INTERNATIONAL STANDARD

ISO/IEC 23008-3

First edition
2015-10-15

Corrected version
2016-03-01

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 23008-3:2015(E)

Documents
Document Sharing Hub
© ISO/IEC 2015

Full document details: ISO_IEC 23008-3_2015



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2015, Published in Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

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

Documents
Document Sharing Hub

© ISO/IEC 2015 – All rights reserved

Full document details: ISO_IEC 23008-3_2015

Contents

Page

Foreword viii

Introduction x

1 Scope 1

2 Normative references 1

3 Terms, definitions and mnemonics 1
3.1 Terms and Definitions 1
3.2 Mnemonics 1

4 Technical Overview 2
4.1 Decoder block diagram 2
4.2 Overview over the codec building blocks 3
4.3 Efficient combination of decoder processing blocks in time domain and QMF domain 4
4.4 Rule set for determining processing domains 5
4.4.1 Audio Core Codec, Processing Domain 5
4.4.2 Mixing 6
4.4.3 Audio Core Codec, Interface Domain to Rendering 6
4.4.4 Rendering Context 6
4.4.5 Post-Processing Context 6
4.4.6 End-of-Chain Context 7

5 MPEG-H 3D Audio Core decoder 7
5.1 Terms and Definitions 7
5.1.1 Joint Stereo 7
5.1.2 MPEG Surround based stereo (MPS 212) 7
5.2 Syntax 7
5.2.1 General 7
5.2.2 Decoder configuration 7
5.2.3 MPEG-H 3D Audio Core bitstream payloads 22
5.3 Data Structure 30
5.3.1 General 30
5.3.2 General Configuration Data Elements 30
5.3.3 Loudspeaker Configuration Data Elements 32
5.3.4 Core Decoder Configuration Data Elements 34
5.3.5 Downmix Matrix Data Elements 37
5.3.6 HOA Rendering Matrix Data Elements 40
5.4 Configuration Element Descriptions 42
5.4.1 General 42
5.4.2 Downmix configuration 43
5.4.3 HOA rendering matrix configuration 47
5.5 Tool Descriptions 51
5.5.1 General 51
5.5.2 Quad Channel Element 52
5.5.3 Transform Splitting 53
5.5.4 MPEG Surround for Mono to Stereo upmixing 60
5.5.5 Enhanced Noise Filling 62
5.5.6 Audio Pre-Roll 82
5.6 Buffer requirements 86
5.6.1 Minimum decoder input buffer 86
5.6.2 Bit reservoir 86
5.6.3 Maximum bit rate 87
5.7 Stream Access Point requirements and inter-frame dependency 87

6 Dynamic Range Control and Loudness Processing 88

Documents
Document Sharing Hub

© ISO/IEC 2015 – All rights reserved

Full document details: ISO_IEC 23008-3_2015

6.1 Introduction ... 88
6.2 Description ... 88
6.3 Syntax ... 88
6.3.1 Loudness Metadata ... 88
6.3.2 Dynamic Range Control Metadata ... 89
6.3.3 Data Elements ... 90
6.4 Decoding Process ... 91
6.4.1 General ... 91
6.4.2 Dynamic Range Control ... 93
6.4.3 Usage of downmixId in MPEG-H ... 93
6.4.4 DRC Set Selection Process ... 94
6.4.5 DRC-1 for SAOC 3D Content ... 95
6.4.6 DRC-1 for HOA Content ... 96
6.4.7 Loudness Normalization ... 98
6.4.8 Peak Limiter ... 98
6.4.9 Time-Synchronization of DRC gains ... 98

7 Object Metadata Decoding ... 98
7.1 Introduction ... 98
7.2 Description ... 98
7.3 Syntax ... 99
7.3.1 Object Metadata Configuration ... 99
7.3.2 Top level object metadata syntax ... 100
7.3.3 Subsidiary payloads for efficient object metadata decoding ... 100
7.3.4 Subsidiary payloads for object metadata decoding with low delay ... 104
7.4 Data Structure ... 108
7.4.1 Definition of ObjectMetadataConfig() payloads ... 108
7.4.2 Efficient Object Metadata Decoding ... 108
7.4.3 Object Metadata Decoding with Low Delay ... 113

8 Object Rendering ... 117
8.1 Description ... 117
8.2 Terms and Definitions ... 117
8.3 Input data ... 117
8.4 Processing ... 119
8.4.1 Imaginary Loudspeakers ... 119
8.4.2 Dividing the Loudspeaker Setup into a Triangle Mesh ... 120
8.4.3 Rendering Algorithm ... 121

9 SAOC 3D ... 126
9.1 Description ... 126
9.2 Definitions ... 126
9.3 Delay and synchronization ... 127
9.4 Syntax ... 127
9.4.1 Payloads for SAOC 3D ... 127
9.4.2 Definition of SAOC 3D payloads ... 131
9.5 SAOC 3D processing ... 133
9.5.1 Compressed data stream decoding and dequantization of SAOC 3D data ... 133
9.5.2 Time/frequency tranforms ... 133
9.5.3 Signals and parameters ... 133
9.5.4 SAOC 3D decoding ... 135
9.5.5 Dual mode ... 140

10 Generic Loudspeaker Rendering/Format Conversion ... 141
10.1 Description ... 141
10.2 Definitions ... 142
10.2.1 General remarks ... 142
10.2.2 Variable definitions ... 142
10.3 Processing ... 143
10.3.1 Application of transmitted downmix matrices ... 143
10.3.2 Application of transmitted equalizer settings ... 148
10.3.3 Downmix processing involving multiple channel groups ... 148

Documents
Document Sharing Hub

© ISO/IEC 2015 – All rights reserved

Full document details: ISO_IEC 23008-3_2015

10.3.4 Initialization of the format converter 149
10.3.5 Audio signal processing 165

11 Immersive Loudspeaker Rendering / Format Conversion 171
11.1 Description 171
11.2 Syntax 172
11.3 Definitions 173
11.3.1 General remarks 173
11.3.2 Variable definitions 173

12 Higher Order Ambisonics (HOA) 221
12.1 Technical Overview 221
12.1.1 Block Diagram 221
12.1.2 Overview of the decoder tools 222
12.2 Syntax 223
12.2.1 Configuration of HOA elements 223
12.2.2 Payloads of HOA elements 224
12.3 Data Structure 229
12.3.1 Definitions of HOA Config 229
12.3.2 Definitions of HOA payload 231
12.4 HOA Tool Description 234
12.4.1 HOA Frame Converter 234
12.4.2 Spatial HOA decoding 243
12.4.3 HOA Renderer 255

13 Binaural Renderer 263
13.1 Introduction 263
13.2 Frequency-Domain Binaural Renderer 264
13.2.1 Introduction 264
13.2.2 Definitions 266
13.2.3 Parameterization of Binaural Room Impulse Responses 270
13.2.4 Frequency-Domain Binaural Processing 282
13.3 Time-Domain Binaural Renderer 289
13.3.1 Introduction 289
13.3.2 Definitions 290
13.3.3 Parameterization of Binaural Room Impulse Responses 291
13.3.4 Time-Domain Binaural Processing 296

14 MPEG-H 3D audio stream (MHAS) 297
14.1 Overview 297
14.2 Syntax 297
14.2.1 Main MHAS syntax elements 297
14.2.2 Subsidiary MHAS syntax elements 299
14.3 Semantics 299
14.3.1 mpeghAudioStreamPacket() 299
14.3.2 MHASPacketPayload() 300
14.4 Description of MHASPacketTypes 300
14.4.1 PACTYP_FILLDATA 300
14.4.2 PACTYP_MPEGH3DACFG 300
14.4.3 PACTYP_MPEGH3DAFRAME 301
14.4.4 PACTYP_SYNC 301
14.4.5 PACTYP_SYNCGAP 301
14.4.6 PACTYP_MARKER 301
14.4.7 PACTYP_CRC16 and PACTYP_CRC32 302
14.4.8 PACTYP_DESCRIPTOR 302
14.4.9 PACTYP_USERINTERACTION 302
14.4.10 PACTYP_LOUDNESS_DRC 302
14.4.11 PACTYP_BUFFERINFO 303
14.5 Application Examples 303
14.5.1 Light-weighted broadcast 303
14.5.2 MPEG-2 Transport Stream 303
14.6 Multi-Stream Delivery and Interface 304

Documents
Document Sharing Hub

© ISO/IEC 2015 – All rights reserved

Full document details: ISO_IEC 23008-3_2015

15 Metadata Audio Elements (MAE) ... 306
15.1 Introduction ... 306
15.2 Syntax ... 307
15.3 Semantics ... 311
15.4 Definition of mae_metaDataElementIDs ... 319

16 Loudspeaker Distance Compensation ... 319

17 Interfaces to the MPEG-H 3D audio decoder ... 320
17.1 General ... 320
17.2 Interface for local setup information ... 321
17.2.1 General ... 321
17.2.2 WIRE output ... 321
17.2.3 Syntax for local setup information ... 321
17.2.4 Semantics for local setup information ... 321
17.3 Interface for local loudspeaker setup and rendering ... 322
17.3.1 General ... 322
17.3.2 Syntax for local loudspeaker signaling ... 322
17.3.3 Semantics for local loudspeaker signaling ... 323
17.4 Interface for binaural room impulse responses (BRIRs) ... 324
17.4.1 Introduction ... 324
17.4.2 Syntax of Binaural Renderer Interface ... 324
17.4.3 Semantics ... 327
17.5 Interface for local screen size information ... 332
17.5.1 General ... 332
17.5.2 Syntax ... 332
17.5.3 Semantics ... 332
17.6 Interface for signaling of local zoom area ... 333
17.6.1 General ... 333
17.6.2 Syntax ... 333
17.6.3 Semantics ... 334
17.7 Interface for user interaction ... 334
17.7.1 Introduction ... 334
17.7.2 Definition of User Interaction Categories ... 334
17.7.3 Definition of an Interface for User Interaction ... 335
17.7.4 Syntax of interaction interface ... 335
17.7.5 Semantics of interaction interface ... 336
17.8 Interface for loudness normalization and dynamic range control (DRC) ... 338

18 Application and processing of local setup information and interaction data ... 338
18.1 *Element Metadata* Preprocessing ... 338
18.2 Interactivity Limitations and Restrictions ... 341
18.2.1 General Information ... 341
18.2.2 WIRE Interactivity ... 341
18.2.3 Position Interactivity ... 342
18.2.4 Screen-Related Element Remapping and Object Remapping for Zooming ... 342
18.2.5 Closest Speaker Playback ... 342
18.3 Screen-Related Element Remapping ... 342
18.4 Object Remapping for Zooming ... 344
18.5 Determination of the Closest Speaker ... 345

Annex A (normative) Tables for arithmetic decoding of IGF information ... 347
A.1 cf_se01[27] ... 347
A.2 cf_se10[27] ... 347
A.3 cf_se02[7][27] ... 347
A.4 short cf_se20[7][27] ... 347
A.5 short cf_se11[7][7][27] ... 348
A.6 cf_off_se01 ... 349
A.7 cf_off_se10 ... 349
A.8 cf_off_se02[7] ... 349
A.9 short cf_off_se20[7] ... 350
A.10 cf_off_se11[7][7] ... 350

Documents
Document Sharing Hub

© ISO/IEC 2015 – All rights reserved

Full document details: ISO_IEC 23008-3_2015

Annex B (normative) **SAOC 3D Decorrelator pre-mixing matrices** **351**
B.1 Premixing matrix for output configurations with small number of output channels **351**
B.2 Premixing matrix for 22.2 output configuration **351**
B.3 Algorithm for generating pre-mixing matrices **352**
B.3.1 Input to the algorithm and representations **352**
B.3.2 Algorithm steps **352**

Annex C (informative) **Encoder Tools** **356**
C.1 General Overview **356**
C.1.1 Encoder block diagram **356**
C.1.2 Overview of the encoder and decoder building blocks **356**
C.2 Core Encoder Tools **357**
C.2.1 Quad Channel Element **357**
C.2.2 Transform Splitting **358**
C.2.3 Calculation of Residual Signal for MPEG Surround with Hybrid Residual Coding **359**
C.2.4 Enhanced Noise Filling **359**
C.3 Object Metadata Encoding **360**
C.3.1 Pre-Processing of the Object Metadata **360**
C.3.2 Efficient Object Metadata Encoding **361**
C.3.3 Object Metadata Encoding with Low Delay **361**
C.3.4 Spatially skipping objects **361**
C.4 SAOC 3D Encoder **361**
C.4.1 Overview **361**
C.4.2 Calculation of the SAOC 3D parameters **361**
C.4.3 Time/frequency transform **362**
C.4.4 Framing **362**
C.4.5 Parameter quantization and coding **362**

Annex D (informative) **Peak limiter for unguided clipping prevention** **384**

Annex E (normative) **Compact Template Downmix Matrices** **385**

Annex F (normative) **HOA Tables** **386**
F.6 32 Uniformly Distributed Positions in Spherical Coordinates **390**
F.12 Table of 256x8 weighting values, WeightValCdbk **404**

Annex G (informative) **Low Complexity HOA Rendering** **421**
G.1 Tool Description **421**
G.2 Predominant Sound Rendering **421**
G.3 Ambient Sound Rendering **422**
G.4 Output Signal Composition **422**

Annex H (informative) **Information on delay and complexity of Time-Domain binauralization** **423**
H.1 Complexity and latency **423**
H.1.1 Algorithm description **423**
H.1.2 Complexity **423**
H.1.3 Latency **424**
H.2 Experimental results **425**
H.3 Alternative low-delay implementations **426**

Bibliography **428**

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

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 on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword - Supplementary information

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

ISO/IEC 23008 consists of the following parts, under the general title *Information technology — High efficiency coding and media delivery in heterogeneous environments*:

— *Part 1:* MPEG media transport (MMT)

— *Part 2:* High efficiency video coding

— *Part 3: 3D audio*

— *Part 4: MMT Reference and Conformance Software*

— *Part 5: Reference software for high efficiency video coding*

— *Part 8: HEVC conformance testing*

— *Part 10: MPEG media transport forward error correction (FEC) codes*

— *Part 11: MPEG media transport composition information*

— *Part 12: Image file format*

— *Part 13: MMT Implementation Guidelines*

Documents
Document Sharing Hub

© ISO/IEC 2015 – All rights reserved

Full document details: ISO_IEC 23008-3_2015

This corrected version of ISO 23008-3:2015 incorporates the following corrections: unreadable equations have been improved.

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

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 part of ISO/IEC 23008-3 “High Efficiency Coding and Media Delivery in Heterogeneous Environments — Part 3: 3D Audio” 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.

The main focus of this specification are 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 speakers built into the device and headphone listening are maintained.

© ISO/IEC 2015 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

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

1 Scope

This part of ISO/IEC 23008-3 specifies technology which supports the efficient transmission of 3D audio signals and flexible rendering for the playback of 3D audio in a wide variety of listening scenarios. These include 3D home theater setups, 22.2 loudspeaker systems, automotive entertainment systems and playback over headphones connected to a tablet or smartphone.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensible for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 13818-1:2013, *Information technology — Generic Coding of moving pictures and associated audio information: 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 23001-8:2013, *Information technology — MPEG systems technologies — Part 8: Coding-independent code-points*

ISO/IEC 23001-8:2013/Amd.1, *Information technology — MPEG systems technologies — Part 8: Coding-independent code-points, AMENDMENT 1: New audio code points*

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

ISO/IEC 23003-2:2010, *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*

3 Terms, definitions and mnemonics

3.1 Terms and Definitions

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

3.2 Mnemonics

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

© ISO/IEC 2015 – All rights reserved

Documents Document Sharing Hub

Full document details: ISO_IEC 23008-3_2015

bslbf	Bit string, left bit first, where “left” is the order in which bit strings are written in ISO/IEC 14496. 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.

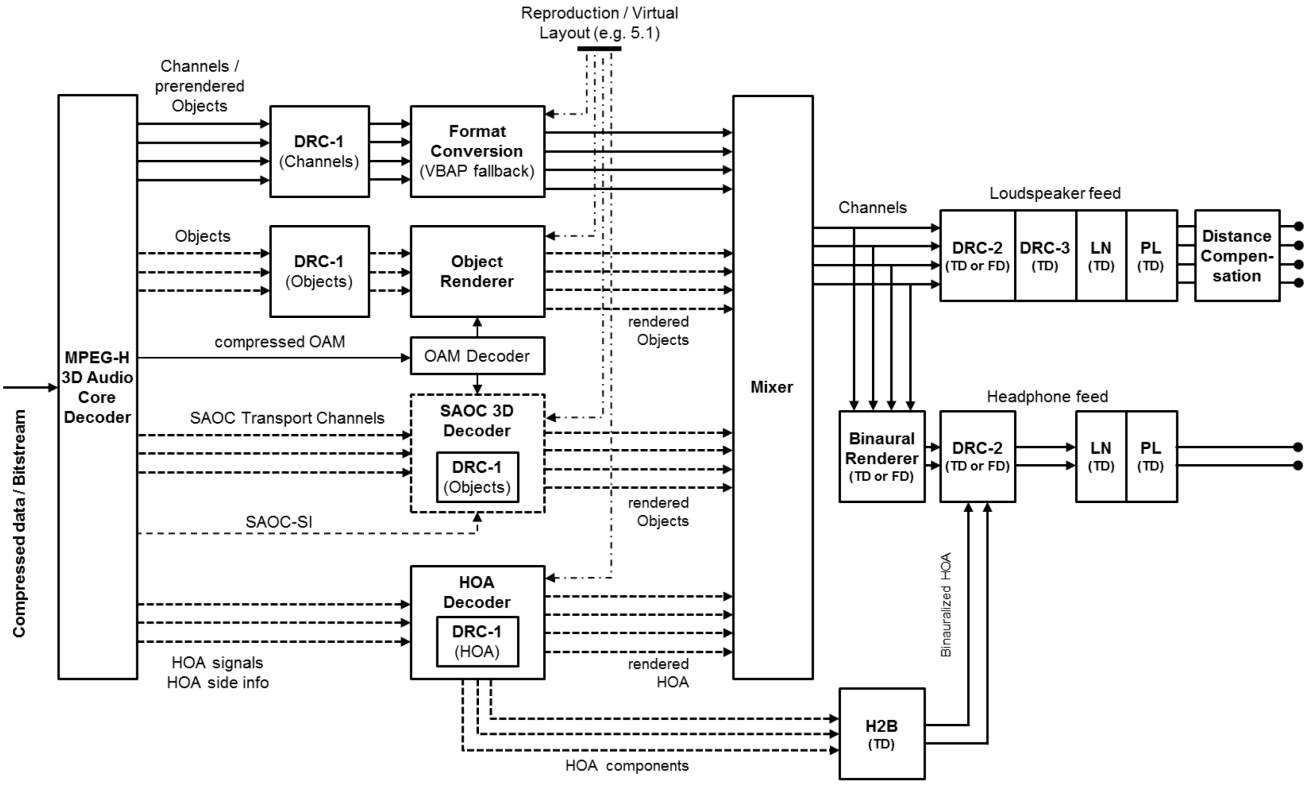


Figure 1 — Block diagram of the 3D-Audio decoder.
(DRC: Dynamc Range Control, SAOC: Spatial Audio Object Coding, HOA: Higher Order Ambisonics, LN: Loudness Normalization, PL: Peak Limiter)

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

4.2 Overview over the codec building blocks

The MPEG-H 3DA 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.

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.

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

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 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.

Virtual layout information fed from an application shall be consistent with the corresponding BRIR set provided (expressed with MeasurementSetup, see in Table 157), 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 time domain and QMF domain

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

— Block operates in time domain (**TD**)

— Block operates in frequency domain (**FD**) – also called QMF domain

— 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 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 2.

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

Processing Context	Functional Block	Processing Domain
Audio Core	MPEG-H 3D Audio Core Coder	FD or TD[1)]
Rendering	DRC-1	if multiband: FD else: neutral
	Format Converter (FC)	FD
	Object Renderer	neutral
	SAOC 3D Decoder	FD
	HOA Decoder	TD
Mixing	Mixer	neutral
Post-processing	DRC-2	if multiband: FD else: neutral
	FD Binauralizer	FD
	TD Binauralizer	TD
End of chain	DRC-3 (only singleband)	TD
	Loudness Normalization	TD
	Peak Limiter	TD
	LS Distance Compensation	TD

[1)] depends on bit stream configuration

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

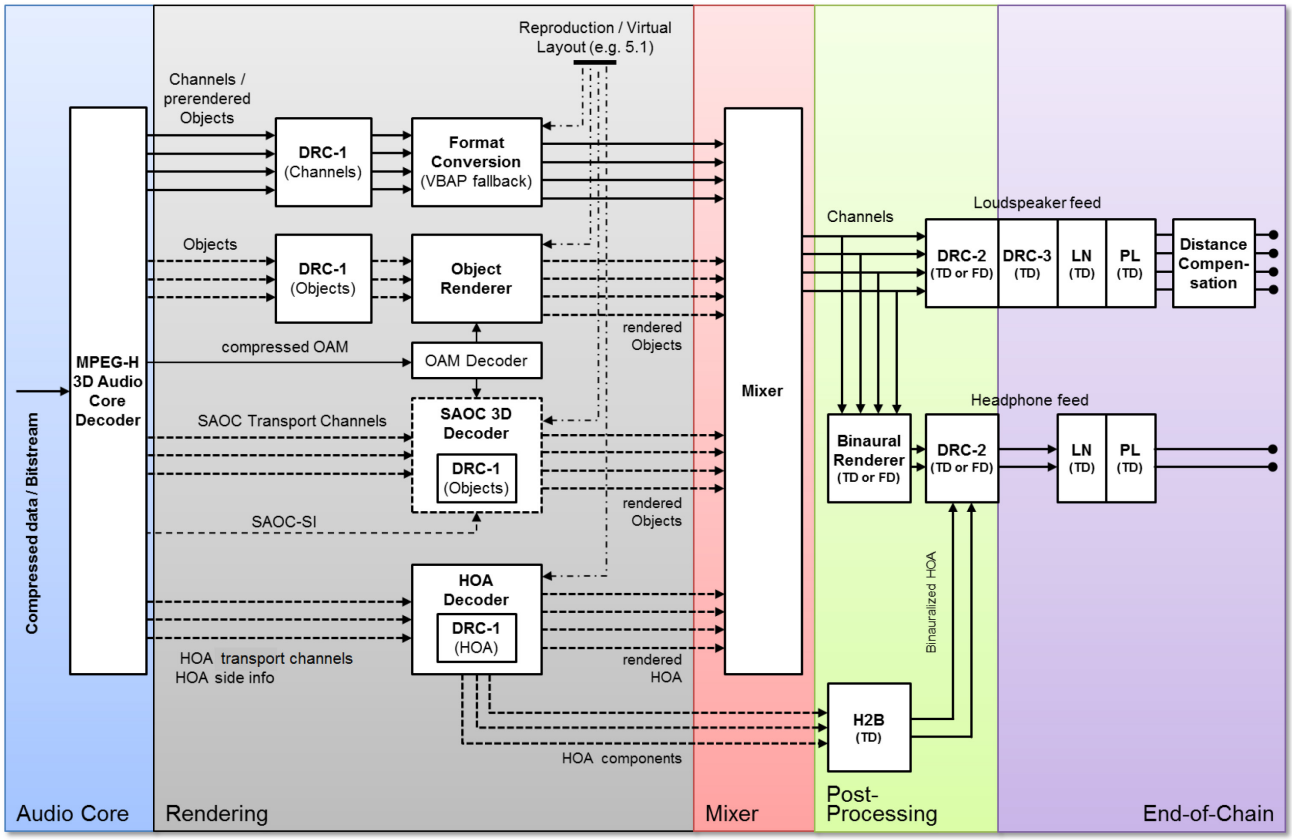


Figure 2 — MPEG-H 3D audio decoder overview with signal processing context

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 can be connected.

4.4 Rule set for determining processing domains

4.4.1 Audio Core Codec, Processing Domain

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

- If last decoding stage is in QMF domain (SBR and/or MPS212 active) (typical for mid to low bit rate coding)
  - formal Audio Core Codec processing domain is **FD**

- If last decoding stage is **not** in QMF domain (neither SBR nor MPS212 active) (typical for high rate coding)
  - formal Audio Core Codec processing domain is **TD**

© ISO/IEC 2015 – All rights reserved

Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

4.4.2 Mixing

— If the Audio Core Codec processing domain is **TD**

  — If any block in Rendering Context requires operation in **FD**
  AND any block in Post-processing Context requires operation in **FD**
    — then: Mixer shall operate in **FD**

  — else:
    — Mixer shall operate in **TD**

— If the Audio Core Codec processing domain is **FD**

  — If any block in Rendering Context requires operation in **TD**
  AND **no** blocks in Post-processing Context require operation in **FD**
    — then: Mixer shall operate in **TD**

  — else:
    — Mixer shall operate in **FD**

4.4.3 Audio Core Codec, Interface Domain to Rendering

— If the Audio Core Codec processing domain is **TD**,

  — If subsequent 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 subsequent rendering block requires operation in **TD**
    — apply QMF synthesis and output **TD** for this rendering block

  — else
    — output **FD**

4.4.4 Rendering Context

Rendering paths are considered individually:

— If Rendering block operates in **FD** and Mixer operates in **TD**
  — apply QMF synthesis after renderer and output **TD**

— If Rendering block operates in **TD** and Mixer operates in **FD**
  — apply QMF analysis after renderer and output **FD**

— else
  — keep processing domain

4.4.5 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

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

- 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

4.4.6 End-of-Chain Context

All blocks shall operate in **TD**.

5 MPEG-H 3D Audio Core decoder

5.1 Terms and Definitions

5.1.1 Joint Stereo

The MDCT domain based joint stereo coding tool with the possibility of Complex Stereo Prediction 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 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, Annex B.21.

5.2 Syntax

5.2.1 General

The bit stream syntax is based on ISO/IEC 23003-3:2012, clause 5.

Modifications and amendments to the existing bit stream 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 2 — Syntax of mpegh3daConfig()

Syntax	No. of bits	Mnemonic
mpegh3daConfig()		
{		
mpegh3daProfileLevelIndication	**8**	**uimsbf**
usacSamplingFrequencyIndex;	**5**	**bslbf**
if (usacSamplingFrequencyIndex == 0x1f) {		
usacSamplingFrequency;	**24**	**uimsbf**
}		
coreSbrFrameLengthIndex;	**3**	**uimsbf**
coreQmfDelayCompensation;	**1**	**uimsbf**
receiverDelayCompensation;	**1**	**uimsbf**

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

referenceLayout = SpeakerConfig3d();		
FrameworkConfig3d();		
mpegh3daDecoderConfig();		
if (**usacConfigExtensionPresent**) {	**1**	**uimsbf**
mpegh3daConfigExtension();		
}		
}		

Table 3 — Syntax of FrameworkConfig3d()

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

Table 4 — 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;		
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;		
}		
if (SignalGroupType[grp] == SignalGroupTypeHOA) {		
numHOATransportChannels += bsNumberOfSignals[grp] + 1;		
}		
}		

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

```
}
```

5.2.2.2 Loudspeaker Configuration Syntax

Table 5 — 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 6 — 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°)) {		NOTE
alsoAddSymmetricPair;	**1**	**uimsbf**
if (alsoAddSymmetricPair) {		
(*also add the speaker with the opposite AzimuthDirection*);		
i++;		
}		
}		
}		
}		
NOTE: The value of AzimuthAngle can be calculated using Table 7.		

Table 7 — Syntax of mpegh3daSpeakerDescription()

Syntax	No. of bits	Mnemonic
mpegh3daSpeakerDescription()		
{		
isCICPspeakerIdx	**1**	**uimsbf**
if (isCICPspeakerIdx) {		
CICPspeakerIdx;	**7**	**uimsbf**
}		
else {		
ElevationClass;	**2**	**uimsbf**

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

if (ElevationClass == 3) {		
ElevationAngleIdx;	**5, 7**	**uimsbf**
if (ElevationAngle != 0°) {		NOTE
ElevationDirection;	**1**	**uimsbf**
}		
}		
AzimuthAngleIdx;	**6, 8**	**uimsbf**
if ((AzimuthAngle != 0°) && (AzimuthAngle != 180°)) {		NOTE
AzimuthDirection;	**1**	**uimsbf**
}		
isLFE;	**1**	**uimsbf**
}		
}		

NOTE: The number of bits for ElevationAngleIdx and AzimuthAngleIdx depends on the value of angularPrecision according to Table 42. The value of ElevationAngle and AzimuthAngle can be derived from Table 44 and Table 46, respectively. In case isCICPspeakerIdx is one, ElevationAngle and AzimuthAngle are signaled by means of a LoudspeakerGeometry as defined in ISO/MPEG 23001-8.

5.2.2.3 Core Decoder Configuration

Table 8 — Syntax of mpegh3daDecoderConfig()

Syntax	No. of bits	Mnemonic
mpegh3daDecoderConfig()		
{		
numElements = escapedValue(4,8,16) + 1**;**		
elementLengthPresent	**1**	
for (elemIdx=0; elemIdx<numElements; ++elemIdx) {		
usacElementType[elemIdx**]**	**2**	**uimsbf**
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;		
}		
}		

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

Table 9 — Syntax of mpegh3daSingleChannelElementConfig()

Syntax	No. of bits	Mnemonic
mpegh3daSingleChannelElementConfig(sbrRatioIndex)		
{		

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

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

Table 10 — Syntax of mpegh3daChannelPairElementConfig()

Syntax	No. of bits	Mnemonic
mpegh3daChannelPairElementConfig(sbrRatioIndex)		
{		
mpegh3daCoreConfig();		
if (enhancedNoiseFilling) {		
igfIndependentTiling;	**1**	**bslbf**
}		
if (sbrRatioIndex > 0) {		
SbrConfig();		
stereoConfigIndex;	**2**	**uimsbf**
} else {		
stereoConfigIndex = 0;		
}		
if (stereoConfigIndex > 0) {		
Mps212Config(stereoConfigIndex);		
}		
qceIndex;	**2**	**uimsbf**
If(qceIndex > 0) {		
shiftIndex0;	**1**	**uimsbf**
if(shiftIndex0 > 0) {		
shiftChannel0;	**nBits**[1)]	
}		
}		
shiftIndex1;	**1**	**uimsbf**
if(shiftIndex1 > 0) {		
shiftChannel1;	**nBits**[1)]	
}		
}		

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

Table 11 — Syntax of mpegh3daCoreConfig()

Syntax	No. of bits	Mnemonic
mpegh3daCoreConfig()		
{		
tw_mdct;	**1**	**bslbf**
noiseFilling;	**1**	**bslbf**
if (**enhancedNoiseFilling**) {	**1**	**bslbf**
igfUseEnf;	**1**	**bslbf**
igfUseWhitening;	**1**	**bslbf**
igfAfterTnsSynth;	**1**	**bslbf**
igfStartIndex;	**4**	**uimsbf**
igfUseHighRes;	**1**	**bslbf**
igfStopIndex;	**3**	**uimsbf**
}		
}		

© ISO/IEC 2015 – All rights reserved

Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

Table 12 — Syntax of mpegh3daLfeElementConfig()

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

Table 13 — Syntax of mpegh3daExtElementConfig()

Syntax	No. of bits	Mnemonic
mpegh3daExtElementConfig() { usacExtElementType = escapedValue(4, 8, 16); usacExtElementConfigLength = escapedValue(4, 8, 16); if (**usacExtElementDefaultLengthPresent**) { usacExtElementDefaultLength = escapedValue(8, 16, 0) + 1; } else { usacExtElementDefaultLength = 0; } **usacExtElementPayloadFrag;** 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; default: while (usacExtElementConfigLength--) { **tmp;** }	**1** **1** NOTE **8**	**uimsbf** **uimsbf** **uimsbf**

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

```
            break;
        }
    }
```

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

Table 14 — 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; default: while (usacConfigExtLength[confExtIdx]--) { **tmp;**	**8**	**uimsbf**
} break; } } }		

Table 15 — References to USAC configuration syntactic elements

Syntax of	**defined in**
escapedValue()	ISO/IEC 23003-3:2012, 5.2, Table 16
SbrConfig()	ISO/IEC 23003-3:2012, 5.2, Table 11
Mps212Config()	ISO/IEC 23003-3:2012, 5.2, Table 13
SAOCSpecificConfig()	ISO/IEC 23003-2:2010, 6.1, Table 5
SpatialSpecificConfig()	ISO/IEC 23003-1:2007, 5.1, Table 5

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

5.2.2.4 Syntax of downmix matrix elements

Table 16 — Syntax of downmixConfig()

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

Table 17 — Syntax of DownmixMatrixSet()

Syntax	No. of bits	Mnemonic
DownmixMatrixSet() { **downmixIdCount**; for (k=0; k< downmixIdCount; ++k) { **downmixId;** **downmixType**; if (downmixType == 0) { **CICPspeakerLayoutIdx;** } else if (downmixType == 1) { **CICPspeakerLayoutIdx;** **bsDownmixMatrixCount[k]** = escapedValue(1,3,0); for (l=0; l< bsDownmixMatrixCount[k]+1; ++l) { **bsNumAssignedGroupIDs[k][l]** = escapedValue(1,4,4); for (m=0; m<bsNumAssignedGroupIDs[k][l]+1; ++m) { **signal_groupID[k][l][m];** } **DmxMatrixLenBits** = escapedValue(8,8,12); DownmixMatrix(inputConfig(audioChannelConfig), inputCount(audioChannelConfig), outputConfig(CICPspeakerLayoutIdx), outputCount(CICPspeakerLayoutIdx)); } } } }	 **5** **7** **2** **6** **6** **1..4** **1..9** **5** **8..28** **DmxMatrixLenBits**	 **uimsbf** **uimsbf** **uimsbf** **uimsbf** **uimsbf** **uimsbf**

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

Table 18 — 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; } }		
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 {		

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

```
        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);
    }
    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;
```


© ISO/IEC 2015 – All rights reserved

Documents Document Sharing Hub

Full document details: ISO_IEC 23008-3_2015

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

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

Table 19 — 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**;	**varies**	**bslbf**
/* limited Golomb-Rice using p = gainLGRParam and N = gainTableSize*/ gainValue = gainTable[gainValueIndex]; } if (gainValue < minGain) gainValue = -infinity; return gainValue; }		

Table 20 — 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 21 — 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;		

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

```
                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 22 — Syntax of HoaRenderingMatrixSet()

Syntax	No. of bits	Mnemonic
HoaRenderingMatrixSet()		
{		
numHoaRenderingMatrices;	**5**	**uimsbf**
for (k=0; k< numHoaRenderingMatrices; ++k) {		
HoaRenderingMatrixId;	**7**	**uimsbf**
CICPspeakerLayoutIdx;	**6**	**uimsbf**
HoaMatrixLenBits = escapedValue(8,8,12);	**8..28**	
HoaRenderingMatrix(NumOfHoaCoeffs, outputConfig(CICPspeakerLayoutIdx), outputCount(CICPspeakerLayoutIdx));	**HoaMatrixLenBits**	
}		
}		

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

Table 23 — 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;		
precisionLevel	**2**	**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**) {	**1**	**uimsbf**
firstSparseOrder	**1**	**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 {		
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 { /* isAnyValueSymmetric==0 */		
if { **isAnySignSymmetric**) {	**1**	**uimsbf**

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

```
            for (i=0; i<numPairs; ++i)
                signSymmetricPairs[i] = boolVal                    1          uimsbf
        }
    }
    hasVerticalCoef;                                               1          uimsbf
    DecodeHoaMatrixData()
}
```

Table 24 — 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**
} 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 */ } }		

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

```
                    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);
            }
        }
}
```

Table 25 — 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; }		

5.2.3 MPEG-H 3D Audio Core bitstream payloads

5.2.3.1 Payloads for MPEG-H 3D Audio Core

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

Table 26 — Syntax of mpegh3daFrame(), top level payload for MPEG-H 3D Audio Core

Syntax	No. of bits	Mnemonic
mpegh3daFrame() {		
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:		

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

```
                mpegh3daSingleChannelElement(usacIndependencyFlag);    elementLength, NOTE 1
                break;
            case ID_USAC_CPE:
                mpegh3daChannelPairElement(usacIndependencyFlag);      elementLength, NOTE 1
                break;
            case ID_USAC_LFE:
                mpegh3daLfeElement(usacIndependencyFlag);              elementLength, NOTE 1
                break;
            case ID_USAC_EXT:
                mpegh3daExtElement(usacIndependencyFlag);
                break;
        }
    }
```

NOTE 1: If present, elementLength represents the length of the corresponding element it refers to in number of bits.

Table 27 — Syntax of mpegh3daSingleChannelElement()

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

Table 28 — 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; } else { nrSbrChannels = 1; } UsacSbrData(nrSbrChannels, indepFlag); } if (stereoConfigIndex > 0) { Mps212Data(indepFlag); } }		

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

Table 29 — Syntax of mpegh3daLfeElement()

Syntax	No. of bits	Mnemonic
mpegh3daLfeElement(indepFlag) { fd_channel_stream(0,0,0,0, indepFlag); }		

Table 30 — Syntax of UsacExtElement()

Syntax	No. of bits	Mnemonic
mpegh3daExtElement(indepFlag) {		
usacExtElementPresent	**1**	**uimsbf**
if (usacExtElementPresent==1) {		
usacExtElementUseDefaultLength;	**1**	**uimsbf**
if (usacExtElementUseDefaultLength) {		
usacExtElementPayloadLength = usacExtElementDefaultLength;		
} else {		
usacExtElementPayloadLength = escapedValue(8,16,0);		
}		
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 31 — 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**
}		
}		

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

5.2.3.2 Subsidiary payloads

Table 32 — 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); } for (ch=0; ch<nrChannels; ch++) { if (core_mode[ch] == 1) { lpd_channel_stream(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, indepFlag); } } }		

Table 33 — 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) {		

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

```
            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;                             [1]         [2]
                    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 {
                for (g = 0; g < num_window_groups; g++) {
                    for (sfb = max_sfb_ste; sfb < igf_sfb_stop; sfb++) {
                        alpha_q_re[g][sfb] = 0;                         [3]         [4]
                        alpha_q_im[g][sfb] = 0;
                    }
                }
            }
        }
    }
    if (tw_mdct) {
        if (common_tw) {                                                1           uimsbf
            tw_data();
        }
    }
    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;
```


© ISO/IEC 2015 – All rights reserved

Documents Document Sharing Hub

Full document details: ISO_IEC 23008-3_2015

```
            }
            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_tw = 0;
    }
}
```

Table 34 — Syntax of fd_channel_stream()

Syntax	No. of bits	Mnemonic
fd_channel_stream(common_window, common_tw, tns_data_present, noiseFilling, indepFlag)		
{		
global_gain;	**8**	**uimsbf**
if (noiseFilling) {		
noise_level;	**3**	**uimsbf**
noise_offset;	**5**	**uimsbf**
} else {		
noise_level = 0;		
}		
if (!common_window) {		
ics_info();		
}		
if (tw_mdct && !common_tw) {		
tw_data();		
}		
scale_factor_data();		
if (enhancedNoiseFilling) {		
igf_AllZero	**1**	**uimsbf**
igf_level(igf_AllZero**,** indepFlag**);**	**0…**	NOTE
if (!igf_AllZero) {		
igf_data(indepFlag);		
} else {		
igfPrevTileIdx = {3};		
igf_PrevWhiteningLevel = {0};		
}		
}		
if (tns_data_present) {		

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

```
        tns_data();
    }
    ac_spectral_data(indepFlag);

    if (fac_data_present) {                                          1          uimsbf
        fac_length = (window_sequence == EIGHT_SHORT_SEQUENCE) ? ccfl/16 : ccfl/8;
        fac_data(1, fac_length);
    }
}
```

NOTE: For details on **igf_level()** see 5.5.5.

Table 35 — 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;	**[5]**	[6]
}		
alpha_q_im[g][sfb] = 0;		
}		
}		
}		

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

Table 36 — Syntax of igf_data()

Syntax	No. of bits	Mnemonic
igf_data(indepFlag)		
{		
if (!indepFlag) {		
if (**igf_UsePrevTileIdx**) {	**1**	**uimsbf**
for (i = 0; i < igfNTiles; i++) {		
igfCurrTileIdx[i] = igfPrevTileIdx[i];		
}		
}		
} else {		
for (i = 0; i < igfNTiles; i++) {		
igfCurrTileIdx[i];	**2**	**uimsbf**
}		
}		
for (i = igfNTiles; i < 4; i++) {		
igfCurrTileIdx[i] = 3;		
igfPrevTileIdx[i] = 3;		
}		
for (i = 0; i < igfNTiles; i++) {		
igfPrevTileIdx[i] = igfCurrTileIdx[i];		
}		
if (igfUseWhitening && window_sequence != EIGHT_SHORT_SEQENCE) {		
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 = igfNTiles; i < 4; i++) {		
igf_WhiteningLevel[i] = 0;		
igf_PrevWhiteningLevel[i] = 0;		
}		
if (igfUseEnf) {		
If (window_sequence != EIGHT_SHORT_SEQENCE) {		
isIGFEnfFlat;	**1**	**uimsbf**

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

```
        }
    }
}
```

Table 37 — 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
lpd_channel_stream()	ISO/IEC 23003-3:2012, 5.3, Table 31
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 Payloads

Table 38 — Syntax of AudioPreRoll()

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

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 directly follow 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 39.

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

Table 39 — Value of mpegh3daProfileLevelIndication

value	Indication of Profile	Indication of Level
0x00	reserved for ISO use	-
0x01-0xff	reserved for future profile definition	

coreQmfDelayCompensation This flag forces the MPEG-H 3D Audio Core coder to apply a delay compensation for the case that eSBR is not applied. If its value is 0, no delay compensation shall be applied. If its value is 1, a delay of 577 samples shall be applied to all audio elements which do not use eSBR, i.e. which do not employ a QMF analysis during decoding.
Further, if processing blocks immediately following the MPEG-H 3D Audio Core require QMF domain input, the delay is bypassed and the output is fed to a QMF analysis instead, such that the MPEG-H 3D Audio Core decoder and the subsequent processing blocks can be efficiently combined in the QMF domain.
Audio elements which *do* employ eSBR are not affected.

receiverDelayCompensation This flag forces the format converter to apply a delay compensation for the case that it is not active. When its value is 0, no delay compensation shall be applied in the bypass path. When its value is 1, a delay of 4033 samples shall be applied, when the format converter is fed with time-domain input, a delay of 2432 samples (corresponding to 38 QMF time slots) shall be applied, when the format converter is fed with QMF domain input.

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 must 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.

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

Table 40 — 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 bit stream field shall correspond to the number of channels as signaled 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 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.

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 41

Table 41 — Meaning of speakerLayoutType

Value	Meaning
0	Loudspeaker layout ist signaled by means of ChannelConfiguration index as defined in ISO/IEC 23001-8.
1	Loudspeaker layout ist signaled by means of a list of LoudspeakerGeometry indices as defined in ISO/IEC 23001-8
2	Loudspeaker layout is signaled by means of a list of explicit geometric position information.
3	*Reserved*

CICPspeakerLayoutIdx ChannelConfiguration value as defined in ISO/MPEG 23001-8

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

CICPspeakerIdx LoudspeakerGeometry value as defined in ISO/MPEG 23001-8

angularPrecision This flag signals the angular precision of loudspeaker geometry information according to Table 42

Table 42 — Meaning of angularPrecision

Value of angularPrecision	angular precision in degrees (°)	Number of bits used for coding of bit stream 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 if the loudspeaker position is signaled 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 43.

Table 43 — 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 signaled explicitly	N.A.

ElevationAngleIdx Index for calculating the elevation (i.e. vertical) angle for a given loudspeaker according to Table 44.

Table 44 — 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 45.

Table 45 — Meaning of ElevationDirection

Value of ElevationDirection	Direction
0	Upwards
1	Downwards

© ISO/IEC 2015 – All rights reserved

Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

AzimuthAngleIdx Index for calculating the azimuth (i.e. horizontal) angle for a given loudspeaker according to Table 46.

Table 46 — 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 47

Table 47 — 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 48. The qceIndex shall be the same for the two subsequent elements forming one QCE.

Table 48 — 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.

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

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.

isIgfEnfFlat Flag indicating the application of IGF-ENF 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 49.

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 49 — 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 bit stream extensions types. The meaning of usacExtElementType is defined in Table 50.

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

Table 50 — 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
/* reserved for ISO use */	9-127
/* 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 51.

Table 51 — Interpretation of data blocks for extension payload decoding

usacExtElementType	The concatenated usacExtElementSegmentData represents:
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	object_metadata()
ID_EXT_ELE_SAOC_3D	Saoc3DFrame()
ID_EXT_ELE_HOA	HOAFrame()
ID_EXT_ELE_FMT_CNVRTR	FormatConverterFrame()
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 52.

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

Table 52 — 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
/* reserved for ISO use */	5-127
/* 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 bit stream element

downmixId This field uniquely defines an ID for a default downmix matrix available on 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 clause 5.7.

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

Table 53 — Value of downmixType

downmixType	meaning
0	Format conversion with default downmix matrix available on 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 corresponds to **ChannelConfiguration** as defined in ISO/IEC 23001-8.

DmxMatrixLenBits length of the following bit stream element in bits

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

paramConfig, inputConfig, outputConfig	Channel configuration vectors specifying the information about each speaker. 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 speaker azimuth angle; AzimuthDirection, the direction of increasing azimuth, 0 (left) or 1 (right); ElevationAngle, the absolute value of the speaker elevation angle; ElevationDirection, the direction of increasing elevation, 0 (up) or 1 (down); isLFE, indicates whether the speaker is a LFE speaker.
paramCount, inputCount, outputCount	Number of speakers in the corresponding channel configuration vectors.
compactParamConfig, compactInputConfig, compactOutputConfig	Compact channel configuration vectors specifying the information about each speaker group. Each entry, compactParamConfig[i], is a structure with the members: pairType, type of the speaker group, which can be SYMMETRIC (a symmetric pair of two speakers), CENTER, or ASYMMETRIC; isLFE, indicates whether the speaker group consists of LFE speakers; originalPosition, position in the original channel configuration of the first (i.e. left) speaker, or the only speaker, in the group; symmetricPair.originalPosition, position in the original channel configuration of the second (i.e. right) speaker in the group, for SYMMETRIC groups only.
compactParamCount, compactInputCount, compactOutputCount	Number of speaker 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 54.

Table 54 — 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 speaker groups satisfy the

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

	separability property.
isSeparable[i]	Boolean indicating whether the output speaker group with index i satisfies the separability property.
isAllSymmetric	Boolean indicating whether all the output speaker groups satisfy the symmetry property.
isSymmetric[i]	Boolean indicating whether the output speaker group with index i satisfies the symmetry property.
mixLFEOnlyToLFE	Boolean indicating whether the LFE speakers are mixed only to LFE speakers and, at the same time, the non-LFE speakers are mixed only to non-LFE speakers.
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 speaker group i and output speaker 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 followeed 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 speaker group; When enabled, every asymmetric input speaker group will have two gain values decoded for each symmetric output speaker 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

© ISO/IEC 2015 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015

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 bit stream element.

HoaRenderingMatrixId This field uniquely defines an ID for a default HOA rendering matrix available on decoder side or a transmitted HOA rendering matrix.

CICPspeakerLayoutIdx This value describes the output loudspeaker layout for the given HOA rendering matrix. The value corresponds to **ChannelConfiguration** as defined in ISO/IEC 23001-8.

HoaRenderingMatrixLenBits Length of the following bit stream 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 speaker azimuth angle;
AzimuthDirection, direction of increasing azimuth, 0 (left) or 1 (right);
ElevationAngle, the absolute value of the speaker elevation angle;
ElevationDirection, direction of increasing elevation, 0 (up) or 1 (down);
isLFE, indicates whether the speaker is a LFE speaker.

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

pairType, can be SYMMETRIC (a symmetric pair of two speakers), CENTER, or ASYMMETRIC;
symmetricPair->originalPosition, position in the original channel configuration of the second (i.e. right) speaker 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.

precisionLevel Precision used for uniform quantization of the gains according to Table 55.

© ISO/IEC 2015 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC 23008-3_2015