; cGrp ++) {		
gca_numberOfChannels[cGrp]	**16**	**uimsbf**
gca_channelLayout[cGrp] = SpeakerConfig3d();		

Syntax	No. of bits	Mnemonic
for (nChn = 0; nChn < gca_numberOfChannels[cGrp]; nChn++ {		
gca_elementID[cGrp][nChn];	**9**	**uimsbf**
}		
/* TRACKING-RELATED METADATA */		
gca_fixedPosition[cGrp];	**1**	**bslbf**
/* GROUP-RELATED METADATA */		
gca_groupPriority[cGrp];	**3**	**uimsbf**
gca_channelGain[cGrp];	**8**	**uimsbf**
/* DOWNMIX MATRIX ELEMENT */		
gca_downmixAvailable	**1**	**bslbf**
if (gca_downmixAvailable) {		
gca_downmixConfig();		
}		
}		
}		

17.10.4.3 Semantics of the interface for channel-based metadata

gca_frameLength This field is used to determine the length of a nominal audio frame in samples:

nominal audio frame length = gca_frameLength << 6

gca_audioTruncation Determines whether the audio output corresponding to this gca frame has been truncated according to Table 268.

Table 268 — Truncation direction signalling

Value	Truncation applied
0	No truncation
1	From the left (beginning of buffer)
2	From the right (end of buffer)

gca_numSamples Valid output PCM samples the decoder produced corresponding to this gca frame. In case of truncation this can be different from the nominal audio frame length.

gca_numberOfOutputChannelGroups Definition of the number of output channel groups.

gca_elementID This field gives the elementIDs of the audio channels out of the current channel group.

gca_numberOfChannels This field gives the number of channels present in the current channel group.

gca_channelLayout	This field defines the position of each channel.
gca_fixedPosition	This field defines if the current channel's position shall be updated during processing of scene displacement (tracking) data. If the position shall not be updated, the flag is set to 1.
gca_groupPriority	This field defines the priority of the group to which the current object belongs to. It can take integer values between 0 and 7.
gca_channelGain	This field defines the gain of the current frame of the current channel group. gain = pow(10.0, **gca_channelGain** - 96.0)/40.0) gain = min(max(gain, 0.004), 5.957)
gca_downmixAvailable	This flag signals the presence of downmix matrix elements for the given channel group. The transmitted downmix matrix information in downmixConfig() should be evaluated by possible external renderers to preserve highest possible artistic intend from the content creator during the rendering process.
gca_downmixConfig()	This is a subset of the original downmixConfig()-element, only including the elements matching the selected reproduction layout (according to subclause 10.3.1).

17.10.5 HOA metadata and audio data

17.10.5.1General

This interface provides the output of metadata for HOA audio content (usually one complete signal group with SignalGroupType set to SignalGroupTypeHOA (given by Signals3d())). It supports the output of the metadata of the HOA content prior the HOA rendering process.

The interface is restricted to metadata for the decoded HOA coefficients, that are enabled for playback (switched on), when the interface output data is requested.

The interface does not support the output of metadata of other audio elements.

Content that is received by the HOA output interface should be processed with the same loudness/DRC processing before presentation to the interface as if they are part of the regular output signals. The application of a peak limiter at the end of the processing chain is highly recommended.

The delay of the received signals should be adjusted before presentation at the interface, such that they have the same delay as the regular output signals. The HOA interface and the HOA renderer should not operate simultaneously: In case an output via the HOA interface is requested, the output of the HOA renderer should be muted and not reproduce audio and vice versa.

If the HOA output interface is provided by an implementation, the following processing shall be applied to the HOA content before the HOA output interface:

— Processing of the metadata preprocessor module;

— DRC-1;

— Resampling;

— Frame truncation (truncation of PCM data);

— DRC-2;

— Loudness normalization;

— Peak limitation.

If the HOA output interface is provided by an implementation, the following metadata shall be provided via the interface to be interpreted and acted upon by potential external renderers:

— HOA order;

— Number of valid PCM samples for the current frame;

— NFC metadata;

— A flag that indicates if HOA content is relative to a screen and if so, the production screen size information;

— HOA rendering matrix elements, if transmitted and matching the selected reproduction layout.

The following sub-clauses provide syntax and semantics for a preferred implementation of such interface. Implementers may choose to provide the information outlined in this preferred implementation in a different format, e.g. using a different structure, different ordering of data.

17.10.5.2 Syntax of an interface for HOA metadata

Table 269 — Syntax of mpegh3da_getHoaMetadata ()

Syntax	No. of bits	Mnemonic
mpegh3da_getHoaMetadata()		
{		
/* FRAME CONFIGURATION */		
gha_frameLength;	**6**	**uimsbf**
gha_audioTruncation;	**2**	**bslbf**
if (gha_audioTruncation>0) {		
gha_numSamples;	**13**	**uimsbf**
} else {		
gha_numSamples = gha_frameLength << 6;		
}		
/* HOA METADATA */		
gha_numberOfHoaGroups;	**9**	**uimsbf**
for (hGrp = 0; hGrp < gha_numberOfHoaGroups; hGrp ++) {		
gha_HoaOrder[hGrp]**;**	**9**	**uimsbf**
gha_UsesNfc[hGrp]**;**	**1**	**bslbf**
if (gha_UsesNfc[hGrp]) {		
gha_NfcReferenceDistance[hGrp]**;**	**32**	**bslbf**
}		
gha_hasSignalledHoaMatrix[hGrp]**;**	**1**	**uimsbf**
if (gha_hasSignalledHoaMatrix[hGrp]) {		
gha_HoaRenderingMatrixSet();		
}		
gha_isScreenRelative[hGrp]**;**	**1**	**uimsbf**
if (gha_isScreenRelative[hGrp]) {		
mae_ProductionScreenSizeData();		
mae_ProductionScreenSizeDataExtension();		
}		
}		
}		

17.10.5.3Semantics of the interface for HOA metadata

gha_frameLength This field is used to determine the length of a nominal audio frame in samples:

nominal audio frame length = gha_frameLength << 6

gha_audioTruncation Determines whether the audio output corresponding to this gha frame has been truncated according Table 270.

Table 270 — Truncation direction signalling

Value	Truncation applied
0	No truncation
1	From the left (beginning of buffer)
2	From the right (end of buffer)

gha_numSamples Valid output PCM samples the decoder produced corresponding to this gha frame. In case of truncation this can be different from the nominal audio frame length.

gha_HoaOrder The HOA order of the outputted audio content.

gha_ numberOfHoaGroups This element indicates the number signal groups of type SignalGroupTypeHOA.

gha_UsesNfc This element determines whether or not the HOA near field compensation (NFC) has been applied to the coefficient signals.

gha_NfcReferenceDistance This element determines the radius in meter that has been used for the HOA NFC (interpreted as float in IEEE 754 format in little-endian).

gha_hasSignalledHoaMatrix This element indicates if the content was transmitted with HOA rendering matrices in the bitstream.

gha_HoaRenderingMatrixSet This is a subset of the original HOARenderingMatrixSet()-element, only including the elements matching the selected reproduction layout.

gha_isScreenRelative This element indicates if the HOA representation shall be rendered with respect to the reproduction screen size.

17.10.6 Audio PCM data

The PCM data of the channels and objects interfaces shall be provided through the decoder PCM buffer, which first contains the regular rendered PCM signals (e.g. 12 signals for a 7.1+4 setup). Subsequently $n_{chan,\ out}$ additional signals carry the PCM data of the originally transmitted channel representation.

These are followed by $n_{obj,\,out}$ signals carrying the PCM data of the un-rendered output objects. Then additional signals carry the $n_{HOA,\,out}$ HOA data which number is indicated in the HOA metadata interface via the HOA order (e.g. 16 signals for HOA order 3). The HOA audio data in the HOA output interface is provided in the so-called equivalent spatial domain representation. The conversion from the HOA domain into the equivalent spatial domain representation and vice versa is described in Annex C.5.1.

The decoder shall signal the offset index of the PCM buffer for the first un-rendered output object.

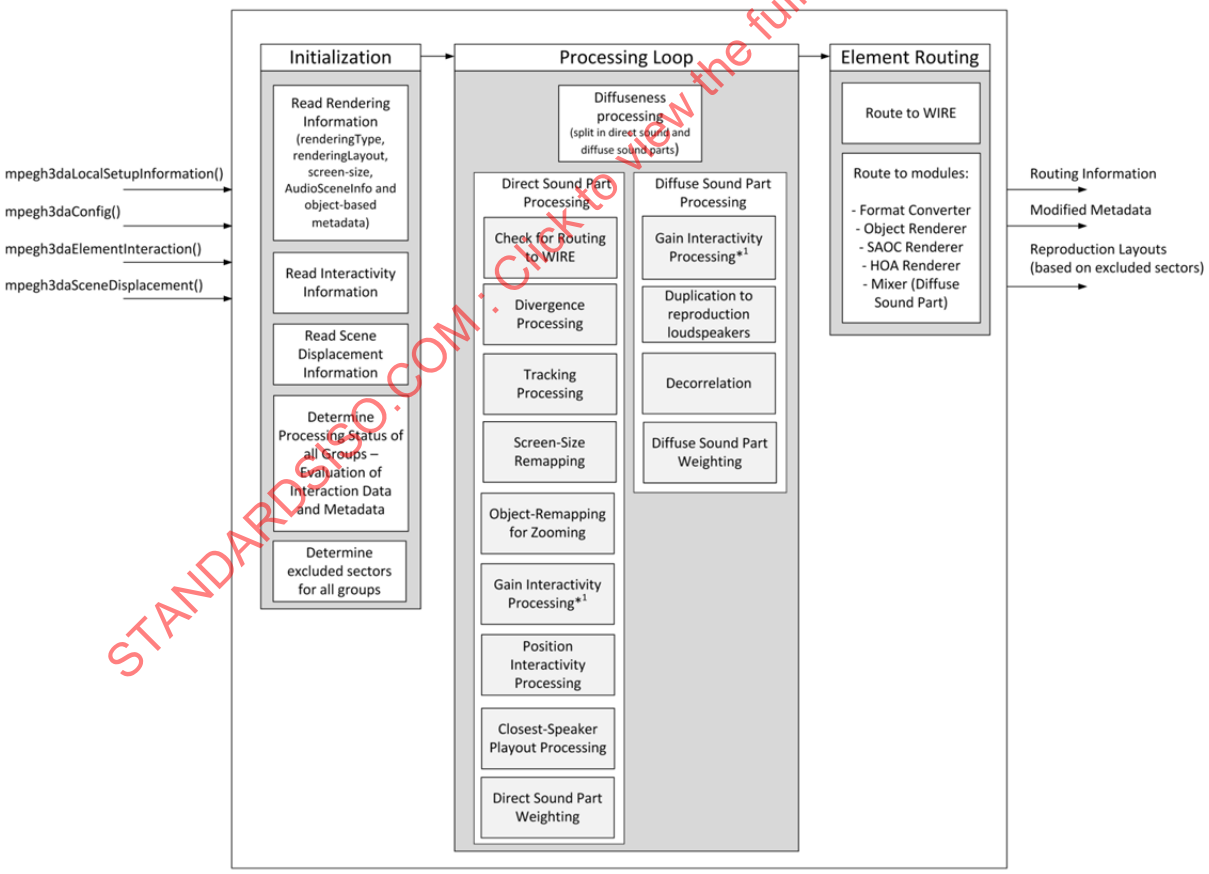
18 Application and processing of local setup information and interaction data and scene displacement data

18.1 Element metadata preprocessing

A processing block is defined that processes the user interaction interface data, the scene displacement data and prepares the audio elements for rendering and play-out. This "element metadata preprocessor" also applies the logic that is imposed by the user interaction and the element metadata.

The following processing structure and order of processing the interface data (interactivity interface mpegh3daElementInteraction(), scene displacement interface mpegh3daSceneDisplacementData() and interface for local setup information mpegh3daLocalSetupInformation()) and metadata audio elements is defined. An overview is given in Figure 101.

The divergence processing has to be conducted before the processing of scene displacement angles (tracking processing) is carried out.



*1 The gain interactivity block may be processed at an arbitrary position in the metadata processing loop, hoewever it has to be ensured that both the direct sound part and the diffuse sound part are adjusted in their gain according to the gain interaction data.

Figure 101 — Processing of metadata and interface data

As a first step the rendering type, rendering layout and screen size information is determined. Then, the syntax element mae_AudioSceneInfo() is retrieved.

The overall number of object-based audio elements that need to be rendered by the object renderer is determined. The overall number of object-based audio elements is:

— the sum of the number of elements with OAM data;
— plus 2 times the number of elements with OAM data with a divergence value bigger than 0 and whose groups are not marked to be sent to a WIRE output.

After the number of object-based audio elements has been determined, the interactivity data is read from the interactivity interface.

The interaction type (basic or advanced) is determined. Next, it is determined which elements have to be processed. This is defined by the on/off status of their corresponding groups, the WIRE routing information and the implicit logic by switch group definitions.

In the basic interaction mode, the user may choose one of a set of defined group presets.

First, it has to be checked if a downmixId is present and if the chosen preset has a group preset extension (defined in mae_GroupPresetDefinitionExtension()) that references the current downmixId. If this is the case, the conditions of the corresponding group preset extension have to be used to determine which groups shall be played back/rendered. The on/off status of the groups and switch groups that are referenced in the chosen group preset's or the applicable group preset extension's conditions are set according to these conditions. Any discrepancy in these values compared to the values in the ei_GroupInteractivityStatus() syntax element shall be ignored.

For the groups that are not referenced in the chosen group preset's conditions, the status is set to the mae_defaultOnOff value as a default, but the user may choose the on/off status accordingly if the definition of the group permits it and no switch group logic is violated.

The elements from the following groups have to be processed in accordance with the following.

— All groups that are switched on and are not member of a switch group.
— All groups that are switched on and that are member of a switch group if the switch group condition is fulfilled (a maximum of one member of this switch group is switched on).
— All groups that are switched off but are default members of a switch group from whose members no other member is marked as switched on. The on/off status of this default group shall then be set to 1 (on).
— All groups that are switched off and are marked to be routed to a WIRE output.

The relevant groups are known at this point in time so the core decoder may be controlled such that irrelevant elements are ignored and not decoded.

Next, the different reproduction layouts are determined by evaluating any excluded sectors, according to subclause 18.10. The different reproduction layouts (target rendering layouts) are used to control one or more instances of the object renderer.

Then the elements of the relevant groups are prepared for playout and rendering. Updated output element characteristics are calculated for each element of each group and the additional virtual objects for groups with divergent values greater than 0, excluding those outputs destined for WIRE output: Gain, azimuth, elevation, and distance. Azimuth and elevation either represent the element's position or the element's loudspeaker's position (in case the closestSpeakerPlayout flag is set to one and the closest

loudspeaker playout is unconditioned, or in case the closestSpeakerPlayout flag is set to one and the loudspeaker is located within the range defined by closestSpeakerThresholdAngle as defined in subclause 18.7 if the closest loudspeaker playout is conditioned).

In case of groups with SignalGroupTypeChannels, azimuth and elevation define the unmodified loudspeaker's position (in case the divergence related to the element is equal to 0 or not existing and the closestSpeakerPlayout flag is set to one and the closest loudspeaker playout is unconditioned, or in case the closestSpeakerPlayout flag is set to one and the loudspeaker is located within the range defined by the closestSpeakerThresholdAngle as defined in subclause 18.7 if the closest loudspeaker playout is conditioned).

For the determination of the updated element characteristics, it shall be established whether each element group is marked to be routed to a WIRE output.

Any evaluation of a valid/selected preset (and therefore the application of preset-dependent values) shall only be performed in the defined 'basic interaction mode' (see subclause 17.7.3). The chosen/selected preset is indicated by the presetID that is received by the mpegh3daElementInteraction() interface. Any preset-dependent value (e.g. group preset gain) is therefore only applied if basic interaction mode is enabled.

If a group is not routed to a WIRE output (non-WIRE group), the screen-related remapping is conducted for all group members that are marked as screen-related. If a group is marked as to be routed to a WIRE output (WIRE group), then this remapping should be skipped.

The divergence processing is the next process to be conducted. Objects with a divergence larger than 0 are replicated: Two additional 'virtual objects' are created in addition to each object with divergence > 0. The 'unmodified' positions and gains for the 'virtual' objects for the reproduction of objects with a divergence larger than 0 are determined according to:

$$\varphi_{\text{virtual,left}} = \left(\varphi_{\text{orig,OAM}} + \varphi_{\text{offset}}\right)$$

$$\text{if } \left(\varphi_{\text{virtual,left}} > 180\right) \ \left\{\varphi_{\text{virtual,left}} = \varphi_{\text{virtual,left}} - 360\right\}$$

$$\text{if } \left(\varphi_{\text{virtual,left}} < -180\right) \ \left\{\varphi_{\text{virtual,left}} = \varphi_{\text{virtual,left}} + 360\right\}$$

$$\varphi_{\text{virtual,right}} = \left(\varphi_{\text{orig,OAM}} - \varphi_{\text{offset}}\right)$$

$$\text{if } \left(\varphi_{\text{virtual,right}} > 180\right) \ \left\{\varphi_{\text{virtual,right}} = \varphi_{\text{virtual,right}} - 360\right\}$$

$$\text{if } \left(\varphi_{\text{virtual,right}} < -180\right) \ \left\{\varphi_{\text{virtual,right}} = \varphi_{\text{virtual,right}} + 360\right\}$$

The gains ρ are based on the current OAM gain and the divergence value:

$$\rho_{\text{orig, temp}} = \sqrt{1 - \varepsilon_{\text{divergence}}^{\,0.58497}}$$

$$\rho_{\text{original}} = \rho_{\text{orig, temp}} \cdot \rho_{\text{orig, OAM}}$$

$$\rho_{\text{virtual, temp}} = \sqrt{0.5 \cdot (\varepsilon_{\text{divergence}})^{0.58497}}$$

$$\rho_{\text{virtual, left}} = \rho_{\text{virtual, right}} = \rho_{\text{virtual, temp}} \cdot \rho_{\text{orig, OAM}}$$

After that, the virtual objects are treated in the same way as the original objects from the bitstream for the calculation of the updated output element characteristics.

In particular, the virtual objects are assigned with the same isScreenRelativeObject-value as their corresponding original object for screen-related remapping and zooming. If the gain interaction is applied after the divergence processing, all ranges and restrictions of the original objects shall also apply to their corresponding virtual object copies.

The duplicated objects are added to the same group as the original object. They are also assigned the original group-dependent characteristics, e.g. for the subsequent interaction processing. As a next step, the scene displacement data is processed if the 'useTrackingMode' flag is equal to one in either binauralRendering() or loudspeakerRendering(). Therefore, the data from the scene displacement interface is read and updated positions for all tracked objects and channels (indicated by 'fixedPosition' = 0) are calculated as defined in subclause 18.8.

Next, the gain interactivity data is processed. This step is applied to all elements of non-WIRE groups and WIRE groups. It is first checked whether the group of the current element allows for gain interactivity (mae_allowGainInteractivity flag is equal to 1).

If that is the case and if the basic interaction mode is active, it is then checked whether the currently active/valid preset disables the gain interactivity.

The gain interactivity may be processed at an arbitrary position in the metadata processing chain. If the gain interactivity processing is performed after the divergence processing, then the replicated objects should be processed with the same gain interaction as the corresponding original objects.

If the gain interactivity is still enabled, the output gain shall be calculated for each element of the group and for any additional virtual objects if the group has a divergence value larger than 0 (in case divergence processing is conducted before gain interaction), taking into account the interactivity gain modification, and either the OAM gain (if available), the unmodified gain of the virtual objects (in case divergence processing is conducted before gain interaction) or the current gain of an element (in case no OAM data is available). If the basic interaction mode is enabled, the possible group gain value of the currently active/selected preset or the applicable group preset extension for the current downmixId shall also be taken into account.

The overall gain modification (interactivity gain in dB plus group gain in dB) shall be restricted according to the given mae_interactivityMinGain and mae_interactivityMaxGain values of the group of the current element. If no gain interaction is allowed, the output gain shall either contain 0dB (in case no OAM data is available), the OAM gain in dB, or the unmodified gain in dB for the virtual objects if divergence processing is conducted before gain interaction.

No gain interaction data is given in the following cases:

— ei_changeGain in ei_GroupInteractivityStatus() is zero, or

— ei_GroupInteractivityStatus() is not existing.

Therefore, the ei_gain is set to the value corresponding to the current mae_groupPresetGain for the gain interaction processing. When an interaction gain is given by means of ei_gain and the ei_changeGain flag is equal to one, the value of ei_gain shall take priority over the group preset gain.

If no mae_groupPresetGain is given, the groupPresetGain shall be assumed to be 0 dB.

If the group of the current element is marked as WIRE output, then no further interactivity processing shall be applied.

If the group of the current element is not marked as WIRE output, then the application of further processing depends on the signal type. If the element is a channel-based element

(SignalGroupTypeChannels), then no further interactivity processing shall be applied. Instead the position of the predefined associated loudspeaker shall be determined from the audioChannelLayout information. It is assumed that the member elements in the member list of the associated group are ordered with respect to the channel ordering signalled in the syntax element Signals3d().

If the current element is accompanied by OAM data, then the position interactivity shall be processed. Therefore, it is first checked if the group of the current element allows for position interactivity (mae_allowPositionInteractivity flag is equal to 1).

If that is the case and if basic interaction mode is also active, then it is established whether the currently active/valid preset or the applicable group preset extension for the current downmixId disables the position interactivity.

If the position interactivity is still enabled for the current group, azimuth, elevation and distance values are determined for each element of the group and the additional virtual objects if the divergence of the group is larger than 0. The calculation of the output values shall take into account the interactivity modification (offset values and distance multiplication factor) and the values from the OAM data or respectively the unmodified gain in dB for the virtual objects. If the basic interaction mode is enabled, the possible group preset values (PresetAzOffset, PresetElOffset, PresetDistFactor) of the currently selected/valid preset are also taken into account as follows: When a preset is selected and no position interaction data is given, the group preset values are to be applied to the relevant elements as initial position interaction.

No position interaction data is given in the following cases

— ei_changePosition in ei_GroupInteractivityStatus() is zero, or

— ei_GroupInteractivityStatus() is not existing

Therefore, the values of ei_azOffset, ei_elOffset and ei_distFact are then set to the values corresponding to the current mae_groupPresetAzOffset, mae_groupPresetElOffset and mae_groupPresetDistFactor for the position interaction processing. When positional interaction data is given by means of ei_azOffset, ei_elOffset and ei_distFact and the ei_changePosition flag is equal to one, the values of ei_azOffset, ei_elOffset and ei_distFact shall take priority over the corresponding group preset values.

If no mae_groupPresetAzOffset is given, the PresetAdditionalAzOffset shall be assumed to be 0°. If no mae_groupPresetElOffset is given, the PresetAdditionalElOffset shall be assumed to be 0°. If no mae_groupPresetDistFactor is given, the PresetAdditionalDistFactor shall be assumed to be equal to 1.

The overall positional interactivity modifications (given by the values of the interactivity azimuth offset, interactivity elevation offset and interactivity distance factor (either ei_azOffset, ei_elOffset and ei_distFact or mae_groupPresetAzOffset, mae_groupPresetElOffset and mae_groupPresetDistFactor respectively as defined above)) shall be restricted according to the given position interactivity ranges.

As a next step, the closest loudspeaker playout processing shall be conducted. If an element is marked as closest loudspeaker playout, it shall be evaluated as an initial step if the closest loudspeaker playout processing shall be performed with or without conditioning.

If the closest loudspeaker playout is unconditioned, the position of the closest loudspeaker to the output position data is determined as defined in subclause 18.6, taking into account all reproduction loudspeakers. No rendering shall be applied; it therefore has to be ensured that the determined loudspeaker position exists within the reproduction loudspeaker setup.

If the closest loudspeaker playout is conditioned (a closestSpeakerThresholdAngle value is given), it has to be determined, which of the reproduction loudspeakers lies in the given range according to subclause 18.7.

After determining the list of loudspeakers in the range, the position of the closest loudspeaker with respect to the output position data is determined as defined in subclause 18.6, taking into account only the reproduction loudspeakers in the range. No rendering shall be applied; it therefore has to be ensured that the determined loudspeaker position exists in the reproduction loudspeaker setup.

After processing the interactivity data, the routing information for the elements is determined. If an element group is marked as WIRE output, the members of this group should be directly sent according to the WIRE ID. The output gain shall be applied before WIRE output. Metadata regarding gain interactivity as well as Loudness/DRC processing should be applied before playing out the audio signal. Content that is routed to a WIRE output should be processed with the same Loudness/DRC processing as if they would be part of the regular output signals. The application of a peak limiter at the end of the processing chain is highly recommended. The signals on the WIRE output should have the same delay as signals after the mixer.

If an element group is not marked as WIRE output, then the output depends on the signal type.

Channel-based elements with 'fixedPosition = 1' are marked to be sent to the format converter, object-based element as well as channel-based groups with 'fixedPosition = 0' are marked to be sent to the object renderer, SAOC-based elements are marked to be sent to the SAOC renderer, HOA-based elements are marked to be sent to the HOA renderer.

In addition to the decoded audio data, each renderer gets azimuth, elevation, distance (either element's position or element's loudspeaker's position), gain and the rendering layout. The SAOC renderer and the HOA renderer may require additional side information.

The group-dependent reproduction layouts depending on the signalled excluded sectors are routed to the object renderer. Depending upon the number of individual reproduction layouts, multiple instances of the object renderer may be used.

The application of the output element gains shall be performed before playout.

18.2 Interactivity limitations and restrictions

18.2.1 General information

The application of user interactivity is restricted by the metadata audio element syntax that is defined for the groups of elements. There, the interaction possibilities may be limited for each group by the metadata fields mae_allowOnOff, mae_allowGainInteractivity and mae_allowPositionInteractivity.

In addition the interaction possibilities are restricted by the logic of groups and switch groups.

18.2.2 WIRE interactivity

WIRE interactivity is only defined if the local reproduction setup and the setup of connected WIRE outputs (including WIRE IDs) are known at the user interface or application. A WIRE output can be a device that supports a single channel or a multichannel signal.

WIRE output are only allowed for groups of type SignalGroupTypeChannels or SignalGroupTypeObject.

The decoder output is a single channel or multichannel signal that is sent to the WIRE output. The number of channels corresponds to the number of members of the element group that is routed to a WIRE output. If multiple groups are routed to the same WIRE output, then the number of channels corresponds to the total number of elements in these groups.

The channel-order corresponds to the order of elements in the group's member list. If multiple groups are routed to the same WIRE output the order of channels is defined with ascending mae_groupID and within each group by the order of elements in the group's member list.

If the number of elements that are routed to a WIRE output does not correspond to the number of channels that the WIRE output could handle (e.g. sending a channel-based element group with stereo dialogue to a WIRE headset with just one output channel), the application has to handle this discrepancy.

As also switched-off groups can be routed to a WIRE output (e.g. for support of alternative languages or voice-over content over WIRE), the audio decoder shall allow output of switched-off groups which are routed to WIRE outputs.

Metadata regarding gain interactivity as well as loudness/DRC processing shall be applied before outputting the audio signal. Content that is routed to a WIRE output shall be processed with the same loudness/DRC processing as if they would be part of the regular output signals. The application of a peak limiter at the end of the processing chain is highly recommended.

If there are WIRE output devices and audio content is routed to a WIRE output, then the order of outputs after the decoder shall be as follows: first the regular signals and then the content for the WIRE outputs sorted by ascending WIRE ID.

18.2.3 Position interactivity

Position interactivity is only defined for elements where positional information is provided (OAM data in the syntax element object_metadata(), see Table 135).

For elements with SignalGroupTypeChannels, no position interactivity is defined, because they are intended to be played back by a specific loudspeaker.

For elements that are marked as "closestSpeakerPlayout", the determination of the closest loudspeaker should be conducted after processing the position interactivity data.

If an element group is marked to be routed to a WIRE output, position interactivity shall not be applied to the members of the group. The unrendered audio content is sent to the WIRE output and no positional information is sent along.

18.2.4 Screen-related element remapping and object remapping for zooming

Screen-related element remapping and object remapping for zooming are only defined for elements that are accompanied by OAM data in the syntax element object_metadata().

If no local screen size information is available, no screen-related element remapping shall be applied. In case only azimuth screen size information is given, the decoder shall not apply any screen-related element remapping of the elevation of screen-related elements.

If no local zoom information is available, no object remapping for zooming shall be applied.

If an element group is marked to be routed to a WIRE output, screen-related element remapping and object remapping for zooming shall be skipped for the members of this group, because the unrendered audio content is sent to the WIRE output without positional information.

18.2.5 Closest loudspeaker playout

Closest loudspeaker playout is only defined for elements that are accompanied by OAM data in the syntax element object_metadata().

If an element group is marked to be routed to a WIRE output, signalling of the closestSpeakerPlayout option shall be ignored for the members of this group, because the unrendered audio content is sent to the WIRE output.

If an element has a divergence value bigger than 0, the signaling of the closestSpeakerPlayout option shall be ignored and no closest loudspeaker playout processing shall be conducted. Closest loudspeaker playout processing shall only be conducted for objects with spread=0° (uniform spread signalling) or spread_width=spread_height=0° (non-uniform spread signalling).

18.3 Screen-related element remapping

Screen-related element remapping is only processed if the bitstream contains screen-related elements (isScreenRelativeObject flag == 1 for at least one audio element) that are accompanied by OAM data and if the local screen size is signalled to the decoder via the LocalScreenSizeInformation() interface.

The geometric positional data (OAM data before any position modification by user interaction has happened) is mapped to a different range of values by the definition and utilization of a mapping-function. The remapping changes the geometric positional data as a pre-processing step to the rendering, such that the renderer is agnostic of the remapping and operates unchanged.

The screen size of a nominal reference screen (used in the mixing and monitoring process) and local screen size information in the playback room are taken into account for the remapping.

The size of the applicable nominal reference screen is in general read from the bitstream, located in the mae_ProductionScreenSizeData() syntax structure.

If the structure mae_ProductionScreenSizeDataExtension() is present in the bitstream, mae_NumPresetProductionScreens is bigger than 0 and the interaction mode is set to 'basic interaction mode', the groupPresetID of the currently chosen/valid group preset shall be used to identify the applicable screen size of the reference screen from the mae_ProductionScreenSizeDataExtension() structure before the remapping is applied.

If no is chosen or the chosen preset has no associated production screen, the production screen from the mae_ProductionScreenSizeData() structure shall be used as a default production screen.

If the structure mae_ProductionScreenSizeDataExtension() is present in the bitstream, and the mae_overwriteProductionScreenSizeData flag as well as the hasNonStandardScreenSize flag from mae_ProductionScreenSizeData() are equal to 1, the azimuth angle data originating from mae_ProductionScreenSizeDataExtension() shall be used instead of the azimuth angle data from mae_ProductionScreenSizeData() to define the default production screen.

If no nominal reference screen size is given, default reference values are used assuming a 4k display and an optimal viewing distance.

If no local screen size information is given, then remapping shall not be applied.

Two linear mapping functions are defined for the remapping of the elevation and the azimuth values.

The screen edges of the nominal screen size are given by:

$$\varphi_{left}^{nominal}, \varphi_{right}^{nominal}, \theta_{top}^{nominal}, \theta_{bottom}^{nominal}$$

The reproduction screen edges are abbreviated by:

$$\varphi_{left}^{repro}, \varphi_{right}^{repro}, \theta_{top}^{repro}, \theta_{bottom}^{repro}$$

Azimuth values of a temporary reproduction screen are defined by:

$$\varphi_{\text{offset}}^{\text{repro}} = \begin{cases} (360° + \varphi_{\text{left}}^{\text{repro}} + \varphi_{\text{right}}^{\text{repro}}) \cdot \frac{1}{2} & \text{if}\left(\varphi_{\text{right}}^{\text{repro}} > \varphi_{\text{left}}^{\text{repro}}\right) \\ (\varphi_{\text{left}}^{\text{repro}} + \varphi_{\text{right}}^{\text{repro}}) \cdot \frac{1}{2} & \text{else} \end{cases}$$

$$\varphi_{\text{left, temp}}^{\text{repro}} = \varphi_{\text{left}}^{\text{repro}} - \varphi_{\text{offset}}^{\text{repro}}$$

$$\varphi_{\text{right, temp}}^{\text{repro}} = \varphi_{\text{right}}^{\text{repro}} - \varphi_{\text{offset}}^{\text{repro}}$$

$$\varphi_{\text{left, temp}}^{\text{repro}} = \begin{cases} \varphi_{\text{left, temp}}^{\text{repro}} - 360° & \text{if}(\varphi_{\text{left, temp}}^{\text{repro}} > 180°) \\ \varphi_{\text{left, temp}}^{\text{repro}} + 360° & \text{if}(\varphi_{\text{left, temp}}^{\text{repro}} < -180°) \\ \varphi_{\text{left, temp}}^{\text{repro}} & \text{else} \end{cases}$$

$$\varphi_{\text{right, temp}}^{\text{repro}} = \begin{cases} \varphi_{\text{right, temp}}^{\text{repro}} - 360° & \text{if}(\varphi_{\text{right, temp}}^{\text{repro}} > 180°) \\ \varphi_{\text{right, temp}}^{\text{repro}} + 360° & \text{if}(\varphi_{\text{right, temp}}^{\text{repro}} < -180°) \\ \varphi_{\text{right, temp}}^{\text{repro}} & \text{else} \end{cases}$$

Azimuth values of a temporary reference screen are defined by:

$$\varphi_{\text{offset}}^{\text{nominal}} = \begin{cases} (360° + \varphi_{\text{left}}^{\text{nominal}} + \varphi_{\text{right}}^{\text{nominal}}) \cdot \frac{1}{2} & \text{if}\left(\varphi_{\text{right}}^{\text{nominal}} > \varphi_{\text{left}}^{\text{nominal}}\right) \\ (\varphi_{\text{left}}^{\text{nominal}} + \varphi_{\text{right}}^{\text{nominal}}) \cdot \frac{1}{2} & \text{else} \end{cases}$$

$$\varphi_{\text{left, temp}}^{\text{nominal}} = \varphi_{\text{left}}^{\text{nominal}} - \varphi_{\text{offset}}^{\text{nominal}}$$

$$\varphi_{\text{right, temp}}^{\text{nominal}} = \varphi_{\text{right}}^{\text{nominal}} - \varphi_{\text{offset}}^{\text{nominal}}$$

$$\varphi_{\text{left, temp}}^{\text{nominal}} = \begin{cases} \varphi_{\text{left, temp}}^{\text{nominal}} - 360° & \text{if}(\varphi_{\text{left, temp}}^{\text{nominal}} > 180°) \\ \varphi_{\text{left, temp}}^{\text{nominal}} + 360° & \text{if}(\varphi_{\text{left, temp}}^{\text{nominal}} < -180°) \\ \varphi_{\text{left, temp}}^{\text{nominal}} & \text{else} \end{cases}$$

$$\varphi_{\text{right, temp}}^{\text{nominal}} = \begin{cases} \varphi_{\text{right, temp}}^{\text{nominal}} - 360° & \text{if}(\varphi_{\text{right, temp}}^{\text{nominal}} > 180°) \\ \varphi_{\text{right, temp}}^{\text{nominal}} + 360° & \text{if}(\varphi_{\text{right, temp}}^{\text{nominal}} < -180°) \\ \varphi_{\text{right, temp}}^{\text{nominal}} & \text{else} \end{cases}$$

A temporal input azimuth value is defined according to:

$$\varphi_{\text{temp}} = \varphi - \varphi_{\text{offset}}^{\text{nominal}}$$

$$\varphi_{\text{temp}} = \begin{cases} \varphi_{\text{temp}} - 360° & \text{if}(\varphi_{\text{temp}} > 180°) \\ \varphi_{\text{temp}} + 360° & \text{if}(\varphi_{\text{temp}} < -180°) \\ \varphi_{\text{temp}} & \text{else} \end{cases}$$

The remapping of the azimuth and elevation position data is defined by the following linear mapping functions:

$$\varphi'_{\text{temp}} = \begin{cases} \dfrac{\varphi^{\text{repro}}_{\text{right,temp}} + 180°}{\varphi^{\text{nominal}}_{\text{right,temp}} + 180°} \cdot \left(\varphi_{\text{temp}} + 180°\right) - 180° & for \quad -180° \le \varphi_{\text{temp}} < \varphi^{\text{nominal}}_{\text{right,temp}} \\ \dfrac{\varphi^{\text{repro}}_{\text{left,temp}} - \varphi^{\text{repro}}_{\text{right,temp}}}{\varphi^{\text{nominal}}_{\text{left,temp}} - \varphi^{\text{nominal}}_{\text{right,temp}}} \cdot \left(\varphi_{\text{temp}} - \varphi^{\text{nominal}}_{\text{right,temp}}\right) + \varphi^{\text{repro}}_{\text{right,temp}} & for \quad \varphi^{\text{nominal}}_{\text{right}} \le \varphi_{\text{temp}} < \varphi^{\text{nominal}}_{\text{left,temp}} \\ \dfrac{180° - \varphi^{\text{repro}}_{\text{left,temp}}}{180° - \varphi^{\text{nominal}}_{\text{left,temp}}} \cdot \left(\varphi_{\text{temp}} - \varphi^{\text{nominal}}_{\text{left,temp}}\right) + \varphi^{\text{repro}}_{\text{left,temp}} & for \quad \varphi^{\text{nominal}}_{\text{left,temp}} \le \varphi_{\text{temp}} < 180° \end{cases}$$

$$\varphi' = \varphi'_{\text{temp}} + \varphi^{\text{repro}}_{\text{offset}}$$

$$\varphi' = \begin{cases} \varphi' - 360° & \text{if}\,(\varphi' > 180°) \\ \varphi' + 360° & \text{if}\,(\varphi' < -180°) \\ \varphi' & \text{else} \end{cases}$$

A mapping function for the mapping of the azimuth is depicted in Figure 102. The curve is defined such that the azimuth values between the nominal reference left edge azimuth and the nominal reference right edge azimuth are mapped (compressed or expanded) to the interval between the given local left screen edge and the given local right screen edge. The other azimuth values are compressed or expanded accordingly, such that the whole range of values is covered.

The remapped azimuth can take values between -180° and 180° and the remapped elevation can take values between -90° and 90°.

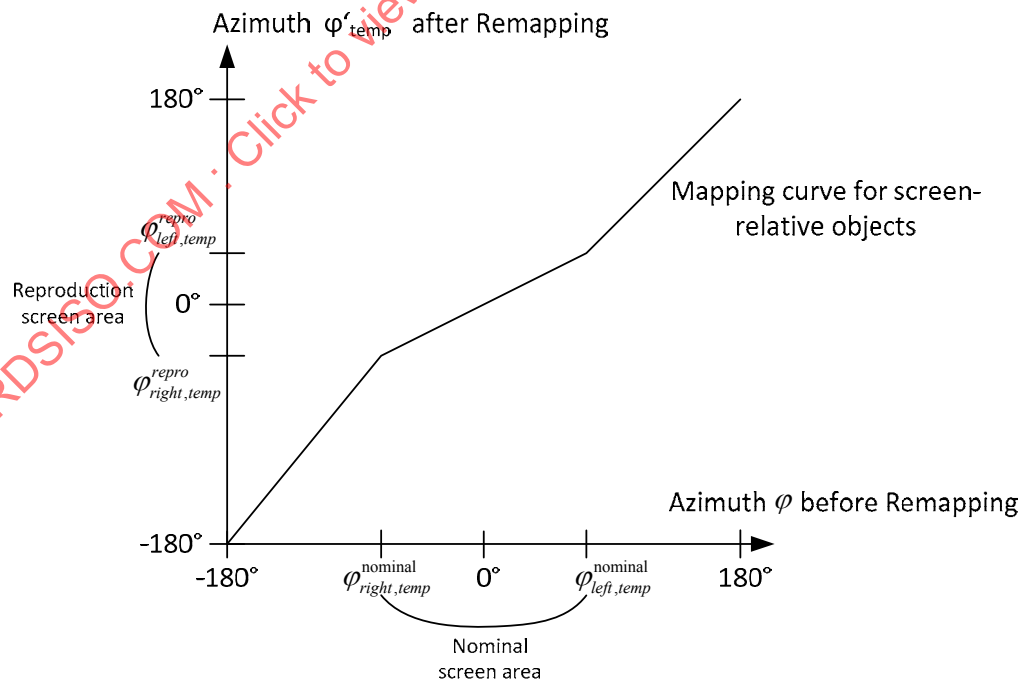


Figure 102 — Remapping function of position data

If the isScreenRelativeObject flag is set to zero, then no screen-related element remapping is applied for the corresponding element and the geometric positional data (OAM data plus positional change by user interactivity) is directly used by the renderer to compute the playback signals.

18.4 Screen-related adaptation and zooming for higher order ambisonics (HOA)

Screen-related adaptation and zooming is only possible if the IsScreenRelative flag in the HOAConfig() (see Table 187) is signalled as 1.

The screen-related adaptation process modifies the HOA rendering matrix and is only computed during the initialization phase. Figure 103 depicts the process. If no local screen size information is available, no screen-related adaptation shall be applied. In case only azimuth screen size information is given, the decoder shall not apply any screen-related adaptation in the vertical dimension.

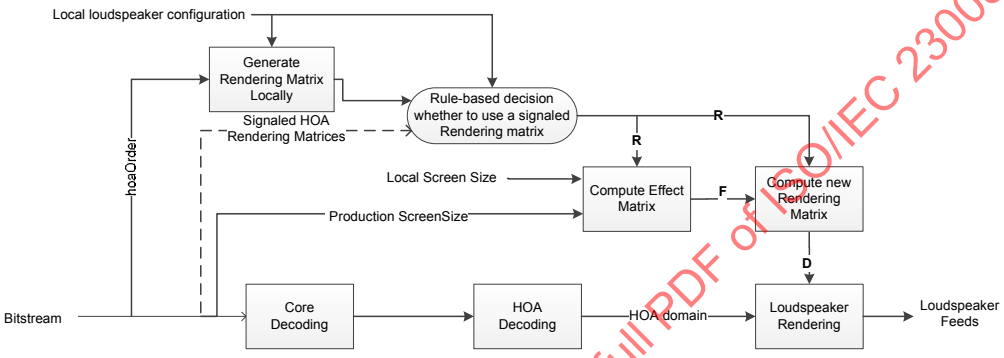


Figure 103 — Screen-related processing for higher order ambisonics

The screen-related adaptation of the rendering matrix is achieved by:

1) Generating a mode matrix $\boldsymbol{\Psi}^{(O,M)}$ as described in Annex F.1.5 with M=900 sampling points which directions (θ, ϕ) are defined in Annex F.9. $\boldsymbol{\Psi}^{(O,M)} := \left[S_1^O \; S_2^O \; \ldots \; S_M^O \right] \epsilon \mathbb{R}^{O \times M}$.

2) The directions of those M sampling points are first modified using the mapping function defined in subclause 18.3. Then a mode matrix $\boldsymbol{\Psi}_m^{(O,M)}$ based on these modified points is computed accordingly.

3) Computing a preliminary effect matrix:

$$\widetilde{\boldsymbol{F}} = \boldsymbol{\Psi}_m^{(O,M)} \, \boldsymbol{\Psi}^{(O,M)\dagger},$$

where $\boldsymbol{\Psi}^{(O,M)\dagger}$ denotes the pseudo-inverse of the mode matrix $\boldsymbol{\Psi}^{(O,M)}$.

4) Computing a loudness correction value by using the HOA rendering matrix R (see subclause 12.4.3.2) for each spatial direction $l = 1 \ldots M$:

$$A(l) = \sqrt{\frac{(\boldsymbol{R}\, S_{m,l}^O)^T (\boldsymbol{R}\, S_{m,l}^O)}{(\boldsymbol{R}\, \widetilde{\boldsymbol{F}} S_l^O)^T (\boldsymbol{R}\, \widetilde{\boldsymbol{F}}\, S_l^O)}}$$

5) Computing the final effect matrix:

$$\boldsymbol{F} = \boldsymbol{\Psi}_m^{(O,M)} diag(A) \, \boldsymbol{\Psi}^{(O,M)\dagger},$$

where $diag(A)$ denotes a diagonal matrix including the vector A.

6) Computing the new rendering matrix:
$\boldsymbol{D} = \boldsymbol{RF}$

The zoom-depending adaptation of higher order ambisonics is depicted in Figure 104. If no local zoom information is available, zooming shall not be applied. The same algorithmic principles as described for the screen-related processing for higher order ambisonics are applied, but the rendering matrix shall be adapted at runtime according to the data provided by the LocalZoomAreaSize() interface. During a dynamic zooming event, a new effect matrix $\boldsymbol{F}$ shall be computed based a mode matrix $\boldsymbol{\Psi}^{(O,M)}$with $M = (N+2)^2$ equally spatial sampling points which directions are given in Annex F.5 to F.9. Once the zoom is stationary, the new effect matrix $\boldsymbol{F}$ is computed based a mode matrix $\boldsymbol{\Psi}^{(O,M)}$with $M =$ 900 spatial sampling points as decribed above.

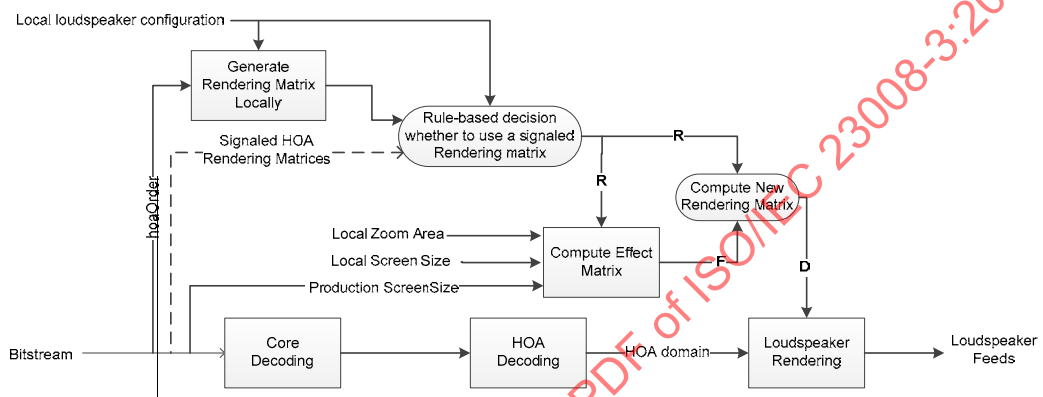


Figure 104 — Overview of zooming for higher order ambisonics

Additional information about screen-related adaptation of HOA can be found in Annex M.

18.5 Object remapping for zooming

Object remapping for zooming is only processed if the bitstream contains screen-related elements (isScreenRelativeObject == 1 for at least one audio element) that are accompanied by OAM data and a local zoom area is signalled to the decoder via the LocalZoomAreaSize() interface.

If zoom area information is given, then object remapping for zooming is applied after screen-related element remapping. Otherwise, no object remapping for zooming shall be applied.

The zoom area defines a part of the video content that is expanded to the whole reproduction screen.

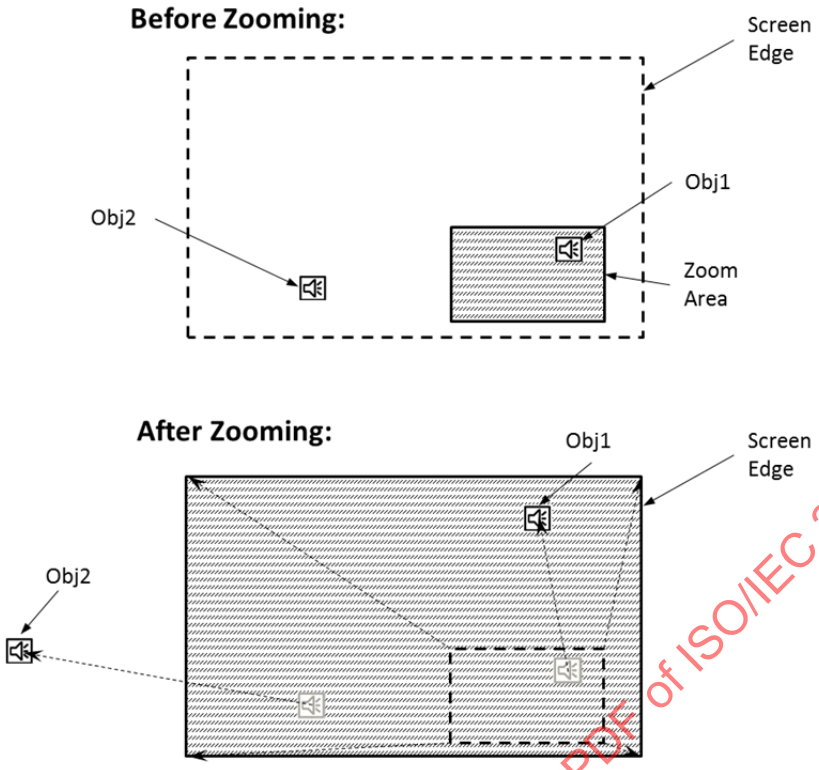


Figure 105 — Screen-related object remapping for zooming

The position of any element that is marked as screen-related and that has accompanying OAM data is changed accordingly. The input values for the object remapping for zooming are the remapped values φ' and θ'.

The edges of the zoom area are given by:

$$\varphi_{left}^{ZoomArea}, \varphi_{right}^{ZoomArea}, \theta_{top}^{ZoomArea}, \theta_{bottom}^{ZoomArea}$$

They are defined as:

$$\varphi_{left}^{ZoomArea} = \varphi_{center}^{ZoomArea} + \varphi_{offset}^{ZoomArea}$$

$$\varphi_{right}^{ZoomArea} = \varphi_{center}^{ZoomArea} - \varphi_{offset}^{ZoomArea}$$

$$\theta_{top}^{ZoomArea} = \theta_{center}^{ZoomArea} + \theta_{offset}^{ZoomArea}$$

$$\theta_{bottom}^{ZoomArea} = \theta_{center}^{ZoomArea} - \theta_{offset}^{ZoomArea}$$

The mapping function for the object remapping for zooming is defined just as the mapping function for the screen-related element remapping; only the edges of the nominal screen size are replaced by the edges of the zoom area.

18.6 Determination of the closest loudspeaker

The distance of two positions P_1 and P_2 in a spherical coordinate system is defined as the absolute difference of their azimuth angles φ and elevation angles θ.

$$\Delta(P_1, P_2) = |\theta_1 - \theta_2| + |\varphi_1 - \varphi_2|$$

This distance has to be calculated for all known position P_1 to P_N of a defined list of N output loudspeakers with respect to the wanted position of the audio element P_{wanted}.

The nearest known loudspeaker position is the one, where the distance to the wanted position of the audio element gets minimal:

$$P_{next} = \min(\Delta(P_{wanted}, P_1), \Delta(P_{wanted}, P_2), \ldots, \Delta(P_{wanted}, P_N))$$

If more than one loudspeaker position is calculated to have the minimal distance to the audio element, then the loudspeaker position shall be chosen which is listed earliest in the list of output loudspeaker channels.

18.7 Determination of a list of loudspeakers for conditioned closest loudspeaker playback

If the closest loudspeaker playback is conditioned, the 'closest loudspeaker processing' shall only be conducted if one or more loudspeakers are located in a defined area around the members of the group.

The corresponding object should then be played back by the closest loudspeaker within this defined range. The area is defined by minimum and maximum values based on the given threshold angle (see subclause 7.4.4 for semantics):

— $\varphi_{min} = \varphi_{obj} - \varphi_{thresh}$

— $\varphi_{max} = \varphi_{obj} + \varphi_{thresh}$

— $\theta_{min} = \theta_{obj} - \theta_{thresh}$

— $\theta_{max} = \theta_{obj} + \theta_{thresh}$

If a loudspeaker lies within the closest loudspeaker playout processing range, both the following conditions have to be true:

— $\varphi_{speaker} \geq \varphi_{obj} - \varphi_{thresh}$ && $\varphi_{speaker} \leq \varphi_{obj} + \varphi_{thresh}$

— $\theta_{speaker} \geq \theta_{obj} - \theta_{thresh}$ && $\theta_{speaker} \leq \theta_{obj} + \theta_{thresh}$

Special cases have to be considered for values out of the allowed range:

— $\varphi_{min} < -180°$

— $\varphi_{max} > +180°$

— $\theta_{min} < -90°$

— $\theta_{max} > +90°$

Minimum or maximum elevation is out of range, minimum and maximum azimuth are within the allowed range:

If $((\theta_{max} > 90° \,||\, \theta_{min} < -90°)$ && $(\varphi_{min} \geq -180°$ && $\varphi_{max} \leq 180°)$, the following condition has to be true:

$$(\varphi_{speaker} \geq \varphi_{min} \,\&\&\, \varphi_{speaker} \leq \varphi_{max}) \,\&\&\, (\theta_{speaker} \geq \theta_{min,1} \,\&\&\, \theta_{speaker} \leq \theta_{max,1})$$

$$||$$

$$(\varphi_{speaker} \geq \varphi_{min,2} \,\&\&\, \varphi_{speaker} \leq \varphi_{max,2}) \,\&\&\, (\theta_{speaker} \geq \theta_{min,2} \,\&\&\, \theta_{speaker} \leq \theta_{max,2})$$

with

$$\left.\begin{aligned}\theta_{\min,1} &= \theta_{\min}\\ \theta_{\max,1} &= 90°\\ \theta_{\min,2} &= 90° - \left|90° - \theta_{\max}\right|\\ \theta_{\max,2} &= 90°\\ \varphi_{\min,2} &= \varphi_{\min} + 180°\\ \varphi_{\max,2} &= \varphi_{\max} + 180°\end{aligned}\right\} \text{if } \theta_{\max} > 90° \qquad \left.\begin{aligned}\theta_{\min,1} &= -90°\\ \theta_{\max,1} &= \theta_{\max}\\ \theta_{\min,2} &= -90°\\ \theta_{\max,2} &= 90° + \left|90° + \theta_{\min}\right|\\ \varphi_{\min,2} &= \varphi_{\min} + 180°\\ \varphi_{\max,2} &= \varphi_{\max} + 180°\end{aligned}\right\} \text{if } \theta_{\min} < -90°$$

Minimum or maximum azimuth is out of range, minimum and maximum elevation are within the allowed range:

If $((\varphi_{max} > 180° \,||\, \varphi_{min} < -180°) \,\&\&\, (\theta_{min} \geq -90° \,\&\&\, \theta_{max} \leq 90°)$, the following condition has to be true:

$$(\varphi_{speaker} \geq \varphi_{min,1} \,\&\&\, \varphi_{speaker} \leq \varphi_{max,1}) \,\&\&\, (\theta_{speaker} \geq \theta_{min} \,\&\&\, \theta_{speaker} \leq \theta_{max})$$
$$||$$
$$(\varphi_{speaker} \geq \varphi_{min,2} \,\&\&\, \varphi_{speaker} \leq \varphi_{max,2}) \,\&\&\, (\theta_{speaker} \geq \theta_{min} \,\&\&\, \theta_{speaker} \leq \theta_{max})$$

with

$$\left.\begin{aligned}\varphi_{\min,1} &= -180°\\ \varphi_{\max,1} &= \varphi_{\max}\\ \varphi_{\min,2} &= 180° - (\varphi_{\min} + 180°)\\ \varphi_{\max,2} &= 180°\end{aligned}\right\} \text{if } \varphi_{\min} < -180° \qquad \left.\begin{aligned}\varphi_{\min,1} &= \varphi_{\min}\\ \varphi_{\max,1} &= 180°\\ \varphi_{\min,2} &= -180°\\ \varphi_{\max,2} &= -180° + (\varphi_{\max} - 180°)\end{aligned}\right\} \text{if } \varphi_{\max} > 180°$$

Minimum or maximum azimuth is out of range, and minimum or maximum elevation is out of range:

If $((\varphi_{max} > 180° \,||\, \varphi_{min} < -180°) \,\&\&\, (\theta_{min} < -90° \,||\, \theta_{max} > 90°)$, the following condition has to be true:

$$(\varphi_{speaker} \geq \varphi_{min,1} \,\&\&\, \varphi_{speaker} \leq \varphi_{max,1}) \,\&\&\, (\theta_{speaker} \geq \theta_{min,1} \,\&\&\, \theta_{speaker} \leq \theta_{max,1})$$
$$||$$
$$(\varphi_{speaker} \geq \varphi_{min,2} \,\&\&\, \varphi_{speaker} \leq \varphi_{max,2}) \,\&\&\, (\theta_{speaker} \geq \theta_{min,1} \,\&\&\, \theta_{speaker} \leq \theta_{max,1})$$
$$||$$
$$(\varphi_{speaker} \geq \varphi_{min,3} \,\&\&\, \varphi_{speaker} \leq \varphi_{max,3}) \,\&\&\, (\theta_{speaker} \geq \theta_{min,2} \,\&\&\, \theta_{speaker} \leq \theta_{max,2})$$

with

$$\left.\begin{aligned}\varphi_{\min,1} &= -180°\\ \varphi_{\max,1} &= \varphi_{\max}\\ \varphi_{\min,2} &= 180° - (\varphi_{\min} + 180°)\\ \varphi_{\max,2} &= 180°\end{aligned}\right\} \text{if } \varphi_{\min} < -180° \qquad \left.\begin{aligned}\varphi_{\min,1} &= \varphi_{\min}\\ \varphi_{\max,1} &= 180°\\ \varphi_{\min,2} &= -180°\\ \varphi_{\max,2} &= -180° + (\varphi_{\max} - 180°)\end{aligned}\right\} \text{if } \varphi_{\max} > 180°$$

$$\left.\begin{aligned}\theta_{\min,1} &= \theta_{\min}\\ \theta_{\max,1} &= 90°\\ \theta_{\min,2} &= 90° - \left|90° - \theta_{\max}\right|\\ \theta_{\max,2} &= 90°\\ \varphi_{\min,3} &= \varphi_{\min} + 180°\\ \varphi_{\max,3} &= \varphi_{\max} + 180°\end{aligned}\right\} \text{if } \theta_{\max} > 90° \qquad \left.\begin{aligned}\theta_{\min,1} &= -90°\\ \theta_{\max,1} &= \theta_{\max}\\ \theta_{\min,2} &= -90°\\ \theta_{\max,2} &= 90° + \left|90° + \theta_{\min}\right|\\ \varphi_{\min,3} &= \varphi_{\min} + 180°\\ \varphi_{\max,3} &= \varphi_{\max} + 180°\end{aligned}\right\} \text{if } \theta_{\min} < -90°$$

For each of the output loudspeaker, it has to be determined if it lies within the defined area. The closest loudspeaker playout should then only take into account the loudspeakers within this range.

18.8 Processing of scene displacement angles for channels and objects (CO)

If the 'useTrackingMode' flag of either binauralRendering() or loudspeakerRendering() is equal to one, the data received by the scene displacement interface mpegh3daSceneDisplacementData() shall be processed. The received angles 'yaw' (α_{yaw}), 'pitch' (β_{pitch}) and 'roll' (θ_{roll}) describe how each object's position has to be updated (i.e. the scene displacement around the z axis, x axis and y axis).

— A **positive yaw value** corresponds to a clockwise rotation about **z axis**.

— A **positive pitch value** corresponds to an anti-clockwise rotation about the **x axis**.

— A **positive roll value** corresponds to an anti-clockwise rotation about the **y axis**.

For each object and channel, it has to be checked if the 'fixedPosition' flag indicated in mae_GroupDefinitionTrackingExtension() of mae_Data() is equal to 0 or 1. Afterwards, for each element with 'fixedPosition' = 0', the position has to be updated.

Channels are therefore interpreted as objects with the position of their corresponding loudspeaker as the object's position: The real playback loudspeaker position shall be taken into account if given by 'known Position'. Otherwise the ideal loudspeaker position shall be assumed.

No position interaction shall be processed for these former channel-based elements, as well as no closest loudspeaker processing, no screen-related remapping, no spread rendering, no excluded sectors processing and no divergence or diffuseness processing.

The position update consists of the following steps.

— First, the object or channel position $p = (az, el, r)$ is determined. For channel-based signals the radius *r* is determined as follows:

  — If the intended loudspeaker exists in the reproduction loudspeaker setup and the distance of the reproduction setup is known, the radius *r* is set to the loudspeaker distance (in cm).

  — If the intended loudspeaker does not exist in the reproduction loudspeaker setup, but the distance of the reproduction loudspeakers is known, the radius *r* is set to the maximum reproduction loudspeaker distance.

  — If the intended loudspeaker does not exist in the reproduction loudspeaker setup and no reproduction loudspeaker distance is known, the radius *r* is set to 1 023 cm.

— After that, the position p is converted to the position p', according to the 'common' convention, where 0° azimuth is at the right ear (positive values going anti-clockwise) and 0° elevation is top of the head (positive values going downwards), resulting in $p' = (az', el', r)$

$$az' = az + 90°$$

$$el' = 90° - el$$

— The position p' is then transferred to Cartesian coordinates (x,y,z), assuming the following direction of coordinate axes:

  — x axis pointing to the right;

  — y axis pointing straight ahead;

  — z axis pointing straight up.

$$x = r \cdot \sin(el') \cdot \cos(az')$$
$$y = r \cdot \sin(el') \cdot \sin(az')$$
$$z = r \cdot \cos(el')$$

— The resulting position $v = (x, y, z)$ is then rotated with the rotation being dependent on the scene displacement input data.

— An intrinsic rotation shall be calculated using a z-x-y convention ('Yaw-Pitch-Roll Convention' (YPR)). The rotation can e.g. be realized by multiplication with a rotation matrix T_{rot}.

— This rotation results in a position $v' = (x', y', z')$

— This updates position v' is then transferred back to an azimuth-elevation notation according to the 'common' convention:

$$el_2' = \arccos\left(\frac{z'}{\sqrt{x'^2 + y'^2 + z'^2}}\right)$$

$$az_2' = \operatorname{atan2}(y', x') = \begin{cases} 90° - \operatorname{sign}(x') \cdot 90° & \text{if} \quad y' == 0 \\ \operatorname{sign}(y') \cdot 90° & \text{else if} \quad x' == 0 \\ \arctan\left(\frac{y'}{x'}\right) & \text{else if} \quad x' > 0 \\ \arctan\left(\frac{y'}{x'}\right) + 180° & \text{else if} \quad (y' > 0) \\ \arctan\left(\frac{y'}{x'}\right) - 180° & \text{else} \end{cases}$$

$$r_2' = \sqrt{x'^2 + y'^2 + z'^2}$$

— Position $p_2' = (az_2', el_2')$ is then transferred back to the MPEG-H notation, resulting in a final updated position $p_2 = (az_2, el_2)$

$$az_2 = az_2' - 90°$$
$$el_2 = 90° - el_2'$$
$$r_2 = r_2'$$

After that, objects and channels are fed to the object renderer with their updated positions. The object renderer renders them using the new object positions. The underlying triangulation stays the same, independent of the tracking data.

The DRC set selection process according to subclause 6.4.4 shall ignore DRC sets for DRC-3 if scene displacement processing is enabled at the decoder.

18.9 Processing of scene displacement angles for scene-based content (HOA)

The rotation of the HOA representation takes place after spatial HOA decoding and DRC-1 but before the optional warping for the screen adaptation and rendering. It can be achieved by two equivalent approaches.

1) The first approach itself consists of three following steps:
   a) Transforming the original HOA representation of order N to the spatial domain, i.e. representing it by general plane waves from $O = (N+1)^2$ directions of incidence $\boldsymbol{\Omega}_q^{(N)}$, $q = 1, \ldots, O$, defined in Annexes F.2 to F.9.
   b) Rotating the directions according to the desired rotation of the HOA representation to provide O rotated directions $\widehat{\boldsymbol{\Omega}}_q^{(N)}$, $q = 1, \ldots, O$.
   c) Applying an inverse spatial transform to the general plane waves assuming the rotated directions to provide the rotated HOA representation.

   The three operations can be expressed by
   $\widehat{\boldsymbol{C}} = \boldsymbol{\Psi}_{new}\ \boldsymbol{\Psi}_{\boldsymbol{O}}^{-1}\ \boldsymbol{C}$,
   with $\boldsymbol{C} \in \mathbb{R}^{(N+1)^2 \times L}$ and $\widehat{\boldsymbol{C}} \in \mathbb{R}^{(N+1)^2 \times L}$ denoting the frames of the original and rotated HOA representations, $\boldsymbol{\Psi}_{\boldsymbol{O}}^{-1}$ denoting the inverse of the mode matrix with respect to the directions $\boldsymbol{\Omega}_q^{(N)}$ expressing the spatial transform and $\boldsymbol{\Psi}_{\mathbf{new}}$ indicating the mode matrix with respect to the rotated directions $\widehat{\boldsymbol{\Omega}}_q^{(N)}$. The mode matrices are computed from the directions as described in according in Annex F.1.5.
   The approach can be beneficial for lowering the computational complexity in cases the HOA representation is already given by in the spatial domain.
   The rotation of the directions can be accomplished as follows:
   First, the directions $\boldsymbol{\Omega}_q^{(N)} = (\theta_q^{(N)}, \phi_q^{(N)})$, $q = 1, \ldots, O$, defined in Annexes F.2 to F.9 and given in spherical coordinates are first converted to positions $\boldsymbol{\Omega}_{\boldsymbol{xyz}}^{(N)} = [x_q, y_q, z_q]^T$ (assuming a radius of one) in Cartesian coordinates , i.e. $\boldsymbol{\Omega}_q^{(N)} \rightarrow \boldsymbol{\Omega}_{\boldsymbol{xyz}}^{(N)}$ by
   $x_q = sin\ \theta_q^N\ \cos \phi_q^N$
   $y_q = sin\ \theta_q^N\ \sin \phi_q^N$
   $z_q = cos\ \theta_q^N$
   Then, the rotated positions are computed by:
   $\widehat{\boldsymbol{\Omega}}_{\mathbf{xyz}}^{(N)} = \boldsymbol{T}_{\boldsymbol{rot}_{\boldsymbol{hoa}}} \boldsymbol{\Omega}_{\boldsymbol{xyz}}^{(N)}$
   where $\boldsymbol{T}_{\boldsymbol{rot}_{\boldsymbol{hoa}}}$ is the rotation matrix for the HOA coordinate system derived from the scene displacement angles (yaw, pitch roll), see Annex I.
   Eventually, the rotated positions are converted back to the spherical coordinate system $\widehat{\boldsymbol{\Omega}}_{\mathbf{xyz}}^{(N)} \rightarrow \widehat{\boldsymbol{\Omega}}^{(N)}$ by:
   $\hat{\theta}_q^{(N)} = atan2\left(\sqrt{\hat{x}^2 + \hat{y}^2}, \hat{z}\right)$,
   $\hat{\phi}_q^{(N)} = atan2(\hat{y}, \hat{x})$
   where $atan2()$ is defined according to 18.8.

2) By expressing the rotation directly as one single matrix multiplication applied to the frame of the original HOA representation as:
   $\widehat{\boldsymbol{C}} = \boldsymbol{M}\ \boldsymbol{C}$

   where the elements of the transform matrix $\boldsymbol{M}$ can be computed by $\boldsymbol{M} = \boldsymbol{\Psi}_{new}\ \boldsymbol{\Psi}_{\boldsymbol{O}}^{-1}$ or directly using so –called Wigner-D functions [13]. Note that the transform matrix $\boldsymbol{M}$ has a special block-diagonal structure, which looks for an HOA order of 2 as follows:

$$\mathbf{M}|_{N=2} = \begin{bmatrix} 1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & [3x3] & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & [5x5] \end{bmatrix}.$$

Due to this rather sparse structure, the matrix multiplication with ***M*** can be accomplished very efficiently.

The DRC set selection process according to subclause 6.4.4 shall ignore DRC sets for DRC-3 if scene displacement processing is enabled at the decoder.

18.10 Determination of a reduced reproduction layout based on excluded sectors

If it should be determined for a specific list of reproduction loudspeakers which ones shall be excluded from rendering based on a list of signalled exclusion sectors, the following processing has to be conducted.

- For each relevant signal group a 'group-related' reproduction layout is determined, based on the signaling of excluded sectors in the EnhancedObjectMetadataConfig() and EnhancedObjectMetadataFrame() structures.
- Therefore, each excluded sector of the current group is associated with a 'condition', conditioning the positions of loudspeakers that shall be excluded for rendering.
  - If the excluded sector is given by a pre-defined sector index, the condition is defined as stated in Table 271.
  - If the excluded sector is an arbitrary sector defined by minimum and maximum excluded elevation and azimuth values, the condition is given by:

$$\left((el \geq el_{\min}) \,\&\&\, (el \leq el_{\max})\right) \,\&\&\, \left((az \geq az_{\min}) \,\&\&\, (az \leq az_{\max})\right)$$

- Each reproduction loudspeaker's position (az, el) is compared to all conditions given by the list of excluded sectors associated with the current group. If a condition is true, the current loudspeaker shall be excluded from rendering.

Table 271 — Conditions of excluded sectors

Sector Index	Short description	Condition
0	No positive elevation	el > 0
1	No negative elevation	el < 0
2	No front	((abs(az) < 90) && (abs(el) < 90))
3	No right side	((az > -180) && (az < 0))
4	No left side	((az > 0) && (az < 180))
5	No surround	((abs(az) >= 90) || ((az=0) && (el=90)))
6	Screen only	(((az > φ^{repro}_{left}) || (az < φ^{repro}_{right})) && ((el > θ^{repro}_{top}) || (el < θ^{repro}_{bottom})))
7-15	Reserved	—

This processing is only applied if

— the targetLayout is signalled in the LoudspeakerRendering(), and

  — the speakerLayoutType is 0 or 1.

Any signalling of 'known Positions' (in LoudspeakerRendering()) shall not be taken into account.

Multiple instances of the object renderer module may be needed to render the content to the 'group-related' reproduction layouts (group-related target rendering layouts).

18.11 Diffuseness rendering

To render objects whose groups have a 'diffuseness' parameter bigger than zero, a weighted sum of a so-called direct sound part and a diffuse sound part has to be calculated depending on the value of the diffuseness parameter before playback.

Two signal versions are created for each object with a diffuseness value bigger than zero: A 'direct sound part' and a 'diffuse sound part'.

The direct sound part is the normal output of the metadata preprocessor, including all processing steps that are related to element metadata preprocessing, such as:

— spread processing;

— position and gain interaction;

— closest loudspeaker playout processing;

— screen-related remapping and zooming;

— processing of excluded sectors;

— etc.

The diffuse sound part is created by replicating the object audio content to the number *N*, with *N* being the number of total available reproduction loudspeakers without LFEs.

Each of these *N* signals is filtered with a decorrelation filter. Gain interaction is applied here as well, such that an interaction gain affects both parts equally. See Annex J for an example of how a decorrelation filter can be realized.

Both the direct sound part and the diffuse sound part are in addition weighted with a gain factor, which is dependent of the diffuseness value of the group, to which the current object belongs to.

The following functions for the two weights g_{dir}, g_{diff} are defined:

$$g'_{\text{diff}} = \sqrt{\text{diffuseness}}$$

$$g_{\text{dir}} = \sqrt{1.0 - g'_{\text{diff}}}$$

$$g_{\text{diff}} = g'_{\text{diff}} \cdot \sqrt{N}$$

EXAMPLE Diffuseness = 0.0 → only the direct part is playing.

$$g_{\text{dir}} = 1.0, \quad g'_{\text{diff}} = 0.0$$

EXAMPLE Diffuseness = 1.0 → only the diffuse part is playing.

$$g_{\text{dir}} = 0.0, \quad g'_{\text{diff}} = 1.0$$

EXAMPLE Diffuseness = 0.5 → both parts are playing with equal weights.

$$g_{\text{dir}} = \sqrt{0.5}, \quad g'_{\text{diff}} = \sqrt{0.5}$$

The direct sound part is sent to the object renderer; the diffuse sound part is directly sent to the mixer, where the two paths are combined and a mix of direct sound and diffuse sound is created. Note that the diffuse part does not contain signals for LFE loudspeakers.

The routing of the diffuse part to the mixer shall be disabled in case an output via the object interface is requested. The weighting of the direct sound part with the factor g_{dir} shall also be omitted.

The overall processing is depicted in Figure 106.

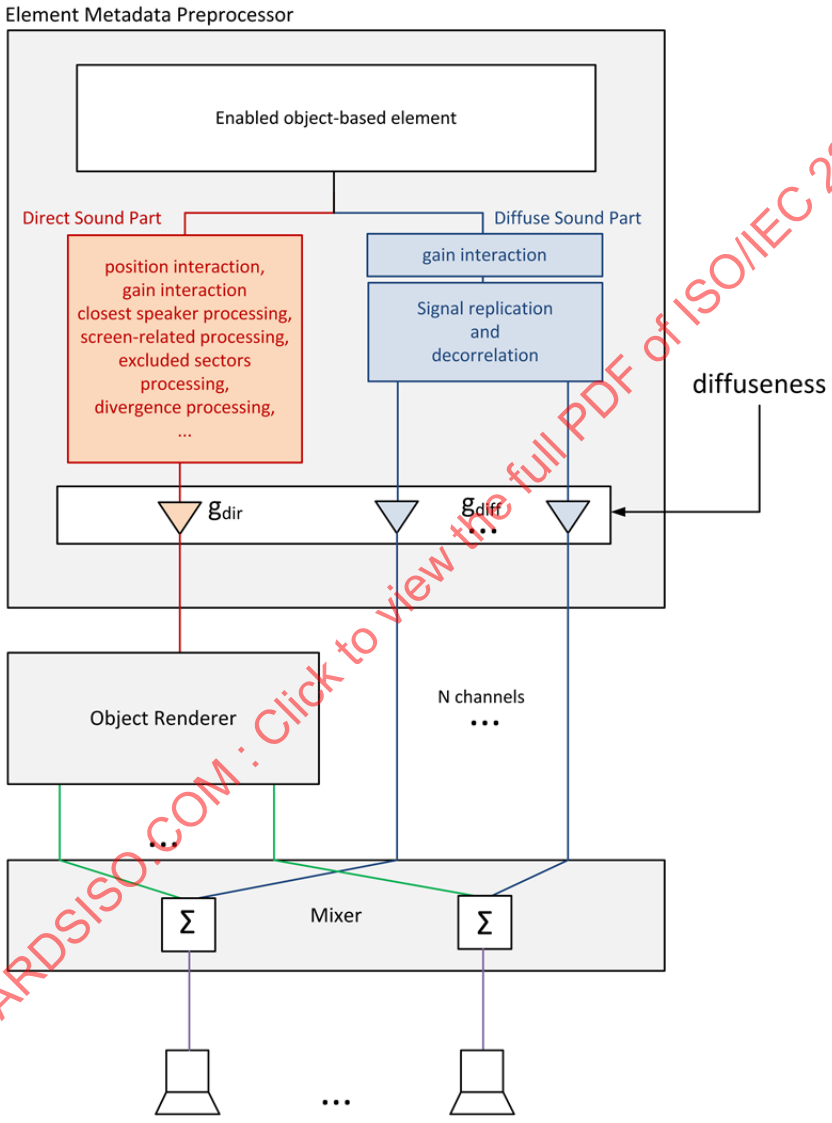


Figure 106 — Diffuseness processing

19 MPEG-H 3D audio profile definition

ISO/IEC 23008-3:2015 (the first edition of this document) captured the definition of the main profile, its associated bitstream syntax, semantics, and decoding process description.

20 Carriage of MPEG-H 3D audio in ISO base media file format

20.1 General

This clause specifies the carriage of MPEG-H 3D audio in the ISO base media file format. Subclause 20.2 describes the signalling of random access points for immediate play-out frames (IPF) and independently decodable frames (IF). Subclause 20.7 describes the additional signalling of dynamic range control and loudness information that might be present in the encoded bitstream. Subclause 20.9 describes the additional signalling of audio scene information data that might be present in the encoded bitstream. If the exact original length (duration) of an input is to be retained throughout the encode/decode process, then this shall be done in accordance with Annex N.

20.2 Random access and stream access

Frames that use AudioPreRoll() following the restrictions in subclause 5.5.6 are considered to be immediate play-out frames (IPF) and shall be signalled as sync samples according to ISO/IEC 14496-12.

If independently decodable frames (IF) as described in subclause 5.7 are to be signalled, they shall be signalled by means of the AudioPreRollEntry according to ISO/IEC 14496-12.

20.3 Overview of new box structures

mha1, mha2, mhm1, mhm2			*	*sample entry*
	mhaC			*configuration*
	mhaD			*dynamic range and loudness*
	maeM			*multi-stream*
	maeI			*audio scene information*
		maeG	*	*group definition*
		maeS		*switch group definition*
		maeP		*preset definition*
		maeL		*text label definition*

20.4 MHA decoder configuration record

20.4.1 Definition

This clause specifies the decoder configuration information for MPEG-H 3D audio (MHA) content.

This record contains a version field. This version of the specification defines version 1 of this record. Incompatible changes to the record will be indicated by a change of version number. Decoders shall not attempt to decode this record or the streams to which it applies if the version number is unrecognized.

20.4.2 Syntax

```
aligned(8) class MHADecoderConfigurationRecord {
   unsigned int(8)   configurationVersion = 1;
   unsigned int(8)   mpegh3daProfileLevelIndication;
   unsigned int(8)   referenceChannelLayout;
   unsigned int(16)  mpegh3daConfigLength;
   bit(8*mpegh3daConfigLength) mpegh3daConfig;
}
```

20.4.3 Semantics

`configurationVersion`	shall be set to 1 in this version of the specification.
`mpegh3daProfileLevelIndication`	defined in subclause 5.2.2.
`referenceChannelLayout`	ChannelConfiguration value in accordance with ISO/IEC 23001-8.
`mpegh3daConfigLength`	length in bytes of mpegh3daConfig.
`mpegh3daConfig`	the MPEG-H 3D audio configuration defined in this document.

20.5 MPEG-H audio sample entry

20.5.1 Definition

Box Types: `'mhaC'`, `'mha1'`, `'mha2'`

Container: Sample Table Box (`'stbl'`)

Mandatory: The mha1 box is mandatory

Quantity: One or more sample entries may be present

The `MHASampleEntry` shall contain a `MHAConfigurationBox`, as defined below. This includes the `MHADecoderConfigurationRecord` as defined in subclause 20.4. If the sample entry type is `'mha1'`, multiple streams shall not be used. If the sample entry name is `'mha2'`, multiple streams may be used.

If an `'mha1'` or `'mha2'` `MHASampleEntry` is present, each sample of the appropriate track shall contain exactly one mpegh3daFrame as defined in this document. An optional `MPEG4BitRateBox` may be present in the `MHASampleEntry` to signal the bit rate information of the MHA stream. Extension descriptors that should be inserted into the elementary stream descriptor, when used in MPEG-4, may also be present. Other boxes may be present in the `MHASampleEntry`. When multiple streams are used, the `MHADecoderConfigurationRecord` for each track shall correspond to the appropriate mpegh3daFrame of that track.

The following optional boxes inherited from `AudioSampleEntry` from ISO/IEC 14496-12/Amd 4:2015 shall not be present.

— `DownMixInstructions()`

— `DRCCoefficientsBasic()`

— DRCInstructionsBasic()

— DRCCoefficientsUniDRC()

— DRCInstructionsUniDRC()

20.5.2 Syntax

```
class MHAConfigurationBox() extends Box('mhaC') {
   MHADecoderConfigurationRecord MHAConfig;
}
class MPEG4BitRateBox() extends Box('btrt') {
   unsigned int(32)  bufferSizeDB;
   unsigned int(32)  maxBitrate;
   unsigned int(32)  avgBitrate;
}
class MPEG4ExtensionDescriptorsBox() extends Box('m4ds') {
   Descriptor Descr[0 .. 255];
}
MHASampleEntry() extends AudioSampleEntry('mha1') {
   MHAConfigurationBox config;
   MPEG4BitRateBox();                // optional
   MPEG4ExtensionDescriptorsBox (); // optional
}
MHASampleEntry() extends AudioSampleEntry('mha2') {
   MHAConfigurationBox config;
   MPEG4BitRateBox();                // optional
   MPEG4ExtensionDescriptorsBox (); // optional
}
```

20.5.3 Semantics

`ChannelCount` inherited from `AudioSampleEntry`, shall be set to 0 (inapplicable) The MPEG-H 3D audio decoder is capable of rendering a scene to any given loudspeaker setup. The `referenceChannelLayout` carried in the `MHADecoderConfigurationRecord` shall be used to signal the preferred reproduction layout for this stream and replaces the `ChannelCount`.

`config` defined in subclause 20.4.

`Descr` is a descriptor which should be placed in in the `ElementaryStreamDescriptor` when this stream is used in an MPEG-4 systems context. This does not include `SLConfigDescriptor` or `DecoderConfigDescriptor`, but includes the other descriptors in order to be placed after the `SLConfigDescriptor`.

`bufferSizeDB` gives the size of the decoding buffer for the elementary stream in bytes.

`maxBitrate` gives the maximum rate in bits/second over any window of 1 second.

`minBitrate` gives the average rate in bits/second over any window of 1 second.

20.6 MPEG-H audio MHAS sample entry

20.6.1 Definition

Box Types: 'mhm1', 'mhm2'

Container: Sample Table Box ('stbl')

Mandatory: No

Quantity: One or more sample entries may be present

Especially in streaming or broadcast environments based on, e.g. MPEG-DASH or MPEG-H MMT, the MPEG-H 3D audio configuration may change at arbitrary positions of the stream and not necessarily only on fragment boundaries. To enable this use-case the 'mhm1' and 'mhm2' MHASampleEntry provides an in-band configuration mechanism for MPEG-H 3D audio files.

If an 'mhm1' or 'mhm2' MHASampleEntry is present, each sample of the appropriate track shall contain exactly one MHAS packet with the MHASPacketType PACTYP_MPEGH3DAFRAME as defined in Clause 14.

A sample may contain additional MHAS Packets of other types: if present, an MHAS packet with MHASPacketType PACTYP_MPEGH3DACFG, PACTYP_AUDIOSCENEINFO or PACTYP_AUDIOTRUNCATION shall directly precede the MHAS packet of type PACTYP_MPEGH3DAFRAME.

MHAS packets with the MHASPacketType PACTYP_CRC16 and PACTYP_CRC32 shall not be present in any sample. Other MHAS packets may be present in a sample.

The first sample of the movie and the first sample of every fragment (when applicable) shall contain a MHAS packet with the type PACTYP_MPEGH3DACFG followed by an MHAS packet with the Type PACTYP_AUDIOSCENEINFO if present.

All samples of the movie that contain an MHAS packet of type PACTYP_MPEGH3DACFG shall be sync samples.

If the movie contains a configuration change, i.e. one of the samples of the movie besides the first sample contains an MHAS packet of type PACTYP_MPEGH3DACFG, all sync samples of the movie shall contain an MHAS packet of type PACTYP_MPEGH3DACFG.

If the sample entry type is 'mhm1', multiple streams shall not be used. If the sample entry name is 'mhm2', multiple streams may be used.

Optional boxes may be present in the MHASampleEntry. Optional boxes for the sample entry type 'mhm1' are handled according to the sample entry type is 'mha1', optional boxes for the sample entry type is 'mhm2' are handled according to the sample entry type is 'mha2'.

In contrast to the sample entry types 'mha1' and 'mha2' the MHAConfigurationBox is optional for the sample entry types 'mhm1' and 'mhm2' and not mandatory.

20.6.2 Syntax

```
MHASampleEntry() extends AudioSampleEntry('mhm1') {
   MHAConfigurationBox config;      // optional
   MPEG4BitRateBox();               // optional
   MPEG4ExtensionDescriptorsBox();  // optional
}
MHASampleEntry() extends AudioSampleEntry('mhm2') {
   MHAConfigurationBox config;      // optional
   MPEG4BitRateBox();               // optional
   MPEG4ExtensionDescriptorsBox();  // optional
}
```

20.7 MHA dynamic range control and loudness

20.7.1 Definition

Box Type: 'mhaD'

Container: MHA sample entry ('mha1', 'mha2', 'mhm1', 'mhm2')

Mandatory: No

Quantity: Zero or one

This box specifies the dynamic range control and loudness information that may be contained in the MPEG-H 3D audio (MHA) track. The provided information represents only a subset of the in-stream configuration according to subclause 6.3.

20.7.2 Syntax

```
aligned(8) class MHADynamicRangeControlAndLoudnessBox()
   extends FullBox('mhaD', version = 0, 0) {
   unsigned int(2)   reserved = 0;
   unsigned int(6)   drcInstructionsUniDrcCount;
   unsigned int(2)   reserved = 0;
   unsigned int(6)   loudnessInfoCount;
   unsigned int(2)   reserved = 0;
   unsigned int(6)   loudnessInfoAlbumCount;
   unsigned int(3)   reserved = 0;
   unsigned int(5)   downmixIdCount;

   for (i=0; i < drcInstructionsUniDrcCount; i++) {
      unsigned int(6)   reserved = 0;
      unsigned int(2)   drcInstructionsType;
      if (drcInstructionsType == 2) {
         unsigned int(1)   reserved = 0;
         unsigned int(7)   mae_groupID;
      }
      if (drcInstructionsType == 3) {
         unsigned int(3)   reserved = 0;
         unsigned int(5)   mae_groupPresetID;
      }
      unsigned int(2)   reserved = 0;
      unsigned int(6)   drcSetId;
```

```
        unsigned int(1)    reserved = 0;
        unsigned int(7)    downmixId;
        unsigned int(5)    reserved = 0;
        unsigned int(3)    additionalDownmixIdCount;
        for (j=0; j < additionalDownmixIdCount; j++) {
            unsigned int(1)    reserved = 0;
            unsigned int(7)    additionalDownmixId;
        }
        unsigned int(16)   drcSetEffect;
        unsigned int(7)    reserved = 0;
        unsigned int(1)    limiterPeakTargetPresent;
        if (limiterPeakTargetPresent == 1) {
            unsigned int(8)    bsLimiterPeakTarget;
        }
        unsigned int(7)    reserved = 0;
        unsigned int(1)    drcSetTargetLoudnessPresent;
        if (drcSetTargetLoudnessPresent == 1) {
            unsigned int(2)    reserved = 0;
            unsigned int(6)    bsDrcSetTargetLoudnessValueUpper;
            unsigned int(2)    reserved = 0;
            unsigned int(6)    bsDrcSetTargetLoudnessValueLower;
        }
        unsigned int(1)    reserved = 0;
        unsigned int(6)    dependsOnDrcSet;
        if (dependsOnDrcSet == 0) {
            unsigned int(1)    noIndependentUse;
        } else {
            unsigned int(1)    reserved = 0;
        }
    }

    for (i=0; i < loudnessInfoCount; i++) {
        unsigned int(6)    reserved = 0;
        unsigned int(2)    loudnessInfoType;
        if (loudnessInfoType == 1 || loudnessInfoType == 2) {
            unsigned int(1)    reserved = 0;
            unsigned int(7)    mae_groupID;
        } else if (loudnessInfoType == 3) {
            unsigned int(3)    reserved = 0;
            unsigned int(5)    mae_groupPresetID;
        }
        LoudnessBaseBox();
    }

    for (i=0; i < loudnessInfoAlbumCount; i++) {
        LoudnessBaseBox();
    }

    for (i=0; i < downmixIdCount; i++) {
        unsigned int(1)    reserved = 0;
        unsigned int(7)    downmixId;
        unsigned int(2)    downmixType;
        unsigned int(6)    CICPspeakerLayoutIdx;
    }
}
```

20.7.3 Semantics

`drcInstructionsUniDrcCount`	number of drcInstructions in the MHA track
`loudnessInfoCount`	number of loudnessInfo blocks in the MHA track
`loudnessInfoAlbumCount`	number of loudnessInfoAlbum blocks in the MHA track
`downmixIdCount`	number of downmixId definitions in the MHA track
`drcInstructionsType`	defined in subclause 6.3 a value of '1' is not defined
`mae_groupID`	defined in subclause 15.3
`mae_groupPresetID`	defined in subclause 15.3
`drcSetId`	defined in ISO/IEC 23003-4:2015, Annex A
`downmixId`	defined in subclause 5.3.5
`additionalDownmixId`	defined in ISO/IEC 23003-4:2015, Annex A
`drcSetEffect`	defined in ISO/IEC 23003-4:2015, Annex A
`bsLimiterPeakTarget`	defined in ISO/IEC 23003-4:2015, Annex A
`bsDrcSetTargetLoudnessValueUpper`	defined in ISO/IEC 23003-4:2015, Annex A
`bsDrcSetTargetLoudnessValueLower`	defined in ISO/IEC 23003-4:2015, Annex A
`dependsOnDrcSet`	defined in ISO/IEC 23003-4:2015, Annex A
`noIndependentUse`	defined in ISO/IEC 23003-4:2015, Annex A
`downmixType`	defined in subclause 5.3.5
`CICPspeakerLayoutIdx`	defined in subclause 5.3.5
`LoudnessBox()`	defined in ISO/IEC 14496-12:2012/Amd.4:2015

20.8 MHA multi-stream signalling

20.8.1 Definition

Box Type: `'maeM'`

Container: MHA sample entry (`'mha1'`, `'mha2'`, `'mhm1'`, `'mhm2'`)

Mandatory: No

Quantity: Zero or one

This box provides information on the location of each `mae_groupID` in case of splitting the audio scene over multiple streams or files. If multiple streams are used, this box shall be present.

20.8.2 Syntax

```
aligned(8) class MHAMultiStreamBox()
   extends FullBox('maeM', version=0, 0) {
   unsigned int(1) isMainStream;
   unsigned int(7) thisStreamID;

   if (isMainStream) {
      unsigned int(1)   reserved = 0;
      unsigned int(7)   mae_numGroups;
      unsigned int(1)   reserved = 0;
      unsigned int(7)   numAuxiliaryStreams;

      for (i=0; i< mae_numGroups; i++) {
         unsigned int(7)   mae_groupID;
         unsigned int(1)   isInMainStream;
         if (!isInMainStream) {
            unsigned int(1)   reserved = 0;
            unsigned int(7)   auxiliaryStreamID;
         }
      }
   }
}
```

20.8.3 Semantics

`isMainStream`	flag indicating if this is the main stream
`thisStreamID`	unique ID of the audio stream in the scope of all available streams in the MHA scene
`mae_numGroups`	total number of groups in the MHA scene. This value can have a value between 1 and 127, a minimum number of 1 and a maximum number of 127 groups. This number shall be equal to `mae_numGroups` in `MHAGroupDefinitionBox()`
`numAuxiliaryStreams`	total number of auxiliary streams available
`mae_groupID`	`mae_groupID` (see subclause 15.3) the loop instance refers to
`isInMainStream`	if this flag is set to 1, the audio data related to the group (indicated by `mae_groupID`) is present in the main stream, otherwise the data is transmitted in an auxiliary stream
`auxiliaryStreamID`	in case the audio data identified by `mae_groupID` is an auxiliary stream, this integer identifies the respective auxiliary stream

20.9 Audio scene information

20.9.1 MHA group definition

20.9.1.1 Definition

Box Type: 'maeG'

Container: MHA scene information ('maeI')

Mandatory: Yes

Quantity: Zero or one

This box provides information about interactivity and priority of groups contained in an MPEG-H 3D audio (MHA) track.

20.9.1.2 Syntax

```
aligned(8) class MHAGroupDefinitionBox()
   extends FullBox('maeG', version = 0, 0) {

   unsigned int(8)   mae_audioSceneID;

   unsigned int(1)   reserved = 0;
   unsigned int(7)   mae_numGroups;
   for (i=0; i < mae_numGroups; i++) {
      unsigned int(1)   reserved = 0;
      unsigned int(7)   mae_groupID;
      unsigned int(3)   reserved = 0;
      unsigned int(1)   mae_allowOnOff;
      unsigned int(1)   mae_defaultOnOff;
      unsigned int(1)   mae_allowPositionInteractivity;
      unsigned int(1)   mae_allowGainInteractivity;
      unsigned int(1)   mae_hasContentLanguage;
      unsigned int(4)   reserved = 0;
      unsigned int(4)   mae_contentKind;

      if (mae_allowPositionInteractivity == 1) {
         unsigned int(1)   reserved = 0;
         unsigned int(7)   mae_interactivityMinAzOffset;
         unsigned int(1)   reserved = 0;
         unsigned int(7)   mae_interactivityMaxAzOffset;
         unsigned int(3)   reserved = 0;
         unsigned int(5)   mae_interactivityMinElOffset;
         unsigned int(3)   reserved = 0;
         unsigned int(5)   mae_interactivityMaxElOffset;
         unsigned int(4)   mae_interactivityMinDistFactor;
         unsigned int(4)   mae_interactivityMaxDistFactor;
      }

      if (mae_allowGainInteractivity == 1) {
         unsigned int(2)   reserved = 0;
         unsigned int(6)   mae_interactivityMinGain;
```

```
            unsigned int(3)    reserved = 0;
            unsigned int(5)    mae_interactivityMaxGain;
        }

        if (mae_hasContentLanguage == 1) {
            unsigned int(8)    mae_contentLanguage;
        }
    }
}
```

20.9.1.3 Semantics

mae_audioSceneID	defined in subclause 15.3
mae_numGroups	defined in subclause 15.3
mae_groupID	defined in subclause 15.3
mae_allowOnOff	defined in subclause 15.3
mae_defaultOnOff	defined in subclause 15.3
mae_allowPositionInteractivity	defined in subclause 15.3
mae_allowGainInteractivity	defined in subclause 15.3
mae_hasContentLanguage	defined in subclause 15.3
mae_contentKind	defined in subclause 15.3
mae_interactivityMinAzOffset	defined in subclause 15.3
mae_interactivityMaxAzOffset	defined in subclause 15.3
mae_interactivityMinElOffset	defined in subclause 15.3
mae_interactivityMaxElOffset	defined in subclause 15.3
mae_interactivityMinDistFactor	defined in subclause 15.3
mae_interactivityMaxDistFactor	defined in subclause 15.3
mae_interactivityMinGain	defined in subclause 15.3
mae_interactivityMaxGain	defined in subclause 15.3
mae_ContentLanguage	defined in subclause 15.3

20.9.2 MHA switch group definition

20.9.2.1 Definition

Box Type: 'maeS'

Container: MHA scene information ('maeI')

Mandatory: No

Quantity: Zero or one

20.9.2.2 Syntax

```
aligned(8) class MHASwitchGroupDefinitionBox()
   extends FullBox('maeS', version = 0, 0) {

   unsigned int(3) reserved = 0;
   unsigned int(5) mae_numSwitchGroups;
   for (i=0; i < mae_numSwitchGroups; i++) {
      unsigned int(3) reserved = 0;
      unsigned int(5) mae_switchGoupID;

      unsigned int(3) reserved = 0;
      unsigned int(5) mae_bsSwitchGroupNumMembers;
      for (j=0; j < mae_bsSwitchGroupNumMembers; j++) {
         unsigned int(1) reserved = 0;
         unsigned int(7) mae_switchGroupMemberID;
      }

      unsigned int(1) reserved = 0;
      unsigned int(7) mae_switchGroupDefaultGroupID;
   }
}
```

20.9.2.3 Semantics

mae_numSwitchGroups	defined in subclause 15.3
mae_switchGroupID	defined in subclause 15.3
mae_switchGroupAllowOnOff	defined in subclause 15.3
mae_switchGroupDefaultOnOff	defined in subclause 15.3 If mae_switchGroupAllowOnOff is 0, then mae_switchGroupDefaultOnOff shall be 0
mae_bsSwitchGroupNumMembers	defined in subclause 15.3
mae_switchGroupMemberID	defined in subclause 15.3
mae_switchGroupDefaultGroupID	defined in subclause 15.3

20.9.3 MHA group preset definition

20.9.3.1 Definition

Box Type: 'maeP'

Container: MHA scene information ('maeI')

Mandatory: No

Quantity: Zero or one

This box provides information about group presets contained in an MPEG-H 3D audio (MHA) track. A preset is a collection of groups which cover a common use-case. In addition, a preset can be used to enable advanced DRC for audio object scenes.

20.9.3.2 Syntax

```
aligned(8) class MHAGroupPresetDefinitionBox()
   extends FullBox('maeP', version=0, 0) {

   unsigned int(3)   reserved = 0;
   unsigned int(5)   mae_numGroupPresets;
   for (i=0; i < mae_numGroupPresets; i++) {
      unsigned int(3)   reserved = 0;
      unsigned int(5)   mae_groupPresetID;
      unsigned int(3)   reserved = 0;
      unsigned int(5)   mae_groupPresetKind;

      unsigned int(8)   numGroupPresetConditions;
      for (j=0; j < numGroupPresetConditions; j++) {
         unsigned int(7) mae_groupPresetGroupID;
         unsigned int(1) mae_groupPresetConditionOnOff;
         if (mae_groupPresetConditionOnOff == 1) {
            unsigned int(4)   reserved = 0;
            unsigned int(1)   mae_groupPresetDisableGainInteractivity;
            unsigned int(1)   mae_groupPresetGainFlag;
            unsigned int(1)
mae_groupPresetDisablePositionInteractivity;
            unsigned int(1)   mae_groupPresetPositionFlag;
            if (mae_groupPresetGainFlag == 1) {
               unsigned int(8)   mae_groupPresetGain;
            }
            if (mae_groupPresetPositionFlag == 1) {
               unsigned int(8)   mae_groupPresetAzOffset;
               unsigned int(8)   mae_groupPresetElOffset;
               unsigned int(8)   mae_groupPresetDistFactor;
            }
         }
      }
   }
}
```

20.9.3.3 Semantics

`mae_numGroupPresets`	defined in subclause 15.3
`mae_groupPresetID`	defined in subclause 15.3
`mae_groupPresetKind`	defined in subclause 15.3
`numGroupPresetConditions`	number of group preset conditions
`mae_groupPresetGroupID`	defined in subclause 15.3
`mae_groupPresetConditionOnOff`	defined in subclause 15.3
`mae_groupPresetDisableGainInteractivity`	defined in subclause 15.3
`mae_groupPresetGainFlag`	defined in subclause 15.3
`mae_groupPresetDisablePositionInteractivity`	defined in subclause 15.3
`mae_groupPresetPositionFlag`	defined in subclause 15.3
`mae_groupPresetGain`	defined in subclause 15.3
`mae_groupPresetAzOffset`	defined in subclause 15.3
`mae_groupPresetElOffset`	defined in subclause 15.3
`mae_groupPresetDistFactor`	defined in subclause 15.3

20.9.4 MHA group description text label

20.9.4.1 Definition

Box Type: `'maeL'`

Container: MHA scene information (`'mael'`)

Mandatory: No

Quantity: Zero or one

20.9.4.2 Syntax

```
aligned(8) class MHAGroupDescrTextLabelBox()
   extends FullBox('maeL', version = 0, 0) {

   unsigned int(4)    reserved = 0;
   unsigned int(4)    numDescLanguage;
   for (j=0; j < numDescLanguage; j++) {
      unsinged int(24)    descriptionLanguage;

      unsigned int(1)    reserved = 0;
      unsigned int(7)    numGroupDescriptions;
```

```
        for (i=0; i < numGroupDescriptions; i++) {
           unsigned int(1)    reserved = 0;
           unsigned int(7)    mae_descritptionGroupID;

           unsigned int(8)    groupDescriptionDataLength;
           for (c=0; c < groupDescriptionDataLength; c++) {
              groupDescriptionData;
           }
        }

        unsigned int(3)    reserved = 0;
        unsigned int(5)    numSwitchGroupDescriptions;
        for (i=0; i < numSwitchGroupDescriptions; i++) {
           unsigned int(3)    reserved = 0;
           unsigned int(5)    mae_descriptionSwitchGroupID;

           unsigned int(8)    switchGroupDescriptionDataLength;
           for (c=0; c < switchGroupDescriptionDataLength; c++) {
              switchGroupDescriptionData;
           }
        }

        unsigned int(3)    reserved = 0;
        unsigned int(5)    numGroupPresets;
        for (i=0; i < numGroupPresets; i++) {
           unsigned int(3)    reserved = 0;
           unsigned int(5)    mae_descriptionGroupPresetID;

           unsigned int(8)    groupPresetDescriptionDataLength;
           for (c=0; c < groupPresetDescriptionDataLength; c++) {
              groupPresetDescriptionData;
           }
        }
     }
}
```

20.9.4.3 Sematics

`numDescLanguage` number of available languages

`descriptionLanguage` language of the description text

The field contains a 3-character code as specified in ISO 639-2. Both ISO 639-2/B and ISO 639-2/T may be used. Each character is coded into 8 bits according to ISO/IEC 8859-1 and inserted in order into the 24-bit field.

EXAMPLE French has 3-character code "fre", which is coded as: "0110 0110 0111 0010 0110 0101".

`numGroupDescription` number of group definition blocks

`mae_descriptionGroupID` defined in subclause 15.3

`groupDescriptionDataLength` length in bytes of `groupDescriptionData`

`groupDescriptionData` description of a group

A string describing the content by a high-level description. The format shall follow UTF-8 encoding according to ISO/IEC 10646.

`numSwitchGroupDescriptions` number of switch group definition blocks

`mae_descriptionSwitchGroupID` defined in subclause 15.3

`switchGroupDescriptionDataLength` length in bytes of `switchGroupDescriptionData`

`switchGroupDescriptionData` description of a switch group

A string describing the content by a high-level description. The format shall follow UTF-8 encoding according to ISO/IEC 10646.

`numGroupPresets` number of group preset definition blocks

`mae_descriptionGroupPresetID` defined in subclause 15.3

`groupPresetDescriptionDataLength` length in bytes of `groupPresetDescriptionData`

`groupPresetDescriptionData` description of a group preset

A string describing the content by a high-level description. The format shall follow UTF-8 encoding according to ISO/IEC 10646.

20.9.5 MHA scene information

20.9.5.1 Definition

Box Type: `'maeI'`

Container: MHA sample entry (`'mha1'`, `'mha2'`, `'mhm1'`, `'mhm2'`)

Mandatory: No

Quantity: Zero or one

20.9.5.2 Syntax

```
class MHASceneInformationBox() extends Box('maeI') {
   MHAGroupDefinitionBox          group;
   MHASwitchGroupDefinitionBox    switchGroup;  // optional
   MHAGroupPresetDefinitionBox    preset;       // optional
   MHAGroupDescrTextLabelBox      label;        // optional
}
```

20.9.5.3 Semantics

`groups` defined in subclause 20.9.1

`switchGroup` defined in subclause 20.9.2

`preset`	defined in subclause 20.9.3
`label`	defined in subclause 20.9.4
`multiStream`	defined in subclause 20.8

20.10 Track references

If multiple streams are used, the track containing the main stream, as indicated by `isMainStream = 1` in the `MHAMultiStreamBox`, shall have an `'maux'` track reference to all associated auxiliary streams that are contained as individual tracks in the same file. All auxiliary streams that are contained as individual tracks in the same file, indicated by `isMainStream = 0` in the `MHAMultiStreamBox`, shall have an `'mbas'` track reference to the respective main stream.

21 Sub-parameters for the MIME type 'Codecs' parameter

21.1 General

When the 'codecs' parameter of a MIME type is used, as defined in RFC 6381, subclause 21.2 documents the sub-parameter when the MIME type identifies the file format of this codec and the 'codecs' parameter starts with a sample-entry code from this specification.

21.2 'Codecs' parameter for MPEG-H 3D audio

When the first element of a value is a code indicating a codec from this specification, as documented in clauses above, such as 'mha1', 'mha2', 'mhm1' or 'mhm2' – indicating MPEG-H 3D audio, the second element is the **mpegh3daProfileLevelIndication** defined in Table 64 (subclause 5.3.2).

22 Timing considerations and decoder behaviour

An access unit is associated with the composition time of the first sample of the respective encoded PCM data. The decoder shall not output any additional samples that might have been produced by initializing, e.g. filter-banks or renderers.

An example decoder might accept multiple access units before delivering the decoded and rendered PCM samples (composition unit). Timing information needs to be calculated accordingly, i.e. re-associated with the composition unit. After the last access unit, it might be necessary to flush the decoder to retrieve remaining PCM samples. The last frame might be shorter than the current granule-length. This may be addressed by using an MHAS packet of PACTYP_AUDIOTRUNCATION. In case of ISO Base Media File Format, the same can be achieved by having a separate entry in the TimeToSampleBox ('stts') for the last access unit indicating a shorter duration (sample_delta).

23 Multi-stream handling

In multi-stream scenarios the main and side-streams are determined by the mae_isMainStream flag as defined in Clause 15.

23.1 Restrictions on extension payloads

In multi-stream scenarios extension payloads may be present in both main and side-stream(s). The restrictions as defined in Table 272 and Table 273 shall apply.

Table 272 — Allowed configuration extensions in main- and side-stream(s)

usacConfigExtType	Main-stream	Side-stream(s)
ID_CONFIG_EXT_DOWNMIX	One element for all present signal groups of type SignalGroupTypeChannels.	One element for all present signal groups of type SignalGroupTypeChannels.
ID_CONFIG_EXT_LOUDNESS_INFO	One element that comprises group specific metadata (loudnessInfoType=1) for present signal groups and full scene metadata including side-stream(s) (loudnessInfoType=0/2/3).	One element that comprises group specific metadata (loudnessInfoType=1) for present signal groups only.
ID_CONFIG_EXT_AUDIOSCENE_INFO	One element with mae_isMainStream=1	One element with mae_isMainStream=0
ID_CONFIG_EXT_HOA_MATRIX	One element for each present signal group of type SignalGroupTypeHOA	One element for each present signal group of type SignalGroupTypeHOA

Table 273 — Allowed bitstream extensions in main- and side-stream(s)

usacExtElementType	Main-stream	Side-stream(s)
ID_EXT_ELE_MPEGS	One element for each present signal group of type SignalGroupTypeChannels.	One element for each present signal group of type SignalGroupTypeChannels.
ID_EXT_ELE_AUDIOPREROLL	One element for all present signal groups.	One element for all present signal groups.
ID_EXT_ELE_UNI_DRC	One element that comprises group specific metadata (DRC-1) for present signal groups and full scene metadata including side-stream(s) (DRC-2/3).	One element that comprises group specific metadata (DRC-1) for present signal groups only.
ID_EXT_ELE_OBJ_METADATA	One element for each present signal group of type SignalGroupTypeObject.	One element for each present signal group of type SignalGroupTypeObject.
ID_EXT_ELE_SAOC_3D	One element for each present signal group of type SignalGroupTypeSAOC.	One element for each present signal group of type SignalGroupTypeSAOC.
ID_EXT_ELE_HOA	One element for each present signal group of type SignalGroupTypeHOA.	One element for each present signal group of type SignalGroupTypeHOA.
ID_EXT_ELE_FMT_CNVRTR	One element for all present signal groups of type SignalGroupTypeChannels. and immersiveDownmixFlag=1.	One element for all present signal groups of type SignalGroupTypeChannels. and immersiveDownmixFlag=1.

24 Low complexity generic loudspeaker rendering/format conversion

24.1 Description

The loudspeaker renderer converts multichannel signals from transmitted channel configurations to desired reproduction formats. It is thus also called 'format converter'. If the channel configuration of the channels routed to the format converter exactly matches the signalled reproduction layout (i.e. the target channel configuration), the format converter shall be bypassed. The system consists of two major building blocks

— An initialization algorithm that takes into account static parameters like the input and output format.

— A signal adaptive downmixing process that operates in a subband domain.

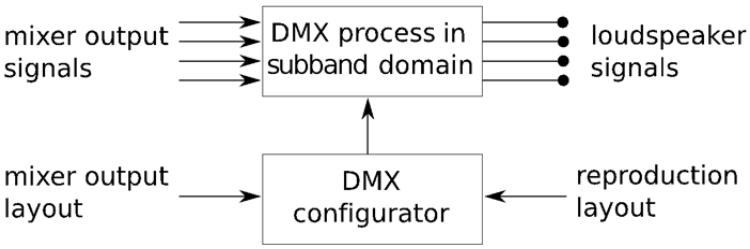


Figure 107 — Main building blocks of the low complexity generic format converter

In the initialization phase the format converter automatically generates optimized downmixing parameters (like the downmixing matrix) for the given combination of input and output formats. It applies an algorithm that selects for each input loudspeaker the most appropriate mapping rule from a list of rules that has been designed to incorporate psychoacoustic considerations. Each rule describes the mapping from one input channel to one or several output loudspeaker channels.

Input channels are

— either mapped to a single output channel,

— or panned to two output channels,

— or (in case of the 'Voice of God' channel) distributed over a larger number of output channels.

The optimal mapping for each input channel is selected depending on the list of output loudspeakers that are available in the desired output format. Each mapping defines downmix gains for the input channel under consideration as well as potentially also an equalizer that is applied to the input channel under consideration.

Output setups with non-standard loudspeaker positions can be signalled to the system by providing the azimuth and elevation deviations from a regular loudspeaker setup.

The actual downmixing of the audio signals is performed on a short time Fourier transform (STFT) representation of the signals. The energy-preserving algorithm avoids signal deteriorations like comb-filtering, coloration, or modulation artifacts.

24.2 Definitions

24.2.1 General remarks

Audio signals that are fed into the format converter are referred to as *input signals* in the following. Audio signals that are the result of the format conversion process are referred to as *output signals*. Note that the audio input signals of the format converter are audio output signals of the core decoder.

Vectors and matrices are denoted by bold-faced symbols. Vector elements or matrix elements are denoted as italic variables supplemented by indices indicating the row/column of the vector/matrix element in the vector/matrix, e.g. $[y_1 \cdots y_a \cdots y_N] = \mathbf{y}$ denotes a vector and its elements. Similarly, $M_{a,b}$ denotes the element in the ath row and bth column of a matrix $\mathbf{M}$.

24.2.2 Variable definitions

N_{in}	Number of channels in the input channel configuration.
N_{out}	Number of channels in the output channel configuration.
$\mathbf{M}_{DMX}$	Downmix matrix containing real-valued non-negative downmix coefficients (downmix gains). $\mathbf{M}_{DMX}$ is of dimension ($N_{out} \times N_{in}$).
$\mathbf{G}_{EQ}$	Matrix consisting of gain values per processing band determining frequency responses of equalizing filters.
$\mathbf{I}_{EQ}$	Vector signalling which equalizer filters to apply to the input channels (if any).
L	Frame length measured in time domain audio samples.
ν	Time domain sample index.
F	Frame index (frame number).
PB	Number of processing band, PB=58.
pb	Processing band index ($0 \le pb < PB$).
N	DFT length
K	Number of STFT frequency bins, K = 257.
k	STFT frequency bin index ($0 \le k < K$).
α	Filter parameter, $\alpha = 0.0435$.
A, B	Channel indices.
eps	Numerical constant, $\text{eps} = 10^{-35}$.

24.3 Processing

24.3.1 Application of transmitted downmix matrices

24.3.1.1 General

MPEG-H 3D audio allows transmission of downmix specifications for specific target channel configurations. downmixIds are assigned to the transmitted downmix specifications, allowing DRC to adapt to the downmix specification applied in the MPEG-H decoder (e.g. to select an appropriate DRC gain sequence). Further, loudness metadata values may be coupled with downmixIds. downmixIds are transmitted in the bitstream together with the downmixType as well as the nominal loudspeaker layouts the embedded downmix matrices (and/or DRC and loudness data) have been designed for.

In the MPEG-H 3D audio decoder the selection of a downmixIds thus:

— determines whether transmitted downmix coefficients (downmixType=1) or decoder side generated downmix coefficients (downmixType=0) are applied in the downmix process;

— influences DRC/loudness processing.

Two arguments speak in favor of applying transmitted downmix coefficients also for moderately displaced reproduction layouts.

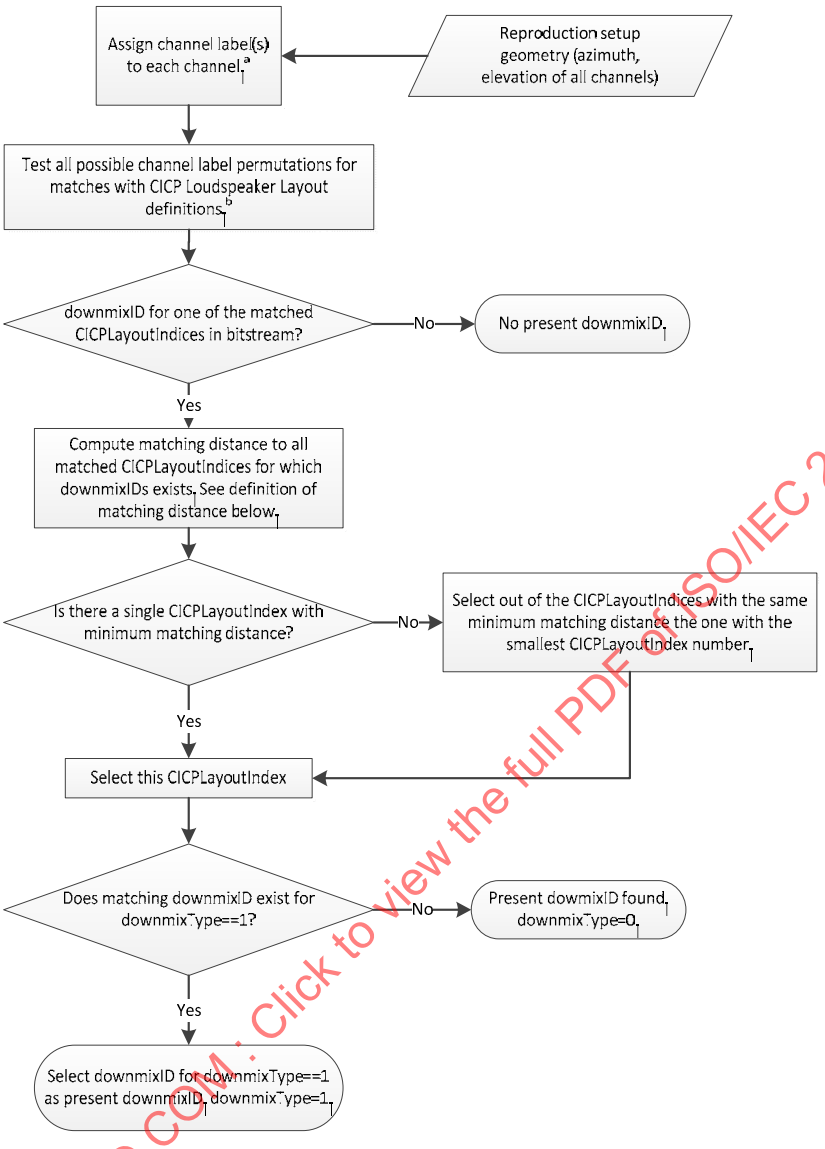
— The selection of a significantly different downmix matrix (transmitted vs. decoder generated) results in large perceptual changes of the downmix result.

— The artistic intent of transmitted downmix coefficients would be lost if a transmitted downmix matrix is not applied.

Of course, in case the loudspeaker displacements of the reproduction setup are too large, the application of the transmitted downmix coefficients may result in a larger perceptual distance from the intended reproduction than the application of decoder generated downmix coefficients. As a consequence, the allowed loudspeaker displacement values are restricted in the following matching scheme.

24.3.1.2 Loudspeaker layout matching scheme

The downmixId matching algorithm is specified by the flow chart of Figure 108. It takes as input the geometry of the actual reproduction setup and outputs as result the present downmixId (if any). The present downmixId determines the downmix processing in the decoder as shown in subclause 24.3.1.4. Further, the present downmixId may affect DRC and loudness processing.



[a] Channel labels shall be assigned according to Table 274.

[b] CICP Loudspeaker layout definitions according to Table 279.

Figure 108 — Flow diagram for loudspeaker layout matching and to determine downmixId

The *matching distance* is defined as the sum of all absolute azimuth and elevation angle differences between the channel positions of the reproduction layout and the tested CICP loudspeaker layout, summed over all channels of the reproduction setup, excluding the LFE channels:

$$d_{\text{matching}} = \sum_{ch \in \left\{ \begin{matrix} \text{all channels} \\ \text{except LFEs} \end{matrix} \right\}} \left| \varphi_{\text{CICPLayout},ch} - \varphi_{\text{reproductionLayout},ch} \right| + \left| \vartheta_{\text{CICPLayout},ch} - \vartheta_{\text{reproductionLayout},ch} \right|$$

where φ denotes azimuth angles and ϑ denotes elevation angles.

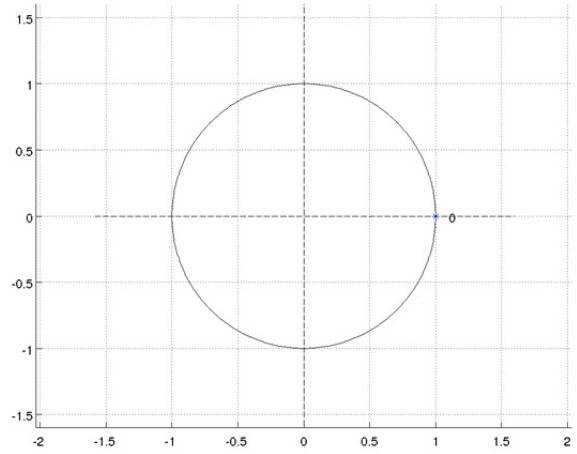
Table 274 — Channel matching tolerances for matching downmixIds to reproduction layouts

Loudspeaker Geometry as defined in ISO/IEC 23001-8)	Channel	Azimuth [deg]	Elevation [deg]	Azimuth start angle of sector [deg]	Azimuth end angle of sector [deg]	Elevation start angle of sector [deg]	Elevation end angle of sector [deg]	Ch. is LFE	Position is relative
	CH_EMPTY	n/a	n/a	n/a	n/a	n/a	n/a	0	0
0	CH_M_L030	30	0	15	45	-15	15	0	0
1	CH_M_R030	-30	0	-45	-15	-15	15	0	0
2	CH_M_000	0	0	-10	10	-15	15	0	0
3	CH_LFE1	0	n/a	n/a	n/a	n/a	n/a	1	0
4	CH_M_L110	110	0	90	130	-15	15	0	0
5	CH_M_R110	-110	0	-130	-90	-15	15	0	0
6	CH_M_L022	22	0	7	37	-15	15	0	0
7	CH_M_R022	-22	0	-37	-7	-15	15	0	0
8	CH_M_L135	135	0	120	150	-15	15	0	0
9	CH_M_R135	-135	0	-150	-120	-15	15	0	0
10	CH_M_180	180	0	170	190	-15	15	0	0
13	CH_M_L090	90	0	70	110	-15	15	0	0
14	CH_M_R090	-90	0	-110	-70	-15	15	0	0
15	CH_M_L060	60	0	40	80	-15	15	0	0
16	CH_M_R060	-60	0	-80	-40	-15	15	0	0
17	CH_U_L030	30	35	15	45	15	55	0	0
18	CH_U_R030	-30	35	-45	-15	15	55	0	0
19	CH_U_000	0	35	-15	15	15	55	0	0
20	CH_U_L135	135	35	115	155	15	55	0	0
21	CH_U_R135	-135	35	-155	-115	15	55	0	0
22	CH_U_180	180	35	165	195	15	55	0	0
23	CH_U_L090	90	35	70	110	15	55	0	0
24	CH_U_R090	-90	35	-110	-70	15	55	0	0
25	CH_T_000	0	90	-180	180	60	90	0	0
26	CH_LFE2	45	n/a	n/a	n/a	n/a	n/a	1	0
27	CH_L_L045	45	-15	25	65	-40	0	0	0
28	CH_L_R045	-45	-15	-65	-25	-40	0	0	0
29	CH_L_000	0	-15	-15	15	-40	0	0	0
30	CH_U_L110	110	35	90	130	15	55	0	0
31	CH_U_R110	-110	35	-130	-90	15	55	0	0
32	CH_U_L045	45	35	30	60	15	55	0	0
33	CH_U_R045	-45	35	-60	-30	15	55	0	0
34	CH_M_L045	45	0	30	60	-15	15	0	0
35	CH_M_R045	-45	0	-60	-30	-15	15	0	0
36	CH_LFE3	-45	n/a	n/a	n/a	n/a	n/a	1	0
37	CH_M_LSCR	60	0	15	80	-15	15	0	1
38	CH_M_RSCR	-60	0	-80	-15	-15	15	0	1
39	CH_M_LSCH	30	0	7	40	-15	15	0	1
40	CH_M_RSCH	-30	0	-40	-7	-15	15	0	1
41	CH_M_L150	150	0	135	165	-15	15	0	0
42	CH_M_R150	-150	0	-165	-135	-15	15	0	0

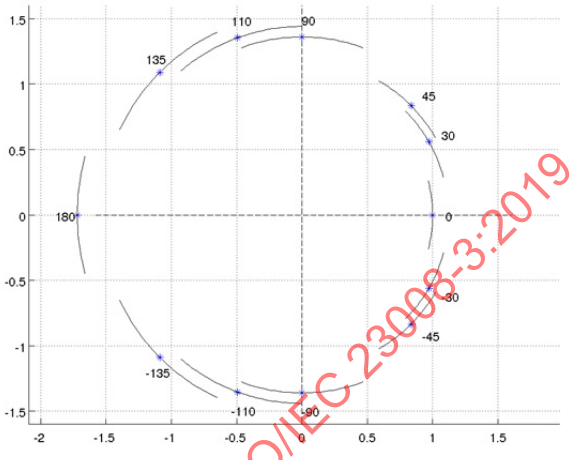
NOTE Azimuth and elevation tolerance intervals are defined as sectors, where azimuth start and end values are connected in counterclockwise direction and elevation start and end values are connected in ascending elevations. Start and end values of the sectors are considered part of the sectors.

24.3.1.3 Visualization of azimuth tolerances

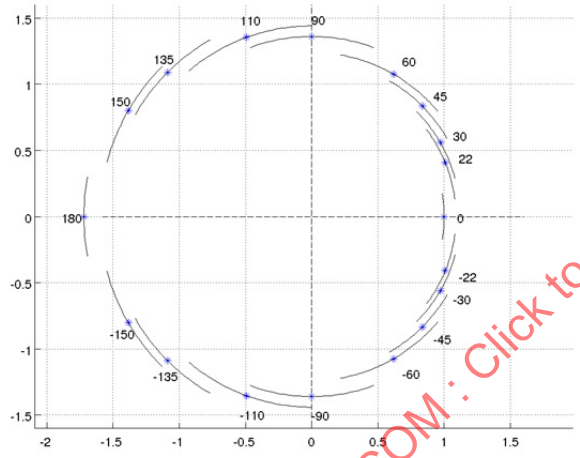
The following figures depict the azimuth matching sectors for each loudspeaker. The tolerances reflect the non-isotropic sound localization performance of the human auditory system.



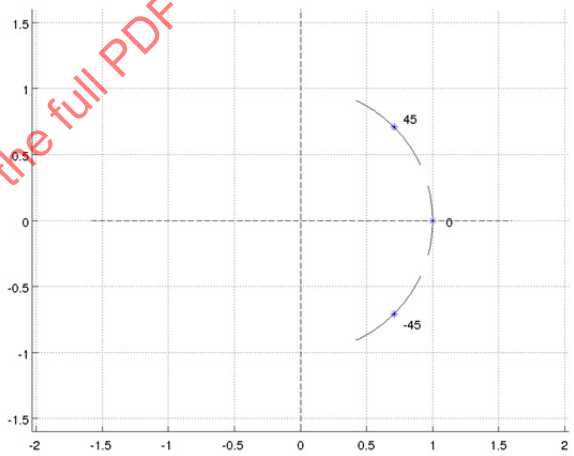
a) Top layer



b) Upper layer



c) Middle layer



d) Lower/bottom layer

NOTE Arcs have been plotted on different radii just for clarity of presentation. Nominal positions of the channels have been marked with asterisks and labels indicating the nominal azimuth angle in degrees.

Figure 109 — Visualization of azimuth tolerances

24.3.1.4 Determination of downmix processing depending on present downmixId

The downmix processing in the MPEG-H decoder is determined by the present downmixId as follows.

— If there is no present downmixId for the current reproduction setup, the downmix coefficients shall be derived as specified in the format converter initialization.

— If downmixType==0 for the present downmixId, the downmix coefficients shall be derived as specified in the format converter initialization.

— If downmixType==1 for the present downmixId, the downmix coefficients transmitted with the downmixId shall be applied in the downmix.

24.3.2 Application of transmitted equalizer settings

Equalizer settings may be transmitted together with downmix matrices, as indicated by the **equalizerPresent** bitstream element. In case equalizer settings have been transmitted together with a downmix matrix that is applied in the format converter, the equalizers shall be applied to this downmix matrix as follows.

The transmitted equalizer parameters shall be decoded into frequency dependent gains (i.e. into equalizer frequency responses) according to subclause 24.3.4.6.4. Next, the transmitted downmix matrix gains shall be multiplied by the frequency dependent equalizer gains to arrive at the final frequency dependent downmix matrix that shall be applied in the downmix. Note that the assignment of equalizer gains to downmix matrix elements is given by the vector **equalizerIndex** that is derived according to Table 31: **equalizerIndex** tells for each input channel whether an equalizer (and if any: which) shall be applied to an input channel by applying the corresponding equalizer gains to all downmix matrix coefficients associated with the respective input channel.

24.3.3 Downmix processing involving multiple channel groups

24.3.3.1 General

In case multiple channel groups are transmitted in the MPEG-H 3D audio bitstream and routed to the format converter, one instance of the format converter shall perform a downmix of all input channels routed to the format converter to the desired target channel configuration. Therefore, all channels routed to the format converter are compiled in a group of channels that constitutes the input to the format conversion process.

If downmix matrices to one or more target channel configurations are transmitted in the bitstream, downmix matrices for those target channel configurations shall be transmitted for all channel groups/channel elements present in the bitstream.

24.3.3.2 Downmix processing with decoder generated downmix gains

In case no appropriate downmix matrices have been transmitted for the signalled target loudspeaker configuration, the downmix gains are generated during the initialization of the format converter according to subclause 24.3.4. The channels are fed to the format converter as a group of all input channels routed to the format converter. The input channel configuration signalled to the format converter shall reflect the channel geometry of this group of channels.

24.3.3.3 Downmix processing with transmitted downmix gains

In case downmix matrices have been transmitted that are applicable to the desired target channel configuration, those downmix matrices shall be applied in the format converter instead of generating downmix gains in the format converter initialization process. Whether a downmix matrix is applicable for a desired target setup, or not, is determined according to subclause 24.3.1.

All channels (and/or groups of channels) shall be compiled in one group of channels, where this group of channels consists of blocks of channels that are the channels (and/or groups of channels) routed to the format converter.

The transmitted downmix matrices assigned to the channels routed to the format converter shall be concatenated according to the order of the blocks of channels in the group of channels that forms the input to the format converter. The concatenated downmix matrix shall then be applied in the format converter to derive the downmixed signal in the desired target channel configuration.

24.3.4 Initialization of the format converter

24.3.4.1 General description of the initialization

The initialization of the format converter is carried out before processing of the audio samples delivered by the core decoder takes place.

The initialization takes into account as input parameters.

— The sampling rate of the audio data to process.

— The channel configuration of the audio data to process with the format converter (number and geometric positions of input channels).

— The channel configuration of the desired output format (number and geometric positions of output channels).

— Optional: Parameters signaling the deviation of the output loudspeaker positions from a standard loudspeaker setup (random setup functionality).

It returns

— A frequency dependent downmix matrix $\mathbf{M}_{DMX}$ that is applied in the audio signal processing of the format converter. $\mathbf{M}_{DMX}$ is also taken into account in the core decoding process, see subclause 5.5.4.1.2.

The input parameters to the initialization algorithm are listed in Table 275.

Table 275 — Format converter initialization input parameters

	Input format: number of channels and nominal channel setup geometry.
	Output format: number of channels and nominal channel setup geometry.
f_s	Sampling frequency in Hertz.
$r_{azi,A}$	For each output channel A, an azimuth angle is specified, determining the deviation from the standard format loudspeaker azimuth.
$r_{ele,A}$	For each output channel A, an elevation angle is specified, determining the deviation from the standard format loudspeaker elevation.

Table 276 lists the output parameters that are derived during the initialization of the format converter.

Table 276 — Format converter initialization output parameters

$\mathbf{M}_{DMX}$	Downmix matrix [linear gains]

24.3.4.2 Assignment of format converter channel labels to input/output format channels

The format converter initialization is based on a system of rules that are defined in terms of *format converter channel labels*, see Table 278. To allow the application of the initialization rules, the channel labels have to be assigned to the channels of the input and output formats. Each format converter channel label is associated with a segment of the surface of the unit sphere, as defined in Table 278. The segments are designed non-overlapping.

The assignment of channel labels to channels is done by geometrically matching the segments to the position data associated with the channels of the input and output formats. The azimuth and elevation

angles in degrees of the position data associated with the channels shall be rounded towards the nearest integer number before performing the channel label assignment. Note that the *nominal* channel positions shall be applied in the following matching to channel label sectors, i.e. the azimuth and elevation angles *without* taking into account potential angle deviations signalled in $r_{azi,A}$ and/or $r_{ele,A}$.

For each channel that is not an LFE (low frequency enhancement) channel.

If the nominal position of the current channel, defined by its azimuth angle and elevation angle, is within or on the border of one of the segments defined in Table 278 then:

— Assign the corresponding channel label (e.g. CH_M_L030) associated with the matching segment.

— Add the angle differences between the nominal position of the current channel and the nominal position associated with the matching segment (i.e. the angles in the second and third column of Table 278) to the angle deviations stored in $r_{azi,A}$ and $r_{ele,A}$.

Else (i.e. no matching sector found), then:

— Assign the CH_EMPTY label.

If an input or output format contains exactly one LFE channel, then the label CH_LFE2 shall be assigned to this channel.

If an input or output format contains exactly two LFE channels, then the labels CH_LFE2 and CH_LFE3 shall be assigned to the two LFE channels in the order that minimizes the maximum azimuth distance from the channels to the assigned CH_LFE2 and CH_LFE3 nominal azimuth positions.

If an input or output format contains more than 2 LFE channels, then those 2 LFE channels out of the considered setup shall be selected that minimize the maximum azimuth distance to the CH_LFE2 and CH_LFE3 nominal azimuth positions. The labels CH_LFE2 and CH_LFE3 shall be assigned as in the case of two LFE channels. The remaining LFE channels shall not be considered further in the calculation of downmix coefficients, i.e. the corresponding lines/columns of the downmix matrix shall remain filled with zeros.

24.3.4.3 Handling for unknown input channels

If the label CH_EMPTY is assigned to an input channel, this channel shall be considered unknown to the rules-based initialization and the downmix coefficients for mapping this input channel to the output channels shall be derived as specified in subclause 24.3.4.6.7.

24.3.4.4 Handling for unknown output formats

If the output format contains at least one channel with the label CH_EMPTY assigned to it, or if at least one channel label is assigned to more than one channel of the output format, the output format shall be considered unknown and the derivation of the downmixing coefficients shall be carried out as specified in subclause 24.3.4.6.7. The rules-based derivation of downmix coefficients shall not be applied for unknown output formats.

24.3.4.5 Handling of deviations from standard loudspeaker positions

If the below conditions are not met, the rules-based initialization is considered to have failed, the output format shall be considered to be unknown, and the downmixing gains shall be obtained as defined in subclause 24.3.4.6.7.

The absolute values of $r_{azi,A}$ and $r_{ele,A}$ shall not exceed 35 and 55 degrees, respectively. The minimum angle between any loudspeaker pair (without LFE channels) shall not be smaller than 15 degrees.

The values of $r_{azi,A}$ shall be such that the ordering by azimuth angles of the horizontal loudspeakers does not change. Likewise, the ordering of the height and low loudspeakers shall not change.

The values of $r_{ele,A}$ shall be such that the ordering by elevation angles of loudspeakers which are (approximately) above/below each other does not change. To verify this, the following procedure is applied:

For each row of Table 284 which contains two or three channels of the output format, do:

— Order the channels by elevation without randomization;

— Order the channels by elevation with considering randomization;

— If the two orderings differ, return an initialization error.

24.3.4.6 Rules-based initialization algorithm

24.3.4.6.1 General

The rules-based initialization algorithm is defined in the following subclauses. The algorithm shall not be applied if the output format is considered unknown as defined in the previous subclause. For clarity the following description makes use of intermediate parameters listed in Table 277 but an implementation may omit the explicit use of these intermediate parameters.

Table 277 — Format converter initialization intermediate parameters

S	Vector of converter source channels [input channel indices]
D	Vector of converter destination channels [output channel indices]
G	Vector of converter gains [linear]
E	Vector of converter EQ indices
$\mathbf{G}_{EQ}$	Matrix containing equalizer gain values for all EQ indices and processing bands

The intermediate parameters describe the dowmixing parameters according to the mapping, i.e. as sets of parameters S_i, D_i, G_i, E_i, per mapping *i*.

The format converter initialization output parameters are derived as described in the following steps.

24.3.4.6.2 Random setups pre-processing

Random output loudspeaker setups, i.e. output setups that contain loudspeakers at positions deviating from the positions defined for the desired output format are signalled by specifying the loudspeaker position deviation angles as input parameters $r_{azi,A}$ and $r_{ele,A}$. The angle deviations are taken into account as a pre-processing step.

Modify in Table 278 the channels' azimuth and elevation angles by adding $r_{azi,A}$ and $r_{ele,A}$ to the corresponding channels' azimuth and elevation angles.

24.3.4.6.3 Derivation of input channel/output channel mapping parameters

The parameters vectors **S**, **D**, **G**, **E** define the mapping of input channels to output channels. For each mapping *i* from an input channel to an output channel with non-zero downmix gain they define the downmix gain as well as an equalizer index that indicates which equalizer curve has to be applied to the input channel under consideration in mapping *i*.

The elements of the parameter vectors **S**, **D**, **G**, **E** are derived by the following algorithm:

Initialize the mapping counter *i*: $i = 1$

For each input channel, ignoring channels with label CH_EMPTY assigned to them:

If the input channel also exists in the output format (e.g. input channel under consideration is CH_M_R030 and channel CH_M_R030 exists in the output format), then:

— S_i = index of source channel in input

EXAMPLE channel CH_M_R030 in ChannelConfiguration 6 is at second place according to Table 279 i.e. has index 2 in this format.

— D_i = index of same channel in output

— $G_i = 1.0$

— $E_i = 0$

— $i = i + 1$

Else (i.e. if the input channel does not exist in the output format)

— search the first entry of this channel in the **Source** column of Table 298, for which the channels in the corresponding row of the **Destination** column exist. The ALL_U destination shall be considered valid (i.e. the relevant output channels exist) if the output format contains at least one "CH_U_" channel. The ALL_M destination shall be considered valid (i.e. the relevant output channels exist) if the output format contains at least one "CH_M_" channel. If for no entry in Table 298 corresponding to the input channel the channels in the **Destination** column exist, the rules-based initialization shall terminate and the downmix gains shall be derived according to subclause 24.3.4.6.7.

— If **Destination** column contains ALL_U, then:

  — For each output channel x with "CH_U_" in its name, do:

    — S_i = index of source channel in input

    — D_i = index of channel x in output

    — G_i = (value of Gain column) / sqrt(number of "CH_U_" output channels)

    — E_i = value of EQ column

  — $i = i + 1$

— Else if **Destination** column contains ALL_M, then:

- For each output channel x with “CH_M_” in its name, do:
  - S_i = index of source channel in input
  - D_i = index of channel x in output
  - G_i = (value of Gain column) / sqrt(number of “CH_M_” output channels)
  - E_i = value of EQ column
  - $i = i + 1$
- Else If there is one channel in the **Destination** column, then:
  - S_i = index of source channel in input
  - D_i = index of destination channel in output
  - G_i = value of Gain column
  - E_i = value of EQ column
  - $i = i + 1$
- Else (two channels in **Destination** column)
  - S_i = index of source channel in input
  - D_i = index of first destination channel in output
  - G_i = (value of Gain column) * g_1
  - E_i = value of EQ column
  - $i = i + 1$
  - $S_i = S_{i-1}$
  - D_i = index of second destination channel in output
  - G_i = (value of Gain column) * g_2
  - $E_i = E_{i-1}$
  - $i = i + 1$
  - The gains g_1 and g_2 are computed by applying tangent law amplitude panning in the following way.
    - Unwrap source destination channel azimuth angles to be positive.
    - The azimuth angles of the destination channels are α_1 and α_2 (see Table 278).
    - The azimuth angle of the source channel (= panning target) is α_{src}.

— $\alpha_0 = \frac{|\alpha_1 - \alpha_2|}{2}$

— $\alpha_{\text{center}} = \frac{\alpha_1 + \alpha_2}{2}$

— $\alpha = (\alpha_{\text{center}} - \alpha_{\text{src}}) \cdot \text{sgn}(\alpha_2 - \alpha_1)$

— $g_1 = \frac{g}{\sqrt{1+g^2}}, \quad g_2 = \frac{1}{\sqrt{1+g^2}} \quad \text{with} \quad g = \frac{\tan\alpha_0 - \tan\alpha + 10^{-10}}{\tan\alpha_0 + \tan\alpha + 10^{-10}}$

24.3.4.6.4 Derivation of equalizer gains G_{EQ}

G_{EQ} consists of gain values per processing band *pb* and equalizer index *e*. The 5 predefined equalizers are combinations of different peak filters. Each equalizer is a serial cascade of one or more peak filters and a gain:

$$G_{EQ,e}^{pb} = 10^{g/20} \prod_{n=1}^{N} peak\left(band(pb)\frac{f_s}{2}, P_{f,n}, P_{Q,n}, P_{g,n}\right)$$

where *band*(*pb*) is the normalized centre frequency of processing band *pb*, specified in Table 281, f_s is the sampling frequency, and function *peak()* is for negative G:

$$\text{peak}(b, f, Q, \text{G}) = \sqrt{\frac{b^4 + \left(\frac{1}{Q^2} - 2\right) f^2 b^2 + f^4}{b^4 + \left(\frac{10^{\frac{-G}{10}}}{Q^2} - 2\right) f^2 b^2 + f^4}}$$

and otherwise:

$$\text{peak}(b, f, Q, \text{G}) = \sqrt{\frac{b^4 + \left(\frac{10^{\frac{G}{10}}}{Q^2} - 2\right) f^2 b^2 + f^4}{b^4 + \left(\frac{1}{Q^2} - 2\right) f^2 b^2 + f^4}}$$

The parameters for the equalizers are specified in Table 283.

24.3.4.6.5 Post-processing for random setups

Once the output parameters are computed, they are modified related to the specific random azimuth and elevations angles. This step has only to be carried out, if not all $r_{ele,A}$ are zero. Definition of the post-processing algorithm.

For each element *i* in D_i, do:

— **if** the output channel with index D_i is a horizontal channel by definition (i.e. output channel label contains the label '_M_'), **and**

  — **if** this output channel is now a height channel (elevation in range 0..60 degrees), **and**

- **if** input channel with index S_i is a height channel (i.e. label contains '_U_'), **then**
  - h = min(elevation of randomized output channel, 35)/35
  - $G_{\text{comp}} = h \cdot \frac{1}{0.85} + (1-h)$
  - Apply compensation gain to DMX gain: $G_i = G_i \cdot G_{\text{comp}}$
  - Define new equalizer with a new index e, where $G_{EQ,e}^{pb} = h + (1-h) \cdot \mathrm{G}_{EQ,E_i}^{pb}$
  - $E_i = e$
- **else if** input channel with index S_i is a horizontal channel (label contains '_M_')
  - h = min(elevation of randomized output channel, 35)/35
  - Define new equalizer with a new index e, where $G_{EQ,e}^{pb} = h \cdot \mathrm{G}_{EQ,5}^{pb} + (1-h) \cdot \mathrm{G}_{EQ,E_i}^{pb}$
  - $E_i = e$

Explanation of the post-processing steps defined above:

h is a normalized elevation parameter indicating the elevation of a nominally horizontal output channel ('_M_') due to a random setup elevation offset $r_{\text{ele},A}$. For zero elevation offset h=0 follows and effectively no post-processing is applied.

The rules table (Table 280) in general applies a gain of 0.85 when mapping an upper input channel ('_U_' in channel label) to one or several horizontal output channels ('_M_' in channel label(s)). In case the output channel gets elevated due to a random setup elevation offset $r_{\text{ele},A}$, the gain of 0.85 is partially (0<h<1) or fully (h=1) compensated for. Similarly the equalizer definitions fade towards a flat EQ-curve ($G_{EQ,e}^{pb} = const. = 1.0$) for h approaching h =1.

In case a horizontal input channel gets mapped to an output channel that gets elevated due to a random setup elevation offset $r_{\text{ele},A}$, the equalizer $G_{EQ,5}^{pb}$ is partially (0<h<1) or fully (h=1) applied.

24.3.4.6.6 Derivation of rules-based initialization downmix matrix

$\mathbf{M}_{\text{DMX}}$ is derived by rearranging the temporary parameters from the mapping-oriented representation (enumerated by mapping counter i) to a channel-oriented representation as defined in the following:

Initialize $\mathbf{M}_{\text{DMX}}^{k}$ as an $N_{\text{out}} \times N_{\text{in}}$ zero matrix for all STFT bins k.

For each i do:

- If (E_i = 0)
  - $M_{\text{DMX},A,B}^{k} = G_i \quad \text{for} \quad A = D_i,\ B = S_i,\ 0 \le k < K$
- Else
  - $M_{\text{DMX},A,B}^{k} = G_i \cdot G_{EQ,E_i}^{pb=pbm(k)} \quad \text{for} \quad A = D_i,\ B = S_i,\ 0 \le k < K$

with the mapping *pbm* between processing bands *pb* and DFT frequency bins as defined in Table 282, and where $M^{k}_{\mathrm{DMX},A,B}$ denotes the matrix element in the *A*th row and *B*th column of $\mathbf{M}^{k}_{\mathrm{DMX}}$. Note that after the rules-based initialization this matrix of downmix coefficients will contain columns of zeros, if unknown channels are present in the input format. Those columns are filled with downmix gains as specified in subclause 24.3.4.6.7.

24.3.4.6.7 VBAP-based downmix coefficients derivation

This subclause defines how to generically derive downmix gains using VBAP in case of unknown output formats or unknown input channels. The following restrictions apply.

— If the target setup contains at least one LFE, then map each LFE channel directly to the LFE of the target setup that minimizes the azimuth angle deviation. No VBAP-based downmix coefficients derivation shall be applied for the LFE channels. The downmix coefficient for the direct mapping shall be set to unity gain, i.e. to 1.0.

— Otherwise apply the VBAP-based downmix coefficients derivation defined in the following also to the LFE channels.

Handling of unknown output formats:

In case the output format is considered unknown, the downmix coefficients for all input channels shall be derived as follows.

Each channel of the input setup is regarded as a static audio object at the position defined by the azimuth and elevation angles associated with the input channel. For each input channel the mixing gains to all output loudspeakers are calculated as VBAP panning gains $\mathbf{g}_{\mathbf{scaled}}$ according to subclause 8.4.4, where the same output format shall be signalled to the VBAP algorithm as to the format converter. The panning gain vectors $\mathbf{g}_{\mathbf{scaled}}$ shall be post-processed according to subclause 24.3.4.6.8.

The downmix matrix $\mathbf{M}^{k}_{\mathrm{DMX}}$ is finally derived by filling each matrix column with the post-processed panning gain vector elements of the corresponding input channel, independently of the DFT bin index k.

Handling of unknown input channels:

In case the input format contains unknown input channels, the downmix coefficients for these channels shall be derived as follows.

Each unknown channel of the input setup is regarded as a static audio object at the position defined by the azimuth and elevation angles associated with the input channel. For each unknown input channel the mixing gains to all output loudspeakers are calculated as VBAP panning gains $\mathbf{g}_{\mathbf{scaled}}$ according to subclause 8.4.4, where the same output format shall be signalled to the VBAP algorithm as to the format converter. The panning gain vectors $\mathbf{g}_{\mathbf{scaled}}$ shall be post-processed according to subclause 24.3.4.6.8.

The downmix matrix $\mathbf{M}^{k}_{\mathrm{DMX}}$ is finally derived by filling each matrix column corresponding to an unknown input channel with the post-processed panning gain vector elements of the corresponding unknown input channel, independently of the DFT bin index k.

24.3.4.6.8 VBAP gains post-processing

The mixing gains obtained from the VBAP rendering algorithm shall be post-processed to avoid excessive use of phantom sources. Therefore, small matrix gains are set to zero, followed by a renormalization of the panning gains to ensure energy-preservation.

For each panning gain vector $\mathbf{g}_{\mathbf{scaled}}$ do:

— If the vector contains at least one panning gain that exceeds the threshold value 0.3, then,

— Set all vector elements smaller or equal to 0.3 to the value 0.0,

— Normalize the gain vector such that the sum of squares of the vector elements remains the same as before the post-processing.

24.3.4.7 Format converter initialization tables

Table 278 lists channel labels, corresponding azimuth and elevation angles, and associated sectors. The sectors are defined as points on the unit sphere, whose azimuth/elevation angles are within or on the borders of the intervals given by the azimuth/elevation start and end values in the table, connecting azimuth start and end values in a counter-clockwise direction and connecting elevation start and end values in the direction of increasing elevation angles.

Table 278 — Channels definitions: Channel labels, corresponding azimuth and elevation angles, and associated sectors

Loudspeaker Geometry as defined in ISO/IEC 23001-8)	**Channel**	**Azimuth [deg]**	**Elevation** [deg]	**Azimuth start angle of sector** [deg]	**Azimuth end angle of sector** [deg]	**Elevation start angle of sector** [deg]	**Elevation end angle of sector** [deg]	**Ch. is LFE**	**Position is relative**
	CH_EMPTY	n/a	n/a	n/a	n/a	n/a	n/a	0	0
0	CH_M_L030	+30	0	+23	+37	-9	+20	0	0
1	CH_M_R030	-30	0	-37	-23	-9	+20	0	0
2	CH_M_000	0	0	-7	+7	-9	+20	0	0
3	CH_LFE1	0	n/a	n/a	n/a	n/a	n/a	1	0
4	CH_M_L110	+110	0	+101	+124	-45	+20	0	0
5	CH_M_R110	-110	0	-124	-101	-45	+20	0	0
6	CH_M_L022	+22	0	+8	+22	-9	+20	0	0
7	CH_M_R022	-22	0	-22	-8	-9	+20	0	0
8	CH_M_L135	+135	0	125	142	-45	+20	0	0
9	CH_M_R135	-135	0	-142	-125	-45	+20	0	0
10	CH_M_180	180	0	158	-158	-45	+20	0	0
13	CH_M_L090	+90	0	+76	+100	-45	+20	0	0
14	CH_M_R090	-90	0	-100	-76	-45	+20	0	0
15	CH_M_L060	+60	0	+53	+75	-9	+20	0	0
16	CH_M_R060	-60	0	-75	-53	-9	+20	0	0
17	CH_U_L030	+30	+35	+11	+37	+21	+60	0	0
18	CH_U_R030	-30	+35	-37	-11	+21	+60	0	0
19	CH_U_000	0	+35	-10	+10	+21	+60	0	0
20	CH_U_L135	+135	+35	+125	+157	+21	+60	0	0
21	CH_U_R135	-135	+35	-157	-125	+21	+60	0	0
22	CH_U_180	180	+35	+158	-158	+21	+60	0	0
23	CH_U_L090	+90	+35	+67	+100	+21	+60	0	0
24	CH_U_R090	-90	+35	-100	-67	+21	+60	0	0
25	CH_T_000	0	+90	-180	+180	+61	+90	0	0
26	CH_LFE2	+45	n/a	n/a	n/a	n/a	n/a	1	0

27	CH_L_L045	+45	-15	+11	+75	-45	-10	0	0
28	CH_L_R045	-45	-15	-75	-11	-45	-10	0	0
29	CH_L_000	0	-15	-10	+10	-45	-10	0	0
30	CH_U_L110	+110	+35	+101	+124	+21	+60	0	0
31	CH_U_R110	-110	+35	-124	-101	+21	+60	0	0
32	CH_U_L045	+45	+35	+38	+66	+21	+60	0	0
33	CH_U_R045	-45	+35	-66	-38	+21	+60	0	0
34	CH_M_L045	+45	0	+38	+52	-9	+20	0	0
35	CH_M_R045	-45	0	-52	-38	-9	+20	0	0
36	CH_LFE3	-45	n/a	n/a	n/a	n/a	n/a	1	0
37	CH_M_LSCR	+60	0	n/a	n/a	n/a	n/a	0	1
38	CH_M_RSCR	-60	0	n/a	n/a	n/a	n/a	0	1
39	CH_M_LSCH	+30	0	n/a	n/a	n/a	n/a	0	1
40	CH_M_RSCH	-30	0	n/a	n/a	n/a	n/a	0	1
41	CH_M_L150	+150	0	143	157	-45	+20	0	0
42	CH_M_R150	-150	0	-157	-143	-45	+20	0	0

Table 279 — Formats with corresponding number of channels and channel ordering

Loudspeaker layout index or ChannelConfiguration as defined in ISO/IEC 23001-8	Number of channels	Channels (with ordering)
1	1	CH_M_000
2	2	CH_M_L030, CH_M_R030
3	3	CH_M_L030, CH_M_R030, CH_M_000
4	4	CH_M_L030, CH_M_R030, CH_M_000, CH_M180
5	5	CH_M_L030, CH_M_R030, CH_M_000, CH_M_L110, CH_M_R110
6	6	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110
7	8	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_M_L060, CH_M_R060
8		n.a.
9	3	CH_M_L030, CH_M_R030, CH_M_180
10	4	CH_M_L030, CH_M_R030, CH_M_L110, CH_M_R110
11	7	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_M_180
12	8	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_M_L135, CH_M_R135
13	24	CH_M_L060, CH_M_R060, CH_M_000, CH_LFE2, CH_M_L135, CH_M_R135, CH_M_L030, CH_M_R030, CH_M_180, CH_LFE3, CH_M_L090, CH_M_R090, CH_U_L045, CH_U_R045, CH_U_000, CH_T_000, CH_U_L135, CH_U_R135, CH_U_L090, CH_U_R090, CH_U_180, CH_L_000, CH_L_L045, CH_L_R045
14	8	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_U_L030, CH_U_R030
15	12	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE2, CH_M_L135, CH_M_R135, CH_LFE3, CH_M_L090, CH_M_R090, CH_U_L045, CH_U_R045, CH_U_180
16	10	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_U_L030, CH_U_R030, CH_U_L110, CH_U_R110
17	12	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_U_L030, CH_U_R030, CH_U_000, CH_U_L110, CH_U_R110, CH_T_000
18	14	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L110, CH_M_R110, CH_M_L150, CH_M_R150, CH_U_L030, CH_U_R030, CH_U_000, CH_U_L110, CH_U_R110, CH_T_000

19	12	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L135, CH_M_R135, CH_M_L090, CH_M_R090, CH_U_L030, CH_U_R030, CH_U_L135, CH_U_R135
20	14	CH_M_L030, CH_M_R030, CH_M_000, CH_LFE1, CH_M_L135, CH_M_R135, CH_M_L090, CH_M_R090, CH_U_L045, CH_U_R045, CH_U_L135, CH_U_R135, CH_M_LSCR, CH_M_RSCR

Table 280 — Converter rules matrix

Source	Destination	Gain	EQ index
CH_M_000	CH_M_L022, CH_M_R022	1.0	0 (off)
CH_M_000	CH_M_L030, CH_M_R030	1.0	0 (off)
CH_M_L022	CH_M_000, CH_M_L030	1.0	0 (off)
CH_M_L022	CH_M_L030	1.0	0 (off)
CH_M_R022	CH_M_000, CH_M_R030	1.0	0 (off)
CH_M_R022	CH_M_R030	1.0	0 (off)
CH_M_L045	CH_M_L030, CH_M_L060	1.0	0 (off)
CH_M_L045	CH_M_L030	1.0	0 (off)
CH_M_R045	CH_M_R030, CH_M_R060	1.0	0 (off)
CH_M_R045	CH_M_R030	1.0	0 (off)
CH_M_L060	CH_M_L045, CH_M_L090	1.0	0 (off)
CH_M_L060	CH_M_L030, CH_M_L090	1.0	0 (off)
CH_M_L060	CH_M_L045, CH_M_L110	1.0	0 (off)
CH_M_L060	CH_M_L030, CH_M_L110	1.0	0 (off)
CH_M_L060	CH_M_L030	0.8	0 (off)
CH_M_R060	CH_M_R045, CH_M_R090,	1.0	0 (off)
CH_M_R060	CH_M_R030, CH_M_R090,	1.0	0 (off)
CH_M_R060	CH_M_R045, CH_M_R110,	1.0	0 (off)
CH_M_R060	CH_M_R030, CH_M_R110,	1.0	0 (off)
CH_M_R060	CH_M_R030,	0.8	0 (off)
CH_M_L090	CH_M_L060, CH_M_L110	1.0	0 (off)
CH_M_L090	CH_M_L045, CH_M_L110	1.0	0 (off)
CH_M_L090	CH_M_L030, CH_M_L110	1.0	0 (off)
CH_M_L090	CH_M_L030	0.8	0 (off)
CH_M_R090	CH_M_R060, CH_M_R110	1.0	0 (off)
CH_M_R090	CH_M_R045, CH_M_R110	1.0	0 (off)
CH_M_R090	CH_M_R030, CH_M_R110	1.0	0 (off)
CH_M_R090	CH_M_R030	0.8	0 (off)
CH_M_L110	CH_M_L135	1.0	0 (off)
CH_M_L110	CH_M_L090	0.8	0 (off)
CH_M_L110	CH_M_L045	0.8	0 (off)
CH_M_L110	CH_M_L030	0.8	0 (off)
CH_M_R110	CH_M_R135	1.0	0 (off)
CH_M_R110	CH_M_R090	0.8	0 (off)
CH_M_R110	CH_M_R045	0.8	0 (off)
CH_M_R110	CH_M_R030	0.8	0 (off)
CH_M_L135	CH_M_L110	1.0	0 (off)

Source	Destination	Gain	EQ index
CH_M_L135	CH_M_L150	1.0	0 (off)
CH_M_L135	CH_M_L090	0.8	0 (off)
CH_M_L135	CH_M_L045	0.8	0 (off)
CH_M_L135	CH_M_L030	0.8	0 (off)
CH_M_R135	CH_M_R110	1.0	0 (off)
CH_M_R135	CH_M_R150	1.0	0 (off)
CH_M_R135	CH_M_R090	0.8	0 (off)
CH_M_R135	CH_M_R045	0.8	0 (off)
CH_M_R135	CH_M_R030	0.8	0 (off)
CH_M_L150	CH_M_L135	1.0	0 (off)
CH_M_L150	CH_M_L110	1.0	0 (off)
CH_M_L150	CH_M_L045	0.8	0 (off)
CH_M_L150	CH_M_L030	0.8	0 (off)
CH_M_R150	CH_M_R135	1.0	0 (off)
CH_M_R150	CH_M_R110	1.0	0 (off)
CH_M_R150	CH_M_R045	0.8	0 (off)
CH_M_R150	CH_M_R030	0.8	0 (off)
CH_M_180	CH_M_R150, CH_M_L150	1.0	0 (off)
CH_M_180	CH_M_R135, CH_M_L135	1.0	0 (off)
CH_M_180	CH_M_R110, CH_M_L110	1.0	0 (off)
CH_M_180	CH_M_R090, CH_M_L090	0.8	0 (off)
CH_M_180	CH_M_R045, CH_M_L045	0.6	0 (off)
CH_M_180	CH_M_R030, CH_M_L030	0.6	0 (off)
CH_U_000	CH_U_L030, CH_U_R030	1.0	0 (off)
CH_U_000	CH_M_L030, CH_M_R030	0.85	0 (off)
CH_U_L045	CH_U_L030	1.0	0 (off)
CH_U_L045	CH_M_L045	0.85	1
CH_U_L045	CH_M_L030	0.85	1
CH_U_R045	CH_U_R030	1.0	0 (off)
CH_U_R045	CH_M_R045	0.85	1
CH_U_R045	CH_M_R030	0.85	1
CH_U_L030	CH_U_L045	1.0	0 (off)
CH_U_L030	CH_M_L030	0.85	1
CH_U_R030	CH_U_R045	1.0	0 (off)
CH_U_R030	CH_M_R030	0.85	1
CH_U_L090	CH_U_L030, CH_U_L110	1.0	0 (off)
CH_U_L090	CH_U_L030, CH_U_L135	1.0	0 (off)
CH_U_L090	CH_U_L045	0.8	0 (off)
CH_U_L090	CH_U_L030	0.8	0 (off)
CH_U_L090	CH_M_L045, CH_M_L110	0.85	2
CH_U_L090	CH_M_L030, CH_M_L110	0.85	2
CH_U_L090	CH_M_L030	0.85	2
CH_U_R090	CH_U_R030, CH_U_R110	1.0	0 (off)
CH_U_R090	CH_U_R030, CH_U_R135	1.0	0 (off)
CH_U_R090	CH_U_R045	0.8	0 (off)
CH_U_R090	CH_U_R030	0.8	0 (off)
CH_U_R090	CH_M_R045, CH_M_R110	0.85	2
CH_U_R090	CH_M_R030, CH_M_R110	0.85	2
CH_U_R090	CH_M_R030	0.85	2
CH_U_L110	CH_U_L135	1.0	0 (off)
CH_U_L110	CH_U_L090	0.8	0 (off)
CH_U_L110	CH_U_L045	0.8	0 (off)
CH_U_L110	CH_U_L030	0.8	0 (off)

Source	Destination	Gain	EQ index
CH_U_L110	CH_M_L110	0.85	2
CH_U_L110	CH_M_L045	0.85	2
CH_U_L110	CH_M_L030	0.85	2
CH_U_R110	CH_U_R135	1.0	0 (off)
CH_U_R110	CH_U_R090	0.8	0 (off)
CH_U_R110	CH_U_R045	0.8	0 (off)
CH_U_R110	CH_U_R030	0.8	0 (off)
CH_U_R110	CH_M_R110	0.85	2
CH_U_R110	CH_M_R045	0.85	2
CH_U_R110	CH_M_R030	0.85	2
CH_U_L135	CH_U_L110	1.0	0 (off)
CH_U_L135	CH_U_L090	0.8	0 (off)
CH_U_L135	CH_U_L045	0.8	0 (off)
CH_U_L135	CH_U_L030	0.8	0 (off)
CH_U_L135	CH_M_L110	0.85	2
CH_U_L135	CH_M_L045	0.85	2
CH_U_L135	CH_M_L030	0.85	2
CH_U_R135	CH_U_R110	1.0	0 (off)
CH_U_R135	CH_U_R090	0.8	0 (off)
CH_U_R135	CH_U_R045	0.8	0 (off)
CH_U_R135	CH_U_R030	0.8	0 (off)
CH_U_R135	CH_M_R110	0.85	2
CH_U_R135	CH_M_R045	0.85	2
CH_U_R135	CH_M_R030	0.85	2
CH_U_180	CH_U_R135, CH_U_L135	1.0	0 (off)
CH_U_180	CH_U_R110, CH_U_L110	1.0	0 (off)
CH_U_180	CH_M_180	0.85	2
CH_U_180	CH_M_R110, CH_M_L110	0.85	2
CH_U_180	CH_U_R030, CH_U_L030	0.8	0 (off)
CH_U_180	CH_M_R030, CH_M_L030	0.85	2
CH_T_000	ALL_U	0.8	3
CH_T_000	ALL_M	0.8	4
CH_L_000	CH_M_000	1.0	0 (off)
CH_L_000	CH_M_L030, CH_M_R030	1.0	0 (off)
CH_L_L045	CH_M_L045	1.0	0 (off)
CH_L_L045	CH_M_L030	1.0	0 (off)
CH_L_R045	CH_M_R045	1.0	0 (off)
CH_L_R045	CH_M_R030	1.0	0 (off)
CH_LFE2	CH_LFE3	1.0	0 (off)
CH_LFE2	CH_M_L030, CH_M_R030	1.0	0 (off)
CH_LFE3	CH_LFE2	1.0	0 (off)
CH_LFE3	CH_M_L030, CH_M_R030	1.0	0 (off)

Table 281 — Normalized centre frequencies of the 58 processing bands

Normalized frequency [0, 1]
0.000000000000000
0.003891050583658
0.007782101167315
0.011673151750973
0.015564202334630
0.019455252918288

Normalized frequency [0, 1]
0.023346303501946
0.027237354085603
0.031128404669261
0.035019455252918
0.038910505836576
0.042801556420233
0.046692607003891
0.050583657587549
0.054474708171206
0.058365758754864
0.062256809338521
0.066147859922179
0.070038910505837
0.073929961089494
0.077821011673152
0.081712062256809
0.085603112840467
0.089494163424125
0.093385214007782
0.097276264591440
0.101167315175097
0.105058365758755
0.108949416342412
0.112840466926070
0.116731517509728
0.120622568093385
0.124513618677043
0.132295719844358
0.143968871595331
0.157587548638132
0.173151750972763
0.188715953307393
0.204280155642023
0.221789883268482
0.241245136186770
0.260700389105058
0.284046692607004
0.311284046692607
0.338521400778210
0.365758754863813
0.394941634241245
0.428015564202335
0.464980544747082
0.505836575875486
0.550583657587549
0.597276264591440
0.647859922178988
0.704280155642023
0.764591439688716
0.828793774319066
0.898832684824903
0.966926070038911

Table 282 — Processing band mapping table *pbm*(*k*)

k	*pbm(k)*	*k*	*pbm(k)*	*k*	*pbm(k)*	*k*	*pbm(k)*	*k*	*pbm(k)*	*k*	*pbm(k)*
0	0	50	37	100	46	150	51	200	54	250	57
1	1	51	38	101	46	151	51	201	54	251	57
2	2	52	38	102	46	152	51	202	54	252	57
3	3	53	38	103	46	153	51	203	54	253	57
4	4	54	38	104	46	154	51	204	54	254	57
5	5	55	39	105	46	155	51	205	55	255	57
6	6	56	39	106	47	156	51	206	55	256	57
7	7	57	39	107	47	157	51	207	55		
8	8	58	39	108	47	158	51	208	55		
9	9	59	39	109	47	159	51	209	55		
10	10	60	40	110	47	160	52	210	55		
11	11	61	40	111	47	161	52	211	55		
12	12	62	40	112	47	162	52	212	55		
13	13	63	40	113	47	163	52	213	55		
14	14	64	40	114	47	164	52	214	55		
15	15	65	41	115	48	165	52	215	55		
16	16	66	41	116	48	166	52	216	55		
17	17	67	41	117	48	167	52	217	55		
18	18	68	41	118	48	168	52	218	55		
19	19	69	41	119	48	169	52	219	55		
20	20	70	42	120	48	170	52	220	55		
21	21	71	42	121	48	171	52	221	55		
22	22	72	42	122	48	172	52	222	56		
23	23	73	42	123	48	173	52	223	56		
24	24	74	42	124	48	174	53	224	56		
25	25	75	42	125	49	175	53	225	56		
26	26	76	42	126	49	176	53	226	56		
27	27	77	43	127	49	177	53	227	56		
28	28	78	43	128	49	178	53	228	56		
29	29	79	43	129	49	179	53	229	56		
30	30	80	43	130	49	180	53	230	56		
31	31	81	43	131	49	181	53	231	56		
32	32	82	43	132	49	182	53	232	56		
33	33	83	43	133	49	183	53	233	56		
34	33	84	44	134	49	184	53	234	56		
35	33	85	44	135	49	185	53	235	56		
36	34	86	44	136	50	186	53	236	56		
37	34	87	44	137	50	187	53	237	56		
38	34	88	44	138	50	188	53	238	56		
39	35	89	44	139	50	189	54	239	56		
40	35	90	44	140	50	190	54	240	56		
41	35	91	45	141	50	191	54	241	57		
42	35	92	45	142	50	192	54	242	57		
43	36	93	45	143	50	193	54	243	57		
44	36	94	45	144	50	194	54	244	57		
45	36	95	45	145	50	195	54	245	57		
46	36	96	45	146	50	196	54	246	57		
47	37	97	45	147	50	197	54	247	57		
48	37	98	46	148	51	198	54	248	57		
49	37	99	46	149	51	199	54	249	57		

Table 283 — Equalizer parameters

Equalizer	P_f [Hz]	P_Q	P_g [dB]	g [dB]
$G_{EQ,1}$	12 000	0,3	-2	1,0
$G_{EQ,2}$	12 000	0,3	-3,5	1,0
$G_{EQ,3}$	200,1 300, 600	0,3, 0,5, 1,0	-6,5, 1,8, 2,0	0,7
$G_{EQ,4}$	5 000, 1 100	1,0, 0,8	4.5, 1,8	-3,1
$G_{EQ,5}$	35	0,25	-1,3	1,0

Table 284 — Vertically corresponding channels

CH_L_000	CH_M_000	CH_U_000
CH_L_L045	CH_M_L030	CH_U_L030
CH_L_L045	CH_M_L030	CH_U_L045
CH_L_L045	CH_M_L045	CH_U_L030
CH_L_L045	CH_M_L045	CH_U_L045
CH_L_L045	CH_M_L060	CH_U_L030
CH_L_L045	CH_M_L060	CH_U_L045
CH_L_R045	CH_M_R030	CH_U_R030
CH_L_R045	CH_M_R030	CH_U_R045
CH_L_R045	CH_M_R045	CH_U_R030
CH_L_R045	CH_M_R045	CH_U_R045
CH_L_R045	CH_M_R060	CH_U_R030
CH_L_R045	CH_M_R060	CH_U_R045
CH_M_180	CH_U_180	
CH_M_L090	CH_U_L090	
CH_M_L110	CH_U_L110	
CH_M_L135	CH_U_L135	
CH_M_L090	CH_U_L110	
CH_M_L090	CH_U_L135	
CH_M_L110	CH_U_L090	
CH_M_L110	CH_U_L135	
CH_M_L135	CH_U_L090	
CH_M_L135	CH_U_L135	
CH_M_R090	CH_U_R090	
CH_M_R110	CH_U_R110	
CH_M_R135	CH_U_R135	
CH_M_R090	CH_U_R110	
CH_M_R090	CH_U_R135	
CH_M_R110	CH_U_R090	
CH_M_R110	CH_U_R135	
CH_M_R135	CH_U_R090	
CH_M_R135	CH_U_R135	
NOTE Each row lists channels which are considered to be above/below each other.		

24.3.5 Audio signal processing

24.3.5.1 General

The audio processing block of the format converter obtains time domain audio samples for N_{in} channels from the core decoder and generates a downmixed time domain audio output signal consisting of N_{out} channels.

The processing takes as input

— the audio data decoded by the core decoder, and

— the static downmix matrix $\mathbf{M}_{\mathrm{DMX}}$ returned by the initialization of the format converter.

It returns an N_{out}-channel time domain output signal for the OutConf channel configuration signalled during the initialization of the format converter.

The format converter operates on contiguous, non-overlapping frames of length L = 256 time domain samples of the input audio signals and outputs one frame of L samples per processed input frame of length L. The algorithm performs a short time Fourier transform (STFT) of length N = 512 with 50 % overlap, i.e. the overlap length as well as the hop size of the STFT is 256 time domain samples. The STFT domain processing takes place in K=256 frequency bins, which are partitioned into PB=58 processing bands.

24.3.5.2 T/F-transform (STFT analysis)

As the first processing step the converter updates the STFT input buffer $\hat{y}_{ch,i}^{\nu}$ by one frame (L=256 samples) of the N_{in} channel time domain input signal $\left[\tilde{y}_{\mathrm{ch},1}^{\nu} \cdots \tilde{y}_{\mathrm{ch},N_{\mathrm{in}}}^{\nu}\right] = \tilde{\mathbf{y}}_{\mathrm{ch}}^{\nu}$:

$$\hat{y}_{ch,i}^{\nu} = \begin{cases} \tilde{y}_{ch,i}^{\nu,F-1} & \text{for} \quad 0 \le \nu < L, \quad 1 \le \mathrm{i} < N_{\mathrm{in}} \\ \tilde{y}_{ch,i}^{\nu-L,F} & \text{for} \quad L \le \nu < N, \quad 1 \le \mathrm{i} < N_{\mathrm{in}} \end{cases}$$

where F denotes the frame index and $\tilde{y}_{ch,i}^{\nu,F-1} = 0$ for the first processing frame. An analysis window is applied and a DFT of length N=512 is calculated for each of the N_{in} signals in the windowed STFT input buffer:

$$y_{ch,i}^{k} = \sum_{\nu=0}^{N-1} w[\nu] \cdot \hat{y}_{ch,i}^{\nu} \cdot e^{-2\pi jk\nu/N} \quad \text{for } 0 \le k < K,\ 1 \le i \le N_{in}$$

with $w[\nu] = \sin\left((\nu + 0.5)\pi / N\right)$.

24.3.5.3 Intermediate downmix signals

Intermediate downmix signals and corresponding energies are calculated according to:

$$\tilde{z}_{ch,o}^{k} = \sum_{i=1}^{N_{in}} y_{ch,i}^{k} M_{\mathrm{DMX},o,i}^{k} \quad \text{for} \quad 1 \le o \le N_{out},\ 0 \le k < K$$

$$\hat{Z}_{ch,o}^{pb} = \sum_{i=1}^{N_{in}} \sum_{\substack{k, \\ pbm(k) \\ = pb}} \left|y_{ch,i}^{k}\right|^2 \left(M_{\mathrm{DMX},o,i}^{k}\right)^2 \quad \text{for} \quad 1 \le o \le N_{out},\ 0 \le pb < PB$$

Similarly, the energies of the intermediate downmix signals are derived in the processing bands as:

$$\widehat{Z}_{ch,o}^{pb} = \sum_{\substack{k, \\ pbm(k) \\ =pb}} \left| \widehat{z}_{ch,o}^{k} \right|^2 \quad \text{for} \quad 1 \le o \le N_{out}, \; 0 \le pb < PB$$

24.3.5.4 Final frequency domain downmix signals

The final downmix signals are obtained in the STFT domain according to:

$$z_{ch,o}^{k} = \widehat{z}_{ch,o}^{k} EQ_{ch,o}^{k} \quad \text{for} \quad 1 \le o \le N_{out}, \; 0 \le k < K$$

with

$$EQ_{ch,o}^{k} = \begin{cases} 1 & \text{if} \quad \text{passiveDownmixFlag == 1} \\ \min\left(10^{0.4}, \max\left(10^{-0.5}, \widehat{EQ}_{ch,o}^{k}\right)\right) & \text{else} \end{cases}$$

and

$$\widehat{EQ}_{ch,o}^{k} = \sqrt{\frac{\overline{\widehat{Z}_{ch,o}^{pb}}}{\text{eps} + \overline{\widehat{Z}_{ch,o}^{pb}}}} \quad \text{where} \quad pb = pbm(k)$$

$$\overline{\widehat{Z}_{ch,o}^{pb,F}} = \alpha \widehat{Z}_{ch,o}^{pb,F} + (1-\alpha)\overline{\widehat{Z}_{ch,o}^{pb,F-1}} \quad \text{and} \quad \overline{\widehat{Z}_{ch,o}^{pb,F}} = \alpha \widehat{Z}_{ch,o}^{pb,F} + (1-\alpha)\overline{\widehat{Z}_{ch,o}^{pb,F-1}}$$

$\overline{\widehat{Z}_{ch,o}^{pb,F-1}}$ as well as $\overline{\widehat{Z}_{ch,o}^{pb,F-1}}$ shall be initialized to zero for the first processing frame.

24.3.5.5 F/T-transform (STFT synthesis)

As last processing step per processed frame of 256 samples, the downmix signal is transformed to the time domain by application of an inverse DFT, windowing and overlap-add update, yielding L time domain output samples $\tilde{z}_{ch,o}^{\nu}$ per output channel. For the current frame (frame index F) the operations read:

$$\hat{z}_{ch,o}^{\nu,F} = w[\nu] \frac{1}{N} \sum_{k=0}^{N-1} z_{ch,o}^{k,compl} \cdot e^{2\pi jk\nu/N} \quad \text{for } 0 \le \nu < N, \; 1 \le o \le N_{out}$$

with $z_{ch,o}^{k,compl} = \begin{cases} z_{ch,o}^{k} & 0 \le k < K \\ conj(z_{ch,o}^{N-k}) & K \le k < N \end{cases}$, where $conj(z)$ denotes the complex conjugate of z.

$$\tilde{z}_{ch,o}^{\nu,F} = \hat{z}_{ch,o}^{\nu,F} + \hat{z}_{ch,o}^{\nu+L,F-1} \quad \text{for } 0 \le \nu < L, \; 1 \le o \le N_{out},$$

where $\hat{z}_{ch,o}^{\nu,F-1}$ shall be initialized with zeros for the first processing frame.

25 Low complexity immersive loudspeaker rendering/format conversion

25.1 Description

For the 5.0 and 5.1 channel output layouts, immersive loudspeaker rendering shall be chosen to provide overhead sound images using surround channel loudspeakers depending on the immersiveDownmixFlag in downmixConfig(). The immersive loudspeaker renderer is a downmixer that converts multichannel signals from transmitted channel configurations with N_{in} channels to desired reproduction format of either 5.1 or 5.0 system. It has a switching scheme between 3D rendering and 2D rendering using different elevation rendering for height input channels depending on the transmitted bitstream **rendering3DType** to provide overhead sound image properly. It is thus also called 'immersive format converter'. The system consists of two major building blocks:

— an initialization algorithm that takes into account static parameters like the input and output format;

— a signal adaptive downmixing process that operates in a subband domain with a switching scheme according to the transmitted flag **rendering3DType**.

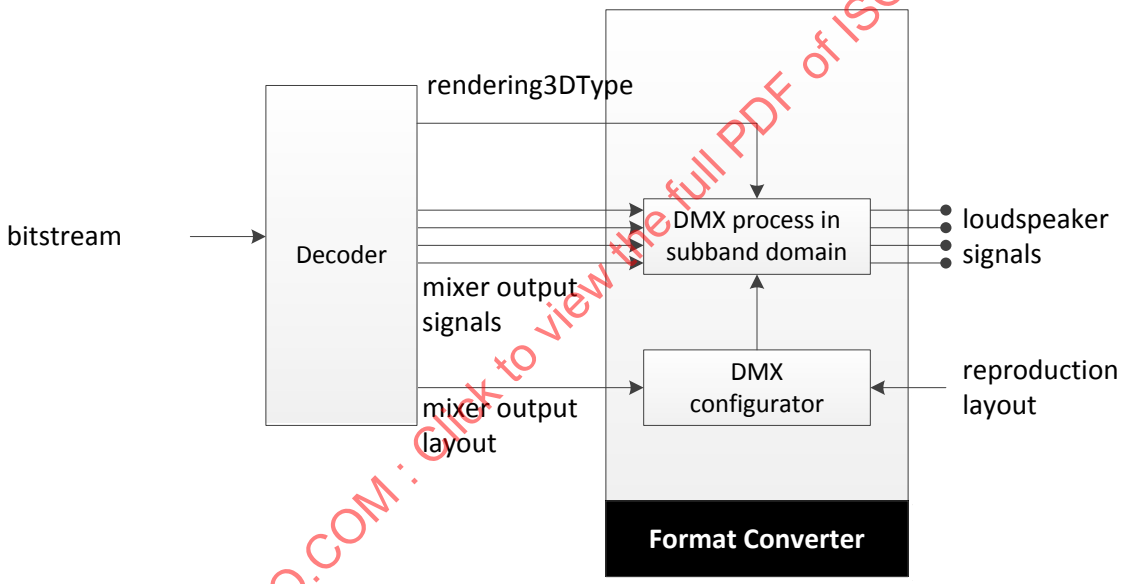


Figure 110 — Main building blocks of the immersive format converter

In the initialization phase the format converter automatically generates optimized downmixing parameters (like the downmixing matrix) for the given combination of input and output formats: It applies an algorithm that selects for each input loudspeaker the most appropriate mapping rule from a list of rules that has been designed to incorporate psychoacoustic considerations. Each rule describes the mapping from one input channel to one or several output loudspeaker channels.

Input channels are

— either mapped to a single output channel,

— or panned to two output channels,

— or (in case of the 'Voice of God' channel) distributed over a larger number of output channels,

— or (in case of the 'elevation rendering') panned to multiple output channels with different panning coefficients over frequency.

The optimal mapping for each input channel is selected depending on the list of output loudspeakers that are available in the desired output format. Each mapping defines downmix gains for the input channel under consideration as well as potentially also an equalizer that is applied to the input channel under consideration.

Output setups with non-standard loudspeaker positions can be signalled to the system by providing the azimuth and elevation deviations from a regular loudspeaker setup. Further, distance variations of the desired target loudspeaker positions are taken into account.

For each frame, a bit called rendering3DType is decoded by the decoder and passed to the immersive format converter. The rendering3DType indicates whether the sound scene is appropriate for the 3D rendering or 2D rendering over either 5.0 or 5.1 channel layout. For the height input channels, the immersive format converter uses the "spatial elevation rendering" for 3D rendering when the rendering3DType is TRUE and "timbral elevation rendering" for the 2D rendering when the rendering3DType is FALSE. For the non-height input channels, non-elevation rendering uses the same downmix coefficients regardless of the rendering3DType.

The actual downmixing of the audio signals is performed on a short time Fourier transform (STFT) representation of the signals. The energy-preserving algorithm avoids signal deteriorations like comb-filtering, coloration, or modulation artifacts.

25.2 Syntax

The FormatConverterFrame defines the proper rendering type for the immersive format converter.

Table 285 — Syntax of FormatConverterFrame()

Syntax	No. of bits	Mnemonic
FormatConverterFrame() { **rendering3DType;**	**1**	**uimsbf**
}		

The flag **rendering3DType** is created at the encoder based on the audio scene. When the audio scene is wideband and highly decorrelated at a frame, the flag **rendering3DType** becomes false and rendering is done by the secondary downmix matrix $\mathbf{M}_{DMX2}$. In all other cases, the flag becomes true and rendering is done by the primary downmix matrix $\mathbf{M}_{DMX}$, which provides elevated sound images.

25.3 Definitions

25.3.1 General remarks

Audio signals that are fed into the format converter are referred to as *input signals* in the following. Audio signals that are the result of the format conversion process are referred to as *output signals*. Note that the audio input signals of the format converter are audio output signals of the core decoder.

Vectors and matrices are denoted by bold-faced symbols. Vector elements or matrix elements are denoted as italic variables supplemented by indices indicating the row/column of the vector/matrix element in the vector/matrix, e.g. $\left[y_1 \cdots y_a \cdots y_N\right] = \mathbf{y}$ denotes a vector and its elements. Similarly, $M_{a,b}$ denotes the element in the ath row and bth column of a matrix $\mathbf{M}$.

25.3.2 Variable definitions

N_{in}	Number of channels in the input channel configuration.
N_{out}	Number of channels in the output channel configuration.
$\mathbf{M}_{DMX}$	Primary downmix matrix containing real-valued non-negative downmix coefficients (downmix gains) for the non-elevation rendering and spatial elevation rendering for 3D rendering over 2D layout. $\mathbf{M}_{DMX}$ is basically of dimension (N_{out} × N_{in}) but possibly increased to (N_{out} × (N_{in} + 5)) depending on the input and output layouts. See how the dimension is changed in subclause 25.4.2.3.
$\mathbf{M}_{DMX2}$	Secondary downmix matrix containing real-valued non-negative downmix coefficients (downmix gains) for the non-elevation rendering and timbral elevation rendering for 2D rendering over 2D layout. The downmix coefficients for the horizontal input channels are identical to those in $\mathbf{M}_{DMX}$. The $\mathbf{M}_{DMX}$ is dimension of (N_{out} × N_{in}) but possibly increased to (N_{out} × (N_{in} + 5)) depending on the input and output layouts. See how the dimension is changed in subclause 25.4.2.3.
$\mathbf{G}_{EQ}$	Matrix consisting of gain values per processing band determining frequency responses of equalizing filters for all rendering mapping. $\mathbf{G}_{EQ,1\sim5}$ are used for the non-elevation rendering and timbral elevation rendering, $\mathbf{G}_{EQ,7\sim14}$ are used for spatial elevation rendering, $\mathbf{G}_{EQ,15\sim20}$ are used for spatial coloration filter, and $\mathbf{G}_{EQ,21\sim}$ are used for modified EQ for randomized setup in subclause 25.4.1.6.5 and coloration filter in subclause 25.4.1.6.7.6.
$\mathbf{I}_{EQ}$	Vector signalling which equalizer filters to apply to the input channels (if any).
L	Frame length measured in time domain audio samples.
ν	Time domain sample index.
F	Frame index (frame number).
PB	Number of processing bands, PB=58.
pb	Processing band index ($0 \le pb < PB$).
N	DFT length.
K	Number of STFT frequency bins, K = 257.
k	STFT frequency bin index ($0 \le k < K$).
α	Filter parameter, α = 0,043 5.
A,B	Channel indices.
eps	Numerical constant, $\text{eps} = 10^{-35}$.

rendering3DType Flag from the bitstream identifying the rendering type for the elevation rendering. True for the general audio scene and false for the highly decorrelated wideband scene (e.g. applause). Accordingly, the primary downmix matrix $\mathbf{M}_{DMX}$ is chosen when the **rendering3DType** is TRUE and the secondary downmnix matrix $\mathbf{M}_{DMX2}$ is chosen when the **rendering3DType** is FALSE in the immersive format converter.

i_{in} Label of the input channel to be rendered by immersive format converter (e.g. CH_U_000)

$G_{vH0,1\sim6}(i_{in})$ Spatial elevation panning coefficients for the input channel (i_{in}) for the 2,8~10 kHz in order to provide overhead image. Note that the coefficient is always normalized to preserve the input power after mixing. The number index represents output channels as shown in Table 294.

$G_{vL0,1\sim6}(i_{in})$ Spatial elevation panning coefficients for the input channel (i_{in}) for below 2,8 kHz and above 10 kHz. Note that the coefficient is always normalized to reserve the input power after mixing. The number index represents output channels as shown in Table 294.

COLOR_A_B Tone coloration filter to the output at the azimuth of ±B degrees from the input at the azimuth of ±A degrees based on the ratio between HRTF at A and HRTF at B. It is determined by a frequency dependent dynamic cue which represents loudspeaker-to-listener orientation.

25.4 Processing

25.4.1 Initialization of the format converter

25.4.1.1 General description of the initialization

The initialization of the format converter is carried out before processing of the audio samples delivered by the core decoder takes place.

The initialization takes into account as input parameters:

— The sampling rate of the audio data to process;

— The channel configuration of the audio data to process with the format converter (number and geometric positions of input channels);

— The channel configuration of the desired output format (number and geometric positions of output channels);

— Optional: Parameters signalling the deviation of the output loudspeaker positions from a standard loudspeaker setup (random setup functionality).

It returns

— The primary frequency dependent downmix matrix $\mathbf{M}_{DMX}$ that is applied in the audio signal processing of the format converter when the rendering3DType is TRUE. Note that the $\mathbf{M}_{DMX}$ is independent variable in the Format Converter and shall not be taken into account in the core decoding process in subclause 5.5.4.1.2;

— The secondary frequency dependent downmix matrix $\mathbf{M}_{DMX2}$ that is applied in the audio signal processing of the format converter when the rendering3DType is FALSE. The $\mathbf{M}_{DMX2}$ is identical to $\mathbf{M}_{DMX}$ if there is no 'height' input channel or 'spatial elevation rendering' is not possible.

The input parameters to the initialization algorithm are listed in Table 286.

Table 286 — Format converter initialization input parameters

	Input format: number of channels and nominal channel setup geometry.
	Output format: number of channels and nominal channel setup geometry.
f_s	Sampling frequency in Hertz.
$r_{azi,A}$	For each output channel A, an azimuth angle is specified, determining the deviation from the standard format loudspeaker azimuth.
$r_{ele,A}$	For each output channel A, an elevation angle is specified, determining the deviation from the standard format loudspeaker elevation.

Table 287 lists the output parameters that are derived during the initialization of the format converter.

Table 287 — Format converter initialization output parameters

$\mathbf{M}_{DMX}$	Primary Downmix matrix [linear gains] for spatial elevation rendering (for rendering3DType == 1)
$\mathbf{M}_{DMX2}$	Secondary Downmix matrix [linear gains] for timbral elevation rendering (for rendering3DType == 0)

Note that the $\mathbf{M}_{DMX1}$ and $\mathbf{M}_{DMX2}$ include the same input-output downmix matrix for non-elevation rendering input channels.

25.4.1.2 Assignment of format converter channel labels to input/output format channels

The format converter initialization is based on a system of rules that are defined in terms of *format converter channel labels*, see Table 296. To allow the application of the initialization rules, the channel labels have to be assigned to the channels of the input and output formats. Each format converter channel label is associated with a segment of the surface of the unit sphere, as defined in Table 296. The segments are designed non-overlapping.

The assignment of channel labels to channels is done by geometrically matching the segments to the position data associated with the channels of the input and output formats. The azimuth and elevation angles in degrees of the position data associated with the channels shall be rounded towards the nearest integer number before performing the channel label assignment. Note that the *nominal* channel positions shall be applied in the following matching to channel label sectors, i.e. the azimuth and elevation angles *without* taking into account potential angle deviations signalled in $r_{azi,A}$ and/or $r_{ele,A}$.

For each channel that is not an LFE (low frequency enhancement) channel:

If the nominal position of the current channel, defined by its azimuth angle and elevation angle, is within or on the border of one of the segments defined in Table 296 then:

Assign the corresponding channel label (e.g. CH_M_L030) associated with the matching segment.

Add the angle differences between the nominal position of the current channel and the nominal position associated with the matching segment (i.e. the angles in the second and third column of Table 296) to the angle deviations stored in $r_{azi,A}$ and $r_{ele,A}$.

Else (i.e. no matching sector found), then:

Assign the CH_EMPTY label.

If an input or output format contains exactly one LFE channel, then the label CH_LFE1 shall be assigned to this channel.

If an input or output format contains exactly two LFE channels, then the labels CH_LFE1 and CH_LFE2 shall be assigned to the two LFE channels in the order that minimizes the maximum azimuth distance from the channels to the assigned CH_LFE1 and CH_LFE2 nominal azimuth positions.

If an input or output format contains more than 2 LFE channels, then those 2 LFE channels out of the considered setup shall be selected that minimize the maximum azimuth distance to the CH_LFE1 and CH_LFE2 nominal azimuth positions. The labels CH_LFE1 and CH_LFE2 shall be assigned as in the case of two LFE channels. The remaining LFE channels shall not be considered further in the calculation of downmix coefficients, i.e. the corresponding lines/columns of the downmix matrix shall remain filled with zeros.

25.4.1.3 Handling for unknown input channels

If the label CH_EMPTY is assigned to an input channel, this channel shall be considered unknown to the rules-based initialization and the downmix coefficients for mapping this input channel to the output channels shall be derived as specified in subclause 25.4.1.6.9.

25.4.1.4 Handling for unknown output formats

If the output format contains at least one channel with the label CH_EMPTY assigned to it, or if at least one channel label is assigned to more than one channel of the output format, the output format shall be considered unknown and the derivation of the downmixing coefficients shall be carried out as specified in subclause 25.4.1.6.9. The rules-based derivation of downmix coefficients shall not be applied for unknown output formats.

25.4.1.5 Handling of deviations from standard loudspeaker positions

If the below conditions are not met, the rules-based initialization is considered to have failed, the output format shall be considered to be unknown, and the downmixing gains shall be obtained as defined in subclause 25.4.1.6.9.

The absolute values of $r_{azi,A}$ and $r_{ele,A}$ shall not exceed 35 and 55 degrees, respectively. The minimum angle between any loudspeaker pair (without LFE channels) shall not be smaller than 15 degrees.

The values of $r_{azi,A}$ shall be such that the ordering by azimuth angles of the horizontal loudspeakers does not change. Likewise, the ordering of the height and low loudspeakers shall not change.

The values of $r_{ele,A}$ shall be such that the ordering by elevation angles of loudspeakers which are (approximately) above/below each other does not change. To verify this, the following procedure is applied:

For each row of Table 302 which contains two or three channels of the output format, do:

— Order the channels by elevation without randomization;

— Order the channels by elevation with considering randomization;

— If the two orderings differ, return an initialization error.

If the below conditions are not met, converter initialization is considered to have failed, and an error shall be returned.

25.4.1.6 Rules-based initialization algorithm

25.4.1.6.1 General

The rules-based initialization algorithm is defined in the following subclauses. The algorithm shall not be applied if the output format is considered unknown as defined in the previous subclause. For clarity the following description makes use of intermediate parameters listed in Table 288 but an implementation may omit the explicit use of these intermediate parameters.

Table 288 — Format converter initialization intermediate parameters

$\mathbf{S}$, $\mathbf{S}^{P}$, $\mathbf{S}^{S}$	Vector of converter source channels [input channel indices]
$\mathbf{D}$, $\mathbf{D}^{P}$, $\mathbf{D}^{S}$	Vector of converter destination channels [output channel indices]
$\mathbf{G}$, $\mathbf{G}^{P}$, $\mathbf{G}^{S}$	Vector of converter gains [linear]
$\mathbf{E}$, $\mathbf{E}^{P}$, $\mathbf{E}^{S}$	Vector of converter EQ indices
$\mathbf{G}_{EQ}$	Matrix containing equalizer gain values for all EQ indices and processing bands

The superscript S/P is the discriminator for the elevation rendering type. Those with designated superscript P are initialized to be used for the 'spatial elevation rendering' and used to create the primary downmix matrix $\mathbf{M}_{DMX}$, those with designated superscript S are for the 'timbral elevation rendering' and used to create the secondary downmix matrix, and those without superscript are for the non-elevation rendering and used to create both the primary and secondary downmix matrixes.

The intermediate parameters describe the dowmixing parameters according to the mapping, i.e. as sets of parameters S_i, D_i, G_i, E_i, per mapping i.

The format converter initialization output parameters are derived as described in the following Figure 111.

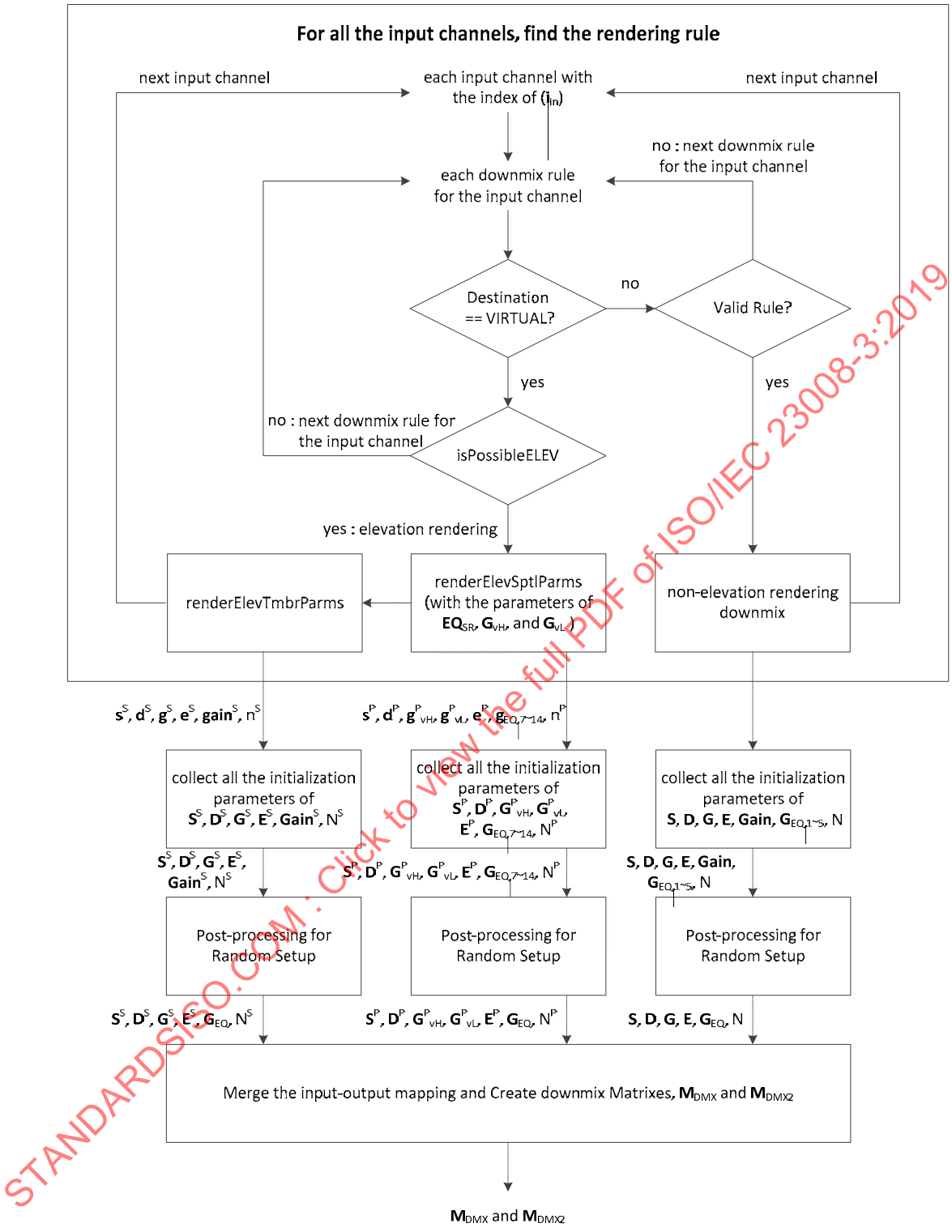


Figure 111 — Rule based downmix initialization flow

with following steps for each channel:

— If the destination of the downmix rule for the input channel (i_{in}) is VIRTUAL, isPossibleElev decides whether the input channel should be rendered by the elevation rendering defined in subclause 25.4.1.6.7.

- If isPossibleElev returns TRUE,
  - A set of parameters of $\mathbf{s}^P$, $\mathbf{d}^P$, $\mathbf{g}^P{}_H$, $\mathbf{g}^P{}_L$, $\mathbf{e}^P$, $\mathbf{G}_{EQ,eq(i_{in})}$, and n^P and another set of parameters of $\mathbf{s}^S$, $\mathbf{d}^S$, $\mathbf{g}^S$, $\mathbf{e}^S$, $\mathbf{gain}^S$, and n^S are initialized by the renderElevParms and renderTmbrParms, respectively. Tthe parameters of n^P and n^S indicate the number of output loudspeakers required for the input channel (i_{in}), $\mathbf{s}^P$, $\mathbf{d}^P$, $\mathbf{g}^P{}_H$, $\mathbf{g}^P{}_L$, and $\mathbf{e}^P$ are n^P column vectors, $\mathbf{s}^S$, $\mathbf{d}^S$, $\mathbf{g}^S$, $\mathbf{e}^S$, and $\mathbf{gain}^S$ are n^S column vectors, $\mathbf{G}_{EQ,eq(i_{in})}$ is a 58 row vector representing the EQ coefficients for the 58 processing bands of the input channel (i_{in}).
  - The initialization parameters are collected among the group of parameters, each primary or secondary, and the downmix rules for next input channel are investigated until all the input channel mapping is found.
- If isPossibleElev returns FALSE, the current downmix rule shall be ignored and the next downmix rule shall be investigated
- If the destination is not VIRTUAL, the downmix rule is investigated whether the rule is valid checking the output layout includes all the channels in the destination column.
  - If the downmix rule is valid,
    - A set of parameters shall be initialized by the non-elevation rendering initialization and added directly to the **S**, **D**, **G**, **E**, and **Gain** as specified in subclause 25.4.1.6.4 and to the $\mathbf{G}_{EQ}$ as specified in subclause 25.4.1.6.5.
    - The downmix rules for next input channel shall be investigated until all the input channel mappings are found.
  - If the downmix rule is invalid, the current downmix rule shall be ignored and the next downmix rule shall be investigated.
- After collecting all the initializations parameters for the 'spatial elevation rendering,' 'timbral elevation rendering,' and 'non-elevation rendering', post-processing for random setup shall be applied.
- Create the primary downmix matrix, combining the 'spatial elevation rendering' and 'non-elevation rendering' parameters, and the secondary downmix matrix, combining the 'timbral elevation rendering' and 'non-elevation rendering' parameters.

25.4.1.6.2 Random setups pre-processing

Random output loudspeaker setups, i.e. output setups that contain loudspeakers at positions deviating from the positions defined for the desired output format are signalled by specifying the loudspeaker position deviation angles as input parameters $r_{azi,A}$ and $r_{ele,A}$. The angle deviations are taken into account as a pre-processing step:

Modify in Table 296 the channels' azimuth and elevation angles by adding $r_{azi,A}$ and $r_{ele,A}$ to the corresponding channels' azimuth and elevation angles.

25.4.1.6.3 Derivation of input channel/output channel mapping parameters

The parameters vectors S, D, G, E define the mapping of input channels to output channels. For each mapping i from an input channel to an output channel with non-zero downmix gain they define the downmix gain as well as an equalizer index that indicates which equalizer curve has to be applied to the input channel under consideration in mapping i.

The elements of the parameter vectors S, D, G, E are derived by the following algorithm:

Initialize the mapping counter i: $i=1$, $i^P=1$, $i^S=1$;

Initialize the EQ counter e: e=21 (EQ slots for e from 1 to 5 are occupied by subclause 25.4.1.6.4, that for e from 7 to 14 are occupied by Table 293.

, and that for e from 15 to 20 are occupied by Table 290. The EQ counter e will be shared in subclauses 25.4.1.6.5 and 25.4.1.6.7.6 in an incremental way.)

For each input channel, ignoring channels with label CH_EMPTY assigned to them:

If the input channel also exists in the output format (e.g. input channel under consideration is CH_M_R030 and channel CH_M_R030 exists in the output format), then:

— S_i = index of source channel in input

EXAMPLE channel CH_M_R030 in ChannelConfiguration 6 is at second place according to Table 297, i.e. has index 2 in this format.

— D_i = index of same channel in output

— G_i = 1.0

— E_i = 0

— $i = i + 1$

Else (i.e. if the input channel does not exist in the output format)

— search the first entry of this channel in the **Source** column of Table 298, for which the channels in the corresponding row of the **Destination** column exist. The **VIRTUAL** destination shall be considered valid if the **isPossibleElev** returns **TRUE**, which indicates the output format contains required channels for the elevation rendering of the input channel. The **isPossibleElev**, **renderElevSptlParms,** and **renderElevTmbrParms** are defined in subclause 25.4.1.6.7. The ALL_U destination shall be considered valid (i.e. the relevant output channels exist) if the output format contains at least one “CH_U_” channel. The ALL_M destination shall be considered valid (i.e. the relevant output channels exist) if the output format contains at least one “CH_M_” channel. If for no entry in Table 298 corresponding to the input channel the channels in the **Destination** column exist, the rules-based initialization shall terminate and the downmix gains shall be derived according to subclause 25.4.1.6.9.

— If **Destination** column contains **VIRTUAL**, then:

  — $[\mathbf{s}^P, \mathbf{d}^P, \mathbf{g}^P{}_H, \mathbf{g}^P{}_L, \mathbf{e}^P, n^P]$= **renderElevSptlParms** (specified in subclause 25.4.1.6.7.4)

  — for n = 1 to n^P

    — $m = i^P$

    — $S^P{}_m = s^P{}_n(\mathrm{i_{in}})$

    — $D^P{}_m = d^P{}_n(\mathrm{i_{in}})$

    — $G^P{}_{mH} = g^P{}_{nH}(\mathrm{i_{in}})$

    — $G^P{}_{mL} = g^P{}_{nH}(\mathrm{i_{in}})$

    — $E^P{}_m = e^P{}_n(\mathrm{i_{in}})$

- $C^{P}_{mi} = (i_{in})$
- C^{P}_{mo} = label of the output channel $d^{P}_{n}(i_{in})$
- $i^{P} = i^{P} + 1$

— [$\mathbf{s}^{S}$, $\mathbf{d}^{S}$, $\mathbf{g}^{S}$, $\mathbf{e}^{S}$, **gain**, n^{S}]= **renderElevSptlParms** (described in subclause 25.4.1.6.7.5)

— for n = 1 to n^{S}

- $m = i^{S}$
- $S^{S}_{m} = s^{S}_{n}(i_{in})$
- $D^{S}_{m} = d^{S}_{n}(i_{in})$
- $G^{S}_{m}\ g^{S}_{n}(i_{in})$
- $E^{S}_{m} = e^{S}_{n}(i_{in})$
- $C^{S}_{mi} = (i_{in})$
- C^{S}_{mo} = label of the output channel $d^{P}_{n}(i_{in})$
- $Gain^{S}_{m} = gain_{n}$
- $i^{S} = i^{S} + 1$

— where i_{in} is the input channel label.

Else, if **Destination** column contains ALL_U, then:

— For each output channel x with "CH_U_" in its name, do:

- S_i = index of source channel in input
- D_i = index of channel x in output
- G_i = (value of Gain column) / sqrt(number of "CH_U_" output channels)
- E_i = value of EQ column
- $Gain_i$ = (value of Gain column)
- $i = i + 1$

Else if **Destination** column contains ALL_M, then:

— For each output channel x with "CH_M_" in its name, do:

- S_i = index of source channel in input
- D_i = index of channel x in output

- G_i = (value of Gain column) / sqrt(number of "CH_M_" output channels)
- E_i = value of EQ column
- $Gain_i$ = (value of Gain column)
- $i = i + 1$

Else If there is one channel in the **Destination** column, then:

- S_i = index of source channel in input
- D_i = index of destination channel in output
- G_i = value of Gain column
- E_i = value of EQ column
- $Gain_i$ = (value of Gain column)
- $i = i + 1$

Else (two channels in **Destination** column)

- S_i = index of source channel in input
- D_i = index of first destination channel in output
- G_i = (value of Gain column) · g_1
- E_i = value of EQ column
- $Gain_i$ = (value of Gain column)
- $i = i + 1$
- $S_i = S_{i-1}$
- D_i = index of second destination channel in output
- G_i = (value of Gain column) · g_2
- $E_i = E_{i-1}$
- $Gain_i$ = (value of Gain column)
- $i = i + 1$
- The gains g_1 and g_2 are computed by applying tangent law amplitude panning in the following way.
  - Unwrap source destination channel azimuth angles to be positive.
  - The azimuth angles of the destination channels are α_1 and α_2 (see Table 296).

— The azimuth angle of the source channel (= panning target) is α_{src}.

— $\alpha_0 = \frac{|\alpha_1 - \alpha_2|}{2}$

— $\alpha_{\text{center}} = \frac{\alpha_1 + \alpha_2}{2}$

— $\alpha = (\alpha_{\text{center}} - \alpha_{\text{src}}) \cdot \text{sgn}(\alpha_2 - \alpha_1)$

— $g_1 = \frac{g}{\sqrt{1+g^2}}, \quad g_2 = \frac{1}{\sqrt{1+g^2}} \quad \text{with} \quad g = \frac{\tan\alpha_0 - \tan\alpha + 10^{-10}}{\tan\alpha_0 + \tan\alpha + 10^{-10}}$

25.4.1.6.4 Derivation of equalizer gains G_{EQ}

G_{EQ} consists of gain values per processing band *pb* and equalizer index *e*. The 5 predefined equalizers are combinations of different peak filters for 1≤*e*≤5. Each equalizer is a serial cascade of one or more peak filters and a gain:

$$G_{EQ,e}^{pb} = 10^{g/20} \prod_{n=1}^{N} peak\left(band(pb)\frac{f_s}{2}, P_{f,n}, P_{Q,n}, P_{g,n}\right)$$

where *band*(*pb*) is the normalized centre frequency of processing band *pb*, specified in Table 299, f_s is the sampling frequency, and function *peak()* is for negative *G*

$$\text{peak}(b,f,Q,G) = \sqrt{\frac{b^4 + \left(\frac{1}{Q^2} - 2\right) f^2 b^2 + f^4}{b^4 + \left(\frac{10^{\frac{-G}{10}}}{Q^2} - 2\right) f^2 b^2 + f^4}}$$

and otherwise

$$\text{peak}(b,f,Q,G) = \sqrt{\frac{b^4 + \left(\frac{10^{\frac{G}{10}}}{Q^2} - 2\right) f^2 b^2 + f^4}{b^4 + \left(\frac{1}{Q^2} - 2\right) f^2 b^2 + f^4}}$$

The parameters for the equalizers are specified in Table 301.

25.4.1.6.5 Post-processing for random setups

Once the output parameters are computed, they are modified related to the specific random azimuth and elevations angles. This step has only to be carried out, if not all $r_{ele,A}$ are zero. Definition of the post-processing algorithm for non-elevation rendering.

For each element i in D_i, do:

if the output channel with index D_i is a horizontal channel by definition (i.e. output channel label contains the label '_M_'), **and**

- **if** this output channel is now a height channel (elevation in range 0..60 degrees), **and**
  - **if** input channel with index S_i is a height channel (i.e. label contains '_U_'), **then**
    - h = min(elevation of randomized output channel, 35) / 35
    - $G_{comp} = h \cdot \frac{1}{Gain_i} + (1 - h)$
    - Apply compensation gain to DMX gain: $G_i = G_i \cdot G_{comp}$
    - Define new equalizer $\mathbf{G}_{\mathrm{EQ},e}$ with the index e, where $G_{EQ,e}^{pb} = h + (1 - h) \cdot G_{EQ,E_i}^{pb}$
    - $E_i = e$
    - $e = e + 1$
  - **else if** input channel with index S_i is a horizontal channel (label contains '_M_')
    - h = min(elevation of randomized output channel, 35) / 35
    - Define new equalizer $\mathbf{G}_{\mathrm{EQ},e}$ with the index e,

      where $G_{EQ,e}^{pb} = \max\left(0.6310, \min\left(1.5849, \left(h \cdot \mathrm{IEQ}\left(D_i\right)^{pb} + \left(1 - h\right)\right) \cdot \mathrm{G}_{EQ,E_i}^{pb}\right)\right)$

      and $\mathrm{IEQ}\left(D_i\right)^{pb}$ is defined in Table 289
    - $E_i = e$
    - $e = e + 1$

Table 289 — Inverse spatial elevation filter ($EQ_{0,lin}$ is defined in Table 293)

Azimuth of D_i	Front centre (-15 15)	Front (-90 -15) or (15 90)	Side/rear [-180 -90] or [90 180]
$IEQ\left(D_i\right)^{pb}$	$\frac{1}{G_{EQ,9}^{pb}}$	$\frac{1}{G_{EQ,7}^{pb}}$	$\frac{1}{G_{EQ,8}^{pb}}$

Explanation of the post-processing steps defined above:

h is a normalized elevation parameter indicating the elevation of a nominally horizontal output channel ('_M_') due to a random setup elevation offset $r_{ele,A}$. For zero elevation offset h=0 follows and effectively no post-processing is applied.

The rule table (Table 298) in general applies a gain of the value in the gain column when mapping an upper input channel ('_U_' in channel label) to one or several horizontal output channels ('_M_' in channel label(s)). In case the output channel gets elevated due to a random setup elevation offset $r_{ele,A}$, the gain is partially ($0<h<1$) or fully (h=1) compensated for. Similarly the equalizer definitions fade towards a flat EQ-curve ($G_{EQ,e}^{pb} = const. = 1.0$) for h approaching h =1.

In case a horizontal input channel gets mapped to an output channel that gets elevated due to a random setup elevation offset $r_{ele,A}$, the equalizer $\mathrm{G}_{\mathrm{EQ,E_i}}^{\mathrm{k}}$ fully applied and $IEQ(D_i)^{pb}$, an inverse form of spatial

elevation filter defined in Table 289, is partially (0<h<1) or fully (h=1) applied. As the spatial elevation filter is defined to provide the tone color of overhead loudspeakers on horizontal loudspeakers, the inverse of the spatial elevation filter is used to provide the tone color of horizontal loudspeakers on overhead loudspeakers. Note that the modified EQ is thresholded within the level of 4 dB, [0,631 0, 1,584 9].

25.4.1.6.6 Spatial coloration filter for the horizontal input channels

When a horizontal input channel at side or rear is panned by two output loudspeakers, e.g. CH_M_L090 is panned by CH_M_L030 and CH_M_L110, the tone color changes. In order to avoid such a change in tone color, a set of horizontal coloration filters of $\mathbf{G}_{EQ,15\sim20}$ are defined as in Table 290. Here, the filter name COLOR_A_B means the coloration filter for the output at the azimuth of ±B from the input at the azimuth of ±A. The filtering algorithm is:

For each element i in **S** do:

A : the magnitude of the azimuth of the input channel with index S_i

B : the magnitude of the azimuth of the output channel with index D_i

If both S_i and D_i are horizontal channels and there exists a filter COLOR_A_B_ Table 290,

If the output channel has no deviation in azimuth and elevation, ($r_{azi,D_i} = 0$ and r_{ele,D_i}=0)

If A==60 && B == 30 E_i = 15 (COLOR_60_30)

Elseif A==90 && B == 30 E_i = 16 (COLOR_90_30)

Elseif A==60 && B == 110 E_i = 17 (COLOR_60_110)

Elseif A==90 && B == 110 E_i = 18 (COLOR_90_110)

Elseif A==135 && B == 110 E_i = 19 (COLOR_135_110)

Elseif A==180 && B == 110 E_i = 20 (COLOR_180_110)

Table 290 — Spatial coloration filters for horizontal channels

processing band	COLOR_180_110 $G_{EQ,20}$	COLOR_090_030 $G_{EQ,16}$	COLOR_060_110 $G_{EQ,17}$	COLOR_135_110 $G_{EQ,19}$	COLOR_090_110 $G_{EQ,18}$	COLOR_060_030 $G_{EQ,15}$
0	1.257512901	1.016393	0.975283	1.057872	0.967818	1.024233
1	1.158560317	0.940154	1.020111	1.025689	1.000571	0.958515
2	0.975947998	0.924468	1.052661	0.972375	1.03672	0.938683
3	0.812670539	1.019364	1.034273	0.940191	1.0457	1.008225
4	0.793650794	1.163062	1.007329	0.931031	1.043259	1.123006
5	0.793650794	1.255373	1.010088	0.925814	1.048451	1.209438
6	0.793651	1.221591	1.044259	0.911691	1.054614	1.209597
7	0.793651	1.164189	1.06016	0.895357	1.033851	1.193814

processing band	COLOR_180_110 $G_{EQ,20}$	COLOR_090_030 $G_{EQ,16}$	COLOR_060_110 $G_{EQ,17}$	COLOR_135_110 $G_{EQ,19}$	COLOR_090_110 $G_{EQ,18}$	COLOR_060_030 $G_{EQ,15}$
8	0.793651	1.151248	1.028108	0.894999	0.988836	1.196971
9	0.793651	1.174002	0.966481	0.915049	0.963343	1.177826
10	0.793651	1.235628	0.915263	0.954878	1.003832	1.126607
11	0.793651	1.26	0.914371	0.9984	1.075405	1.132111
12	0.793651	1.26	0.942626	1.002731	1.061876	1.26
13	0.793651	1.26	0.931554	0.959758	0.946306	1.26
14	0.793651	1.26	0.880058	0.919532	0.851115	1.26
15	0.793651	1.26	0.825	0.9062	0.840038	1.26
16	0.793651	1.22276	0.814011	0.917664	0.892063	1.115775
17	0.793651	1.160128	0.864691	0.949673	0.94146	1.065528
18	0.793651	1.089699	0.936597	0.993391	0.956656	1.066851
19	0.793651	1.037545	0.997666	1.029246	0.96065	1.077524
20	0.793651	0.986758	1.042468	1.031808	0.976646	1.053261
21	0.793651	0.929318	1.071918	1.001631	1.011122	0.985194
22	0.793651	0.883289	1.103262	0.96871	1.066606	0.913645
23	0.793651	0.858581	1.176045	0.959278	1.143043	0.88337
24	0.839215	0.849521	1.26	0.97236	1.234495	0.901514
25	0.895853	0.85036	1.26	0.976487	1.26	0.93963
26	0.921032	0.867297	1.26	0.950306	1.26	0.976639
27	0.915816	0.902004	1.26	0.916408	1.26	1.021141
28	0.907476	0.943387	1.26	0.910758	1.26	1.085708
29	0.902366	0.977346	1.26	0.928637	1.26	1.146713
30	0.900041	1.008207	1.26	0.934616	1.26	1.170319
31	0.903678	1.043796	1.26	0.90898	1.26	1.167112
32	0.895045	1.063373	1.26	0.866561	1.26	1.176141
33	0.827595	1.056064	1.26	0.794462	1.26	1.26
34	0.81639	1.075206	1.26	0.807225	1.26	1.229941
35	0.793651	1.002088	1.26	0.793651	1.26	1.094809
36	0.848212	0.893956	1.26	0.793651	1.26	0.991888
37	0.843676	0.944609	1.26	0.793651	1.26	1.054892
38	0.807275	1.002863	1.26	0.793651	1.26	1.176083
39	0.793651	1.199923	1.26	0.793651	1.26	1.26
40	0.793651	1.26	1.26	0.793651	1.26	1.26
41	0.793651	1.26	1.26	0.793651	1.26	1.26
42	0.793651	1.26	1.119248	0.793651	1.26	1.26
43	0.793651	1.26	0.912938	0.793651	1.186579	1.26
44	0.793651	1.26	0.793651	0.793651	1.047277	1.26

processing band	COLOR_180_110 $G_{EQ,20}$	COLOR_090_030 $G_{EQ,16}$	COLOR_060_110 $G_{EQ,17}$	COLOR_135_110 $G_{EQ,19}$	COLOR_090_110 $G_{EQ,18}$	COLOR_060_030 $G_{EQ,15}$
45	0.793651	1.26	0.793651	0.793651	0.958699	1.26
46	0.793651	1.26	0.793651	0.836434	0.937138	1.26
47	0.793651	1.26	1.102116	0.793651	1.092959	1.26
48	0.793651	1.26	1.092848	0.793651	1.26	1.26
49	0.793651	1.26	0.793651	0.815618	1.207313	1.26
50	0.793651	1.26	0.793651	0.818016	1.021066	0.793651
51	0.793651	0.922395	0.793651	0.899999	1.008149	0.793651
52	0.793651	0.793651	1.114079	0.856971	1.047784	0.793651
53	0.793651	0.793651	1.26	0.802702	1.26	0.793651
54	0.793651	0.976107	1.26	0.793651	1.26	0.992957
55	0.793651	1.153876	1.26	0.793651	1.26	1.034704
56	0.793651	1.26	1.231165	0.793651	1.26	1.26
57	0.793651	1.26	0.976335	0.793651	1.067507	1.26

25.4.1.6.7 Elevation rendering

25.4.1.6.7.1 General

Elevation rendering is intended to provide an overhead sound image when using a 5.1 channel layout. When the **Destination** column contains **VIRTUAL**, the **isPossibleElev** determines whether the input channel shall be rendered by the elevation rendering and is determined by comparing the output channel configuration and the required output channels for the spatial elevation rendering.

Table 291 — Detail information of the elevation rendering initialization parameters

$\mathbf{S}^P$	Vector of converter source channels [input channel indices]	Initialization parameters for spatial elevation rendering (renderElevSptlParms)
$\mathbf{D}^P$	Vector converter destination channels [output channel indices]	
$\mathbf{G}^P_H$	Vector of converter gains for 2.8~10kHz components	
$\mathbf{G}^P_L$	Vector of converter gains below 2.8kHz and above 10kHz components	
$\mathbf{E}^P$	Vector of converter EQ indices	
$\mathbf{S}^S$	Vector of converter source channels [input channel indices]	Initialization parameters for timbral elevation rendering (renderElevTmbrParms)
$\mathbf{D}^S$	Vector converter destination channels [output channel indices]	
$\mathbf{G}^S$	Vector of converter gains [linear]	
$\mathbf{E}^S$	Vector of converter EQ indices	
$\mathbf{G}_{EQ}$	Matrix containing equalizer gain values for all EQ indices and frequency bands	EQ matrix for all rendering types

When **isPossibleElev** is **TRUE**, two sets of parameters for the spatial elevation rendering and timbral elevation rendering are initialized using **renderElevSptlParms** and **renderElevTmbrParms**. The parameters of $\mathbf{S}^P$, $\mathbf{D}^P$, $\mathbf{G}^P_H$, $\mathbf{G}^P_L$, and $\mathbf{E}^P$, initialized by **renderElevSptlParms**, are for the spatial elevation

rendering, and $\mathbf{S}^S$, $\mathbf{D}^S$, $\mathbf{G}^S$, and $\mathbf{E}^S$, initialized by **renderElevTmbrParms** are for the timbral elevation rendering. The intermediate parameters describe the dowmixing parameters i per mapping index *i*, i.e. as sets of parameters S^P_i, D^P_i, G^P_{iH}, G^P_{iL}, and E^P_i.

25.4.1.6.7.2 isPossibleElev : Decision whether elevation rendering is valid for the output layout

The **isPossibleElev** boolean function returns **TRUE** when all of the required output channels exist. The required output channels are defined in Table 292, indicating 1 for the required channel and 0 for the unnecessary channel. For example, CH_M_L030, CH_M_L110, and CH_M_R110 are required in order to use elevation rendering for the input channel CH_U_L030. If any of the required output channels for the input channel is not in the output channel configuration, **isPossibleElev** shall return **FALSE** and elevation rendering shall not be applied for that input channel.

Table 292 — Required output channels for elevation rendering for isPossibleElev

Input Channel	Required output channels					
	CH_M_L030	CH_M_R030	CH_M_000	CH_LFE1	CH_M_L110	CH_M_R110
CH_U_000	1	1	1	0	1	1
CH_U_L045	1	1	0	0	1	1
CH_U_R045	1	1	0	0	1	1
CH_U_L030	1	1	0	0	1	1
CH_U_R030	1	1	0	0	1	1
CH_U_L090	1	0	0	0	1	1
CH_U_R090	0	1	0	0	1	1
CH_U_L110	1	0	0	0	1	1
CH_U_R110	0	1	0	0	1	1
CH_U_L135	1	0	0	0	1	1
CH_U_R135	0	1	0	0	1	1
CH_U_180	1	1	0	0	1	1
CH_T_000	1	1	1	0	1	1

25.4.1.6.7.3 initElevSptlParms: Initialization of elevation rendering parameters based on the input channel elevation

The initial values of the spatial elevation filters and the elevation panning coefficients are defined in Table 293, Table 294 and Table 295 for the 'height' input channels, except CH_T_000. When a 'height' input channel (except CH_T_000) has the elevation higher than 35 degrees, the spatial elevation filter coefficients and elevation panning coefficients shall be updated according to the degrees of the elevation. Note that the elevation sector is defined from +21 to +60 degrees for the height channels in Table 298.

For the elevation rendering of an input channel, an EQ is selected from the EQ column of the Table 298. For convenience, $eq(i_{in})$ is defined as the EQ column, e. g. if the i_{in} is CH_U_000, $eq(i_{in})$ is 9 (EQVFC).

Table 293 — Spatial elevation filter initial values (for the 35 degrees in elevation)

Processing bands (58 bands)	$EQ_{0,lin}$ (.)							
	7 (EQVF)	8 (EQVB)	9 (EQVFC)	10 (EQVBC)	11 (EQVOG)	12 (EQVS)	13 (EQBTM)	14 (EQVBA)
0	0.841395	0.819093	0.685085	0.770312	0.625280	0.877674	0.922571	0.877674
1	0.841395	0.819093	0.685085	0.770312	0.625280	0.877674	0.922571	0.877674
2	0.926119	0.922571	0.830000	0.940445	0.625280	0.901571	1.143756	0.988553
3	0.926119	0.980995	0.830000	1.015469	0.675162	1.000000	1.117721	1.051155
4	0.926119	0.980995	0.830000	1.015469	0.675162	1.000000	1.117721	1.051155
5	0.944061	0.887837	0.768679	1.000000	0.729024	0.905038	1.023293	0.951335
6	0.944061	0.887837	0.768679	1.000000	0.729024	0.905038	1.023293	0.951335
7	0.944061	0.887837	0.768679	1.000000	0.729024	0.905038	1.023293	0.951335
8	0.944061	0.887837	0.768679	1.000000	0.729024	0.905038	1.023293	0.951335
9	0.944061	0.887837	0.768679	1.000000	0.729024	0.905038	1.023293	0.951335
10	0.944061	0.887837	0.768679	1.000000	0.729024	0.905038	1.023293	0.951335
11	0.908518	0.958665	0.768679	0.887837	0.701576	1.039122	0.922571	1.027228
12	0.908518	0.958665	0.768679	0.887837	0.701576	1.039122	0.922571	1.027228
13	0.908518	0.958665	0.768679	0.887837	0.701576	1.039122	0.922571	1.027228
14	0.908518	0.958665	0.768679	0.887837	0.701576	1.039122	0.922571	1.027228
15	0.908518	0.958665	0.768679	0.887837	0.701576	1.039122	0.922571	1.027228
16	0.908518	0.958665	0.768679	0.887837	0.701576	1.039122	0.922571	1.027228
17	0.908518	0.958665	0.768679	0.887837	0.701576	1.039122	0.922571	1.027228
18	0.908518	0.958665	0.768679	0.887837	0.701576	1.039122	0.922571	1.027228
19	0.944061	0.936843	0.798751	0.838172	0.729024	1.023293	0.940445	0.860994
20	0.944061	0.936843	0.798751	0.838172	0.729024	1.023293	0.940445	0.860994
21	0.944061	0.936843	0.798751	0.838172	0.729024	1.023293	0.940445	0.860994
22	0.944061	0.936843	0.798751	0.838172	0.729024	1.023293	0.940445	0.860994
23	0.944061	0.936843	0.798751	0.838172	0.729024	1.023293	0.940445	0.860994
24	0.944061	0.936843	0.798751	0.838172	0.729024	1.023293	0.940445	0.860994
25	0.944061	0.936843	0.798751	0.838172	0.729024	1.023293	0.940445	0.860994
26	0.944061	0.936843	0.798751	0.838172	0.729024	1.023293	0.940445	0.860994
27	0.944061	0.936843	0.798751	0.838172	0.729024	1.023293	0.940445	0.860994
28	0.980995	0.962351	0.830000	0.922571	0.649743	0.940445	0.940445	0.992354
29	0.980995	0.962351	0.830000	0.922571	0.649743	0.940445	0.940445	0.992354
30	0.980995	0.962351	0.830000	0.922571	0.649743	0.940445	0.940445	0.992354
31	0.980995	0.962351	0.830000	0.922571	0.649743	0.940445	0.940445	0.992354
32	0.980995	0.962351	0.830000	0.922571	0.649743	0.940445	0.940445	0.992354
33	0.980995	0.962351	0.830000	0.922571	0.649743	0.940445	0.940445	0.992354
34	0.980995	0.962351	0.830000	0.922571	0.649743	0.940445	0.940445	0.992354

Processing bands (58 bands)	$EQ_{0,lin}$ (.)							
	7 (EQVF)	8 (EQVB)	9 (EQVFC)	10 (EQVBC)	11 (EQVOG)	12 (EQVS)	13 (EQBTM)	14 (EQVBA)
35	0.980995	0.962351	0.830000	0.922571	0.649743	0.940445	0.940445	0.992354
36	0.929311	0.892125	0.786271	0.855249	0.615512	0.871818	0.871818	0.919939
37	0.881628	0.908657	0.717845	0.908657	0.630513	0.944205	0.822369	0.901709
38	0.870100	0.892065	0.708457	0.892065	0.622267	0.926965	0.807353	0.885244
39	0.926766	0.855247	0.754596	0.823047	0.662793	0.909407	0.875168	0.848708
40	0.913913	0.838687	0.744132	0.807111	0.653601	0.891798	0.858223	0.832275
41	0.902215	0.826864	0.734606	0.823697	0.620943	0.875859	0.934905	0.886001
42	0.889454	0.810537	0.724216	0.807432	0.612160	0.858564	0.916444	0.868506
43	0.876028	0.793461	0.713285	0.790422	0.602920	0.840476	0.897137	0.850208
44	0.897706	0.793209	0.730935	0.775153	0.642011	0.856488	0.924814	0.907238
45	0.789863	0.779063	0.643126	0.809542	0.564885	0.809542	0.995954	0.911813
46	0.779856	0.733647	0.634979	0.706025	0.536730	0.765280	0.835899	0.786116
47	0.660004	0.546217	0.537392	0.484952	0.490480	0.567586	0.692945	0.585282
48	0.650985	0.535796	0.530049	0.475700	0.483778	0.556758	0.679725	0.574116
49	0.641943	0.525406	0.522686	0.466475	0.477058	0.545961	0.666543	0.562982
50	0.632968	0.515151	0.515379	0.457370	0.388260	0.535305	0.653533	0.551994
51	0.624467	0.505491	0.508457	0.448794	0.383046	0.525267	0.641279	0.541644
52	0.616092	0.496025	0.501638	0.440390	0.377909	0.515431	0.629270	0.531501
53	0.607606	0.486487	0.494729	0.431921	0.372704	0.505519	0.617170	0.521280
54	0.599370	0.477280	0.488023	0.423747	0.367652	0.495952	0.605490	0.511415
55	0.591397	0.468415	0.481530	0.415876	0.362761	0.486740	0.594243	0.501916
56	0.583481	0.459661	0.475085	0.408104	0.357906	0.477644	0.583138	0.492536
57	0.576447	0.451922	0.469358	0.401233	0.353590	0.469602	0.573319	0.484243

When the 'height' input channel i_{in} has higher elevation than 35 degrees, the spatial elevation filter is calculated from $EQ_{0,lin}^{pb}(\mathrm{eq}(i_{in}))$ in Table 293 by:

If the input channel, i_{in}, is frontal side : azimuth is in the range of (-90, 90)

$$EQ_{1,db}^{pb}\left(eq\left(i_{in}\right)\right)=\begin{cases}20\times\log_{10}\left(EQ_{0,lin}^{pb}\left(eq\left(i_{in}\right)\right)\right)+0.05\times\log_{2}\left(f_{pb}\times f_{s}/6000\right)+1 & \text{for } f_{pb}\times f_{s}>8000\\ 20\times\log_{10}\left(EQ_{0,lin}^{pb}\left(eq\left(i_{in}\right)\right)\right)+1 & \text{otherwise}\end{cases}$$

$$EQ_{2,db}^{pb}\left(eq\left(i_{in}\right)\right)=EQ_{1,lin}^{pb}\left(eq\left(i_{in}\right)\right)\times\left(1+\min(\max(\mathrm{elv}-35,0),25)\times 0.05\right)$$

$$G_{\mathrm{EQ},eq(i_{in})}^{pb}=\begin{cases}10^{\left(EQ_{2,db}^{pb}\left(eq(i_{in})\right)-1\right)/20-0.05\times\log_{2}\left(f_{pb}\times f_{s}/6000\right)} & \text{for } f_{pb}\times f_{s}>8000\\ 10^{\left(EQ_{2,db}^{pb}\left(eq(i_{in})\right)-1\right)/20} & \text{otherwise}\end{cases}$$

else (the input channel is rear side : azimuth is either in [-180, -90] or in [90, 180])

$$EQ_{1,db}^{pb}\left(eq\left(i_{in}\right)\right)=\begin{cases}20\times\log_{10}\left(EQ_{0,lin}^{pb}\left(eq\left(i_{in}\right)\right)\right)+0.07\times\log_{2}\left(f_{pb}\times f_{s}/6000\right)+1 & \text{for } f_{pb}\times f_{s}>8000\\ 20\times\log_{10}\left(EQ_{0,lin}^{pb}\left(eq\left(i_{in}\right)\right)\right)+1 & \text{otherwise}\end{cases}$$

$$EQ_{2,db}^{pb}\left(eq\left(i_{in}\right)\right)=EQ_{1,lin}^{pb}\left(eq\left(i_{in}\right)\right)\times\left(1+\min(\max(\text{elv}-35,0),25)\times 0.05\right)$$

$$G_{\text{EQ},eq\left(i_{in}\right)}^{pb}=\begin{cases}10^{\left(EQ_{2,db}^{pb}\left(eq\left(i_{in}\right)\right)-1\right)/20-0.07\times\log_{2}\left(f_{pb}\times f_{s}/6000\right)} & \text{for } f_{pb}\times f_{s}>8000\\ 10^{\left(EQ_{2,db}^{pb}\left(eq\left(i_{in}\right)\right)-1\right)/20} & \text{otherwise}\end{cases}$$

where

f_{pb} is the normalized centre frequency of processing band *pb*, specified in Table 299;

f_s is the sampling frequency;

elv is the elevation of the input channel.

When the 'height' input channel i_{in} does not have higher elevation than 35 degrees, use the initial filter coefficients according to:

$$G_{\text{EQ},\text{eq}(i_{in})}^{pb}=EQ_{0,lin}^{pb}(\text{eq}(i_{in}))$$

The spatial elevation panning coefficients shall also be updated for the 'height' input channels, except CH_T_000 and CH_U_180, for the different elevation degrees. Table 294 and Table 295 show the initial panning coefficients for the input channels with the elevation of 35 degrees. When the elevation of the input channel is higher than 35 degrees, the ipsilateral gain applied to the input channel shall be reduced and the contralateral channel shall be boosted with the gain difference $g_I(\text{elv})$ and $g_C(\text{elv})$.

Table 294 — Initial spatial localization panning coefficients for 2,8 kHz ~ 10 kHz (35 degrees in elevation)

Channel label, i_{in}	$G_{vH0,1\sim6}(i_{in})$					
	CH_M_L030 ($G_{vH0,1}(i_{in})$)	CH_M_R030 ($G_{vH0,2}(i_{in})$)	CH_M_000 ($G_{vH0,3}(i_{in})$)	CH_LFE1 ($G_{vH0,4}(i_{in})$)	CH_M_L110 ($G_{vH0,5}(i_{in})$)	CH_M_R110 ($G_{vH0,6}(i_{in})$)
CH_U_000	0.49146774	0.49146774	0.34746769	0	0.44507593	0.44507593
CH_U_L045	0.70918131	0.27444959	0	0	0.56982642	0.31150770
CH_U_R045	0.27444959	0.70918131	0	0	0.31150770	0.56982642
CH_U_L030	0.70918131	0.27444959	0	0	0.56982642	0.31150770
CH_U_R030	0.27444959	0.70918131	0	0	0.31150770	0.56982642
CH_U_L090	0.56040317	0	0	0	0.81550622	0.14456093
CH_U_R090	0	0.56040317	0	0	0.14456093	0.81550622
CH_U_L110	0.34278116	0	0	0	0.91200900	0.22525696
CH_U_R110	0	0.34278116	0	0	0.22525696	0.91200900
CH_U_L135	0.34278116	0	0	0	0.91200900	0.22525696
CH_U_R135	0	0.34278116	0	0	0.22525696	0.91200900
CH_U_180	0.22851810	0.22851810	0	0	0.66916323	0.66916323
CH_T_000	0.45328009	0.45328009	0.33519593	0	0.48822021	0.48822021

Table 295 — Initial spatial localization panning coefficients below 2,8 kHz and above 10 kHz (35 degrees in elevation)

Channel label, i_{in}	$G_{vL0,1\sim6}(i_{in})$					
	CH_M_L030 ($G_{vL0,1}(i_{in})$)	CH_M_R030 ($G_{vL0,2}(i_{in})$)	CH_M_000 ($G_{vL0,3}(i_{in})$)	CH_LFE1 ($G_{vL0,4}(i_{in})$)	CH_M_L110 ($G_{vL0,5}(i_{in})$)	CH_M_R110 ($G_{vL0,6}(i_{in})$)
CH_U_000	0.61940062	0.61940062	0.43791625	0	0	0
CH_U_L045	1	0	0	0	0	0
CH_U_R045	0	1	0	0	0	0
CH_U_L030	1	0	0	0	0	0
CH_U_R030	0	1	0	0	0	0
CH_U_L090	0.36730000	0	0	0	0.93010002	0
CH_U_R090	0	0.36730000	0	0	0	0.93010002
CH_U_L110	0	0	0	0	1	0
CH_U_R110	0	0	0	0	0	1
CH_U_L135	0.34278116	0	0	0	0.91200900	0.22525696
CH_U_R135	0	0.34278116	0	0	0.22525696	0.91200900
CH_U_180	0.22851810	0.22851810	0	0	0.66916323	0.66916323
CH_T_000	0.45328009	0.45328009	0.33519593	0	0.48822021	0.48822021

For all height input channel i_{in}, the $G_{vH,1\sim6}$ and $G_{vL,1\sim6}$ shall be initialized with $G_{vH0,1\sim6}$:

$$G_{vH,1\sim6}(i_{in}) = G_{vH0,1\sim6}(i_{in})$$

$$G_{vL,1\sim6}(i_{in}) = G_{vL0,1\sim6}(i_{in})$$

For each height input channel i_{in}, the $G_{vH,1\sim6}$ shall be calculated from $G_{vH0,1\sim6}$:

If the input channel is CH_U_000,

$$G_{vH,5}(i_{in}) = 10^{(0.25\cdot\min(\max(elv-35,0),25))/20}\cdot G_{vH0,5}(i_{in})$$

$$G_{vH,6}(i_{in}) = 10^{(0.25\cdot\min(\max(elv-35,0),25))/20}\cdot G_{vH0,6}(i_{in})$$

Elseif the input channel is not CH_U_180

if the input channel is side : azimuth is either in (-110, -70) or in (70, 110),

$$g_I(elv) = 10^{(-0.05522\cdot\min(\max(elv-35,0),25))/20}$$

$$g_C(elv) = 10^{(0.41879\cdot\min(\max(elv-35,0),25))/20}$$

else (the input channel is frontal or rear in [-70, 70], [-180, -110], or [110 180])

$$g_I(elv) = 10^{(-0.047401\cdot\min(\max(elv-35,0),25))/20}$$

$$g_C(elv) = 10^{(0.14985\cdot\min(\max(elv-35,0),25))/20}$$

For each output channels ipsilateral to the input channel,
(e.g. CH_M_L030 and CH_M_L110 for CH_U_L045)

$G_{vH,I}(i_{in}) = g_I(elv) \cdot G_{vH0,I}(i_{in})$

where the I of $G_{vH,I}(i_{in})$ represents the indices ipsilateral to the i_{in}

For each output channels contralateral to the input channel,
(e.g. CH_M_R030 and CH_M_R110 for CH_U_L045)

$G_{vH,C}(i_{in}) = g_C(elv) \cdot G_{vH0,C}(i_{in})$

where the C of $G_{vH,C}(i_{in})$ represents the indices contralateral to the i_{in}

A set of spatial localization panning coefficients for the input channel is normalized to make the sum of powers be 1.

$$P_{G_{vH}}(i_{in}) = \sqrt{\sum_{o=1}^{6} G_{vH,o}^2\,(i_{in})}$$

$$G_{vH,1\sim6}(i_{in}) = \frac{1}{P_{G_{vH}}} G_{vH,1\sim6}(i_{in})$$

If the input channel is in [-160, -110) or (110 160], $G_{vL,C}(i_{in})$ shall be updated:

$$G_{vL,I}(i_{in}) = g_I(elv) \cdot G_{vL0,I}(i_{in})$$

$$G_{vL,C}(i_{in}) = g_C(elv) \cdot G_{vL0,C}(i_{in})$$

$$P_{G_{vL}}(i_{in}) = \sqrt{\sum_{o=1}^{6} G_{vL,o}^2(i_{in})}$$

$$G_{vL,1\sim6}(i_{in}) = \frac{1}{P_{G_{vL}}} G_{vL,1\sim6}(i_{in})$$

Note that only the panning coefficients of the $G_{vL,1\sim6}$ for the rear input channels change and those for the frontal/side input channel do not.

As a result of **initElevSptlParms** following parameters are derived.

$G^{pb}_{EQ,eq(i_{in})}$: Updated spatial elevation filter coefficient vector (58 bands) for the input channel i_{in}.

$G_{vH,1\sim6}(i_{in})$: Updated spatial elevation panning coefficients for the input signal in the range of 2,8~10 kHz

$G_{vL,1\sim6}(i_{in})$: Updated spatial elevation panning coefficients for the input signal below 2,8 kHz and above 10 kHz

25.4.1.6.7.4 renderElevSptlParms : Derivation of input-output channel mapping and equalizer for spatial elevation rendering

renderElevSptlParms initializes the input-output channel mapping for spatial elevation rendering for an input channel (i_{in}).

Initialize the mapping counter i=1;

For m=1 to 6

If $G_{vH,m}(i_{in}) > 1$

$s^P{}_i$ = index of source channel i_{in}

$d^P{}_i$ = index of channel m in output

$g^P{}_{iH}$ = (value of Gain column) * $G_{vH,m}(i_{in})$

$g^P{}_{iL}$ = (value of Gain column) * $G_{vL,m}(i_{in})$

$e^P{}_i$ = eq(i_{in})

$i = i + 1$

$n^P = i - 1$;

return { $\mathbf{s}^P$, $\mathbf{d}^P$, $\mathbf{g}^P{}_H$, $\mathbf{g}^P{}_L$, $\mathbf{e}^P$, and n^P }

Note that the initialization does not add the input-output channel mapping that the gain $g^P{}_{iH}$ is zero and eq(i_{in}) is from 7 to 15.

By applying the downmix rules defined in Table 298 with the spatial elevation filter and spatial localization panning coefficients, the spatial elevation rendering parameters for each input channel is summarized as:

- CH_U_L135, CH_U_R135, CH_U_180, or CH_T_000
  - Spatial Elevation Filter : the HRTF-based EQs (EQVB , EQVBC, or EQVOG)
  - Spatial Elevation Panning : Filtered signal is multiplied by the same panning coefficients over all frequency (the $\mathbf{g}^P{}_H$ is identical to the $\mathbf{g}^P{}_L$)
- CH_U_L110, and CH_U_R110
  - Spatial Elevation Filter : the HRTF-based EQ (EQVBA)
  - Spatial Elevation Panning : Filtered EQ signal is multiplied by the different panning coefficients over frequency range
    - $\mathbf{g}^P{}_H$ for the elevation-effective range (2,8 k ~ 10 kHz)
    - $\mathbf{g}^P{}_L$ for the rest frequency range : Use the “add-to-the-closest channel” method in order to provide enough envelopment and keep the audio channel wide enough.
- CH_U_L090 and CH_U_R090
  - Spatial Elevation Filter : the HRTF-based EQs (EQVS)
  - Spatial Elevation Panning : Filtered EQ signal is multiplied by the different panning coefficients over frequency range
    - $\mathbf{g}^P{}_H$ for the elevation-effective range (2,8 k ~ 10 kHz)
    - $\mathbf{g}^P{}_L$ for the rest frequency range : Panned at 90 degrees using front and surround loudspeakers in order to provide enough envelopment and keep the audio channel wide enough.

— CH_U_L030, CH_U_R030, CH_U_L045, CH_U_R045
  — Spatial Elevation Filter : the HRTF-based EQs (EQVF)
  — Spatial Elevation Panning : Filtered EQ signal is multiplied by the different panning coefficients over frequency range
    — $\mathbf{g}^{P}_{H}$ for the elevation-effective range (2,8 k ~ 10 kHz)
    — $\mathbf{g}^{P}_{L}$ for the rest frequency range : Use the “add-to-the-closest channel” method in order to provide enough envelopment and keep the audio channel wide enough.
  — Signals for the surround loudspeakers, CH_M_L110 and CH_M_R110, are delayed by one STFT hop size in order to avoid front-back confusion using the precedence effect. For details see subclause 25.4.2.3
— CH_U_000
  — Spatial Elevation Filter: the HRTF-based EQ (EQVFC)
  — Spatial Elevation Panning : Filtered EQ signal is multiplied by the different panning coefficients over frequency range
    — $\mathbf{g}^{P}_{H}$ for the elevation-effective range (2,8 k ~ 10 kHz)
    — $\mathbf{g}^{P}_{L}$ for the rest frequency range : panned among three frontal output channels, CH_M_L030, CH_M_000, and CH_M_R030.
  — Signals for the surround loudspeakers, CH_M_L110 and CH_M_R110, are delayed by one STFT hop size in order to avoid front-back confusion using the precedence effect. For details see subclause 25.4.2.3

25.4.1.6.7.5 renderElevTmbrParms : Derivation of input-output channel mapping and equalizer for timbral elevation rendering

In the same manner as **renderElevSptlParms**, **renderElevTmbrParms** initializes another input-output channel mapping for timbral elevation rendering for the same input channel i_{in}. The parameters are initialized following subclause 25.4.1.6.3 but ignoring the downmix rules with a destination field of **VIRTUAL**. As a result, a set of $\mathbf{s}^{S}$, $\mathbf{d}^{S}$, $\mathbf{g}^{S}$, $\mathbf{e}^{S}$, and $\mathbf{gain}^{S}$ are defined for the input channel i_{in} by the process:

Initialize the mapping counter i=1;

Search the first entry of the input channel in the Source column of the Table 298 and the channels in the Destination column exist below the entry with VIRTUAL destination.

If **Destination** column contains ALL_U, then:

For each output channel x with “CH_U_” in its name, do:
- s^{S}_{i} = index of source channel in input
- d^{S}_{i} = index of channel x in output
- g^{S}_{i} = (value of Gain column) / sqrt(number of “CH_U_” output channels)
- e^{S}_{i} = value of EQ column
- $gain_{i}$ = (value of Gain column)
- $i = i + 1$

Else if **Destination** column contains ALL_M, then:
For each output channel x with “CH_M_” in its name, do:

- s^S_i = index of source channel in input
- d^S_i = index of channel x in output
- g^S_i = (value of Gain column) / sqrt(number of "CH_M_" output channels)
- e^S_i = value of EQ column
- $gain_i$ = (value of Gain column)
- $i = i + 1$

Else If there is one channel in the **Destination** column, then:

- s^S_i = index of source channel in input
- d^S_i = index of destination channel in output
- g^S_i = value of Gain column
- e^S_i = value of EQ column
- if e^S_i == 13 (EQBTM)
- e^S_i = 0 (No Process)
- $gain_i$ = (value of Gain column)
- $i = i + 1$

Else (two channels in **Destination** column)

- s^S_i = index of source channel in input
- d^S_i = index of first destination channel in output
- g^S_i = (value of Gain column) * g_1
- e^S_i = value of EQ column
- if e^S_i == 13 (EQBTM)
- e^S_i = 0 (No Process)
- $gain_i$ = (value of Gain column)
- $i = i + 1$

- $s^S_i = s^S_{i-1}$
- d^S_i = index of second destination channel in output
- g^S_i = (value of Gain column) * g_2
- $e^S_i = e^S_{i-1}$
- $gain_i$ = (value of Gain column)
- $i = i + 1$

$n^S = i - 1$;
return { $\mathbf{s}^S$, $\mathbf{d}^S$, $\mathbf{g}^S$, $\mathbf{e}^S$, **gain** and n^S }

The gains g_1 and g_2 are computed by applying tangent law amplitude panning in the following way.

— Unwrap source destination channel azimuth angles to be positive.

— The azimuth angles of the destination channels are α_1 and α_2 (see Table 296).

— The azimuth angle of the source channel (= panning target) is α_{src}.

— $\alpha_0 = \frac{|\alpha_1 - \alpha_2|}{2}$

— $\alpha_{\text{center}} = \frac{\alpha_1 + \alpha_2}{2}$

— $\alpha = (\alpha_{\text{center}} - \alpha_{\text{src}}) \cdot \text{sgn}(\alpha_2 - \alpha_1)$

— $g_1 = \frac{g}{\sqrt{1+g^2}}, \quad g_2 = \frac{1}{\sqrt{1+g^2}} \quad \text{with} \quad g = \frac{\tan\alpha_0 - \tan\alpha + 10^{-10}}{\tan\alpha_0 + \tan\alpha + 10^{-10}}$

25.4.1.6.7.6 Post-processing for random setups with elevation rendering

After the parameters of $\mathbf{S}^P$, $\mathbf{D}^P$, $\mathbf{G}^P_H$, $\mathbf{G}^P_L$, and $\mathbf{E}^P$ are initialized based on channel information, they are modified according to the azimuth and elevation deviations. For the convenience, C^P_{ii} refers the label of S^P_i, C^P_{io} refers the label of D^P_i, r_{ele,D_i}refers the elevation deviation of the C^P_{io}, and r_{azi,D_i}refers the azimuth deviation of the C^P_{io},

1) Elevation post-processing 1 : Find the “practically identical” channel

For each element *i* in $\mathbf{S}^P$, do

$flag(i)$=0

For each element *i* in $\mathbf{S}^P$, do

If $r_{ele,D_i} > 20$ and $r_{azi,D_i} \leq 15$
if (C^P_{io} == CH_M_L030 and (C^P_{ii} == CH_U_L030 || C^P_{ii} == CH_M_L045)) ||
(C^P_{io} == CH_M_R030 and (C^P_{ii} == CH_U_R030 || C^P_{ii} == CH_M_R045)) ||
(C^P_{io} == CH_M_000 and C^P_{ii} == CH_U_000) ||
{
G^P_{iH} = 1
G^P_{iL} = 1
$flag(i)$= 1
For each element *j* in $\mathbf{S}^P$, do
If $C^P_{ii} == C^P_{ji}$ && $i \neq j$
G^P_{jH} = 0
G^P_{jL} = 0
$flag(j)$= 1
}
If $r_{ele,D_i} > 20$ and $r_{azi,D_i} \leq 25$
if (C^P_{io} == CH_M_L110 and (C^P_{ii} == CH_U_L110 || C^P_{ii} == CH_M_L135)) ||
(C^P_{io} == CH_M_R110 and (C^P_{ii} == CH_U_R110 || C^P_{ii} == CH_M_R135))
{
G^P_{iH} = 1
G^P_{iL} = 1
$flag(i)$= 1

For each element j in $\mathbf{S}^{P}$, do
If $C^{P}_{ii} == C^{P}_{ji}$ && $i \neq j$
$G^{P}_{jH} = 0$
$G^{P}_{jL} = 0$
$flag(j) = 1$
}
For each element i in $\mathbf{S}^{P}$, do
If $r_{ele,D_i} > 20$ and $C^{P}_{io} ==$ CH_M_L110 and $C^{P}_{ii} ==$ CH_U_L090
For each element j in $\mathbf{S}^{P}$, do
If $r_{ele,D_j} > 20$ and $C^{P}_{jo} ==$ CH_M_L030 and $C^{P}_{ii} ==$ CH_U_L090
For each element k in $\mathbf{S}^{P}$, do
If $C^{P}_{ki} ==$ CH_U_L090
$G^{P}_{kH} = 0$
$G^{P}_{kL} = 0$
$flag(k) = 1$
$G^{P}_{iH} = g_1$
$G^{P}_{iL} = g_1$
$G^{P}_{jH} = g_2$
$G^{P}_{jL} = g_2$
$flag(i) = 1$
$flag(j) = 1$
If $r_{ele,D_i} > 20$ and $C^{P}_{io} ==$ CH_M_R110 and $C^{P}_{ii} ==$ CH_U_R090
For each element j in $\mathbf{S}^{P}$, do
If $r_{ele,D_j} > 20$ and $C^{P}_{jo} ==$ CH_M_R030 and $C^{P}_{ii} ==$ CH_U_R090
For each element k in $\mathbf{S}^{P}$, do
If $C^{P}_{ki} ==$ CH_U_R090
$G^{P}_{kH} = 0$
$G^{P}_{kL} = 0$
$flag(k) = 1$
$G^{P}_{iH} = g_1$
$G^{P}_{iL} = g_1$
$G^{P}_{jH} = g_2$
$G^{P}_{jL} = g_2$
$flag(i) = 1$
$flag(j) = 1$

The gains g_1 and g_2 are computed by applying tangent law amplitude panning in the following way.

— Unwrap source destination channel azimuth angles to be positive.

— The azimuth with the deviation for C^{P}_{io} and C^{P}_{jo} are α_1 and α_2.

— The azimuth angle of the source channel (= panning target) is α_{src}.