
**Information technology — MPEG
audio technologies —**

**Part 3:
Unified speech and audio coding**

*Technologies de l'information — Technologies audio MPEG —
Partie 3: Codage unifié parole et audio*





COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2020

All rights reserved. Unless otherwise specified, or required in the context of its implementation, 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
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword.....	vii
Introduction.....	viii
1 Scope.....	1
2 Normative references.....	1
3 Terms, definitions, symbols and abbreviated terms	1
3.1 Terms and definitions	1
3.2 Symbols and abbreviated terms.....	3
4 Technical overview	4
4.1 Decoder block diagram.....	4
4.2 Overview of the decoder tools.....	5
4.3 Combination of USAC with MPEG Surround and SAOC.....	9
4.4 Interface between USAC and systems.....	9
4.4.1 Decoder behaviour	10
4.5 USAC profiles and levels	10
4.5.1 General.....	10
4.5.2 MPEG-4 HE AACv2 compatibility	11
4.5.3 Baseline USAC profile	12
4.5.4 Extended high efficiency AAC profile.....	13
4.6 Combination of USAC with MPEG-D DRC	14
5 Syntax	15
5.1 General.....	15
5.2 Decoder configuration (UsacConfig).....	15
5.3 USAC bitstream payloads	20
5.3.1 Payloads for audio object type USAC	20
5.3.2 Subsidiary payloads.....	23
5.3.3 Payloads for enhanced SBR.....	34
5.3.4 Payloads for MPEG Surround.....	43
5.3.5 Payload of extension elements	53
6 Data structure	53
6.1 USAC configuration.....	53
6.1.1 Definition of elements	53
6.1.2 UsacConfig()	63
6.1.3 Usac Output Sampling Frequency	63
6.1.4 UsacChannelConfig()	63
6.1.5 UsacDecoderConfig()	64
6.1.6 UsacSingleChannelElementConfig().....	64
6.1.7 UsacChannelPairElementConfig()	64
6.1.8 UsacLfeElementConfig()	64
6.1.9 UsacCoreConfig().....	64
6.1.10 SbrConfig()	65
6.1.11 SbrDfltHeader()	65
6.1.12 Mps212Config()	65
6.1.13 UsacExtElementConfig().....	65
6.1.14 UsacConfigExtension()	65
6.1.15 Unique stream identifier (Stream ID)	66
6.2 USAC payload.....	66
6.2.1 Definition of elements	66
6.2.2 UsacFrame().....	68
6.2.3 UsacSingleChannelElement()	69
6.2.4 UsacExtElement().....	69
6.2.5 UsacChannelPairElement()	70
6.2.6 Low frequency enhancement (LFE) channel element, UsacLfeElement()	70
6.2.7 UsacCoreCoderData()	71

6.2.8	StereoCoreToolInfo()	71
6.2.9	fd_channel_stream() and ics_info()	72
6.2.10	lpd_channel_stream()	76
6.2.11	Spectral noiseless coder	79
6.2.12	Enhanced SBR	80
6.2.13	Definition of MPEG Surround 2-1-2 payloads	82
6.2.14	Buffer requirements	84
7	Tool descriptions	85
7.1	Quantization	85
7.1.1	Tool description	85
7.1.2	Definition of elements	85
7.1.3	Decoding process	85
7.2	Noise filling	85
7.2.1	Tool description	85
7.2.2	Definition of elements	86
7.2.3	Decoding process	86
7.2.4	Generation of random signs for spectral noise filling	87
7.3	Scale factors	87
7.4	Spectral noiseless coding	87
7.4.1	Tool description	87
7.4.2	Definition of elements	88
7.4.3	Decoding process	89
7.5	enhanced SBR tool (eSBR)	93
7.5.1	Modifications to SBR tool	93
7.5.2	Additional pre-processing in the MPEG-4 SBR within USAC	108
7.5.3	DFT based harmonic transposer	110
7.5.4	QMF based harmonic transposer	120
7.5.5	4:1 Structure for SBR in USAC	128
7.5.6	Predictive vector coding (PVC) decoding process	138
7.6	Inter-subband-sample temporal envelope shaping (inter-TES)	141
7.6.1	Tool Description	141
7.6.2	Definition of elements	142
7.6.3	Inter-TES	142
7.7	Joint stereo coding	144
7.7.1	M/S stereo	144
7.7.2	Complex stereo prediction	144
7.8	TNS	151
7.8.1	General	151
7.8.2	Definition of elements	151
7.8.3	Decoding process	152
7.8.4	Maximum TNS bandwidth	152
7.9	Filterbank and block switching	153
7.9.1	Tool description	153
7.9.2	Definition of elements	153
7.9.3	Decoding process	153
7.10	Time-warped filterbank and blockswitching	161
7.10.1	Tools description	161
7.10.2	Definition of elements	161
7.10.3	Decoding process	163
7.11	MPEG Surround for mono to stereo upmixing	169
7.11.1	Tool description	169
7.11.2	Decoding process	170
7.12	AVQ decoding	182
7.13	LPC-filter	189
7.13.1	Tool description	189
7.13.2	Definition of elements	189
7.13.3	Number of LPC filters	189
7.13.4	General principle of the inverse quantizer	189
7.13.5	Decoding of the LPC quantization mode	190

7.13.6	First-stage approximation	191
7.13.7	AVQ refinement	191
7.13.8	Reordering of quantized LSFs	193
7.13.9	Conversion into LSP parameters.....	193
7.13.10	Interpolation of LSP parameters.....	194
7.13.11	LSP to LP conversion.....	194
7.13.12	LPC initialization at decoder start-up	195
7.14	ACELP.....	196
7.14.1	General.....	196
7.14.2	Definition of elements.....	196
7.14.3	ACELP initialization at USAC decoder start-up	197
7.14.4	Setting of the ACELP excitation buffer using the past FD synthesis and LPC0.....	197
7.14.5	Decoding of CELP excitation.....	197
7.14.6	Excitation postprocessing.....	203
7.14.7	Synthesis	204
7.14.8	Writing in the output buffer.....	204
7.15	MDCT based TCX.....	205
7.15.1	Tool description	205
7.15.2	Decoding process	205
7.16	Forward aliasing cancellation (FAC) tool.....	209
7.16.1	Tool description	209
7.16.2	Definition of elements.....	209
7.16.3	Decoding process	210
7.16.4	Writing in the output buffer.....	211
7.17	Post-processing of the synthesis signal	212
7.18	Audio pre-roll	214
7.18.1	General.....	214
7.18.2	Semantics	214
7.18.3	Decoding process	215
8	Conformance testing.....	217
8.1	General.....	217
8.2	USAC conformance testing.....	217
8.2.1	Profiles	217
8.2.2	Conformance tools and test procedure.....	218
8.3	USAC bitstreams	222
8.3.1	General.....	222
8.3.2	USAC configuration.....	222
8.3.3	Framework	225
8.3.4	Frequency domain coding (FD mode).....	226
8.3.5	Linear predictive domain coding (LPD mode).....	228
8.3.6	Common core coding tools	229
8.3.7	Enhanced spectral band replication (eSBR)	230
8.3.8	eSBR – Predictive vector coding (PVC).....	232
8.3.9	eSBR – Inter temporal envelope shaping (inter-TES).....	233
8.3.10	MPEG Surround 2-1-2	233
8.3.11	Configuration Extensions.....	235
8.3.12	AudioPreRoll.....	235
8.3.13	DRC	236
8.3.14	Restrictions depending on profiles and levels.....	236
8.4	USAC decoders.....	238
8.4.1	General.....	238
8.4.2	FD core mode tests	238
8.4.3	LPD core mode tests.....	245
8.4.4	Combined core coding tests	250
8.4.5	eSBR tests.....	251
8.4.6	MPEG Surround 212 tests	260
8.4.7	Bitstream extensions.....	262
8.5	Decoder settings.....	265
8.5.1	General.....	265

8.5.2	Target loudness [Lou-<x>]	265
8.5.3	DRC effect type request [Eff-<x>]	265
8.6	Decoding of MPEG-4 file format parameters to support exact time alignment in file-to-file coding	265
9	Reference software	266
9.1	Reference software structure	266
9.1.1	General	266
9.1.2	Copyright disclaimer for software modules	266
9.2	Bitstream decoding software	268
9.2.1	General	268
9.2.2	USAC decoding software	268
Annex A (normative)	Tables	269
Annex B (informative)	Encoder tools	274
Annex C (normative)	Tables for arithmetic decoder	314
Annex D (normative)	Tables for predictive vector coding	320
Annex E (informative)	Adaptive time/frequency post-processing	329
Annex F (informative)	Audio/systems interaction	335
Annex G (informative)	Reference software	337
Annex H (normative)	Carriage of MPEG-D USAC in ISO base media file format	338
Bibliography		339

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.

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) or the IEC list of patent declarations received (see <http://patents.iec.ch>).

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

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

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

This second edition cancels and replaces the first edition (ISO/IEC 23003-3:2012), which has been technically revised. It also incorporates ISO/IEC 23003-3:2012/Cor.1:2012, ISO/IEC 23003-3:2012/Cor.2:2013, ISO/IEC 23003-3:2012/Cor.3:2015, ISO/IEC 23003-3:2012/Cor.4:2015, ISO/IEC 23003-3:2012/Amd.1:2014, ISO/IEC 23003-3:2012/Amd.1:2014/Cor.1:2015, ISO/IEC 23003-3:2012/Amd.2:2015, ISO/IEC 23003-3:2012/Amd.2:2015/Cor.1:2015 and ISO/IEC 23003-3:2012/Amd.3:2016.

A list of all parts in the ISO/IEC 23003 series can be found on the ISO website.

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

Introduction

As mobile appliances become multi-functional, multiple devices converge into a single device. Typically, a wide variety of multimedia content is required to be played on or streamed to these mobile devices, including audio data that consists of a mix of speech and music.

This document specifies unified speech and audio coding (USAC), which allows for coding of speech, audio or any mixture of speech and audio with a consistent audio quality for all sound material over a wide range of bitrates. It supports single and multi-channel coding at high bitrates and provides perceptually transparent quality. At the same time, it enables very efficient coding at very low bitrates while retaining the full audio bandwidth.

Where previous audio codecs had specific strengths in coding either speech or audio content, USAC is able to encode all content equally well, regardless of the content type.

In order to achieve equally good quality for coding audio and speech, the developers of USAC employed the proven MDCT-based transform coding techniques known from MPEG-4 audio and combined them with specialized speech coder elements like ACELP. Parametric coding tools such as MPEG-4 spectral band replication (SBR) and MPEG-D MPEG surround were enhanced and tightly integrated into the codec. The result delivers highly efficient coding and operates down to the lowest bit rates.

The main focus of this codec are applications in the field of typical broadcast scenarios, multimedia download to mobile devices, user-generated content such as podcasts, digital radio, mobile TV, audio books, etc.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of a patent.

ISO and IEC take no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured ISO and IEC that he/she is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO and IEC. Information may be obtained from the patent database available at www.iso.org/patents.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those in the patent database. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Information technology — MPEG audio technologies —

Part 3: Unified speech and audio coding

1 Scope

This document specifies a unified speech and audio codec which is capable of coding signals having an arbitrary mix of speech and audio content. The codec has a performance comparable to, or better than, the best known coding technology that might be tailored specifically to coding of either speech or general audio content. The codec supports single and multi-channel coding at high bitrates and provides perceptually transparent quality. At the same time, it enables very efficient coding at very low bitrates while retaining the full audio bandwidth.

This document incorporates several perceptually-based compression techniques developed in previous MPEG standards: perceptually shaped quantization noise, parametric coding of the upper spectrum region and parametric coding of the stereo sound stage. However, it combines these well-known perceptual techniques with a source coding technique: a model of sound production, specifically that of human speech.

2 Normative references

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

ISO/IEC 14496-3:2019, *Information technology — Coding of audio-visual objects — Part 3: Audio*

ISO/IEC 14496-26:2010, *Information technology — Coding of audio-visual objects — Part 26: Audio conformance*

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

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

3 Terms, definitions, symbols and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 14496-3, ISO/IEC 23003-1 and the following apply.

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

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <http://www.electropedia.org/>

3.1.1

algebraic codebook

fixed codebook where an algebraic code is used to populate the excitation vectors (innovation vectors)

Note 1 to entry: The excitation contains a small number of nonzero pulses with predefined interlaced sets of potential positions. The amplitudes and positions of the pulses of the k^{th} excitation codevector can be derived from its index k through a rule requiring no or minimal physical storage, in contrast with stochastic codebooks whereby the path from the index to the associated codevector involves look-up tables.

3.1.2

algebraic vector quantizer

AVQ

process associating, to an input block of 8 coefficients, the nearest neighbour from an 8-dimensional lattice and a set of binary indices to represent the selected lattice point

Note 1 to entry: This definition describes the encoder. At the decoder, AVQ describes the process to obtain, from the received set of binary indices, the 8-dimensional lattice point that was selected at the encoder.

3.1.3

closed-loop pitch

result of the adaptive codebook search, a process of estimating the pitch (lag) value from the weighted input speech and the long-term filter state

Note 1 to entry: In the closed-loop search, the lag is searched using error minimization loop (analysis-by-synthesis). In USAC, closed-loop pitch search is performed for every subframe.

3.1.4

fractional pitch

set of pitch lag values having sub-sample resolution

Note 1 to entry: In the LPD USAC, a sub-sample resolution of $1/4^{\text{th}}$ or $1/2^{\text{nd}}$ of a sample is used.

3.1.5

zero-input response

ZIR

output of a filter due to past inputs, i.e., due to the present state of the filter, given that an input of zeros is applied

3.1.6

immediate play-out frame

IPF

audio frame that contains an extension payload of type ID_EXT_ELE_AUDIOPREROLL

Note 1 to entry: The extension payload shall contain a Config() element as defined in 7.18, i.e., configLen > 0. It should also contain an adequate number of audio pre-roll frames, i.e., numPreRollFrames > 0, for pre-rolling the audio decoder.

3.1.7

independently decodable frame

IF

audio frame in which the bitstream element usacIndependencyFlag has a value of 1

3.1.8

bitstream

encoded audio data

3.1.9

conformance data

conformance test sequences and conformance tools

3.1.10

conformance tool

tool to check certain conformance criteria

3.1.11**conformance test sequence**

conformance test bitstreams and corresponding reference waveforms

3.1.12**conformance test bitstream**

USAC bitstream used for testing the conformance of a USAC decoder

3.1.13**conformance test condition**

condition which applies to properties of a conformance test bitstream in order to test a certain functionality of the USAC decoder

3.1.14**conformance test case**

combination of one or more conformance test conditions for which a set of conformance test sequences is provided

3.1.15**main audio channel**

audio channel conveyed by means of a `UsacSingleChannelElement` or `UsacChannelPairElement`

3.1.16**reference waveform**

decoded counterpart of a bitstream

3.1.17**USAC bitstream**

data encoded according to this document

3.2 Symbols and abbreviated terms

For the purposes of this document, the symbols and abbreviated terms given in ISO/IEC 14496-3 and the following apply.

ACELP algebraic code-excited linear predictor

PVC predictive vector coding

uclbf unary code, left bit first

NOTE "left bit first" refers to the order in which the unary codes are received. The value is encoded using a conventional unary code, where any decimal value d is represented by d '1' bits followed by one '0' stop-bit.

USAC unified speech and audio coding

UsacCPE `UsacChannelPairElement`

UsacEXT `UsacExtElement`

UsacLFE `UsacLfeElement`

UsacSCE `UsacSingleChannelElement`

$v[] = \{a\}$ This expression indicates that all elements of the array v shall be set to the value a .

4 Technical overview

4.1 Decoder block diagram

The block diagram of the USAC decoder as shown in Figure 1 reflects the general structure of MPEG-D USAC which can be described as follows (from bottom to top): There is a common pre/postprocessing stage consisting of an MPEG Surround functional unit to handle stereo processing (MPS212) and an enhanced SBR (eSBR) unit which handles the parametric representation of the higher audio frequencies in the input signal. Then there are two branches, one consisting of a modified advanced audio coding (AAC) tool path (frequency domain, "FD") and the other consisting of a linear prediction coding (LP or LPC domain, "LPD") based path. The latter can use either a frequency domain representation or a time domain representation of the LPC residual. All transmitted spectra for both FD and LPD path are represented in MDCT domain. The quantized spectral coefficients are coded using a context adaptive arithmetic coder. The time domain representation uses an ACELP excitation coding scheme.

In case of transmitted spectral information, the decoder shall reconstruct the quantized spectra, process the reconstructed spectra through whatever tools are active in the bitstream payload in order to arrive at the actual signal spectra as described by the input bitstream payload, and finally convert the frequency domain spectra to the time domain. Following the initial reconstruction and scaling of the spectrum, there are optional tools that modify one or more of the spectra in order to provide more efficient coding.

In case of transmitted time domain signal representation, the decoder shall reconstruct the quantized time signal, process the reconstructed time signal through whatever tools are active in the bitstream payload in order to arrive at the actual time domain signal as described by the input bitstream payload.

For each of the optional tools that operate on the signal data, the option to "pass through" is retained, and in all cases where the processing is omitted, the spectra or time samples at its input are passed directly through the tool without modification.

In places where the bitstream changes its signal representation from time domain to frequency domain representation or from LP domain to non-LP domain or vice versa, the decoder shall facilitate the transition from one domain to the other by means of an appropriate transition mechanism.

eSBR and MPS212 processing is applied in the same manner to both coding paths after transition handling.

The USAC specification offers in some instances multiple decoding options that serve to provide different quality/complexity trade-offs.

NOTE An informative overview of encoder tools is given by Annex B.

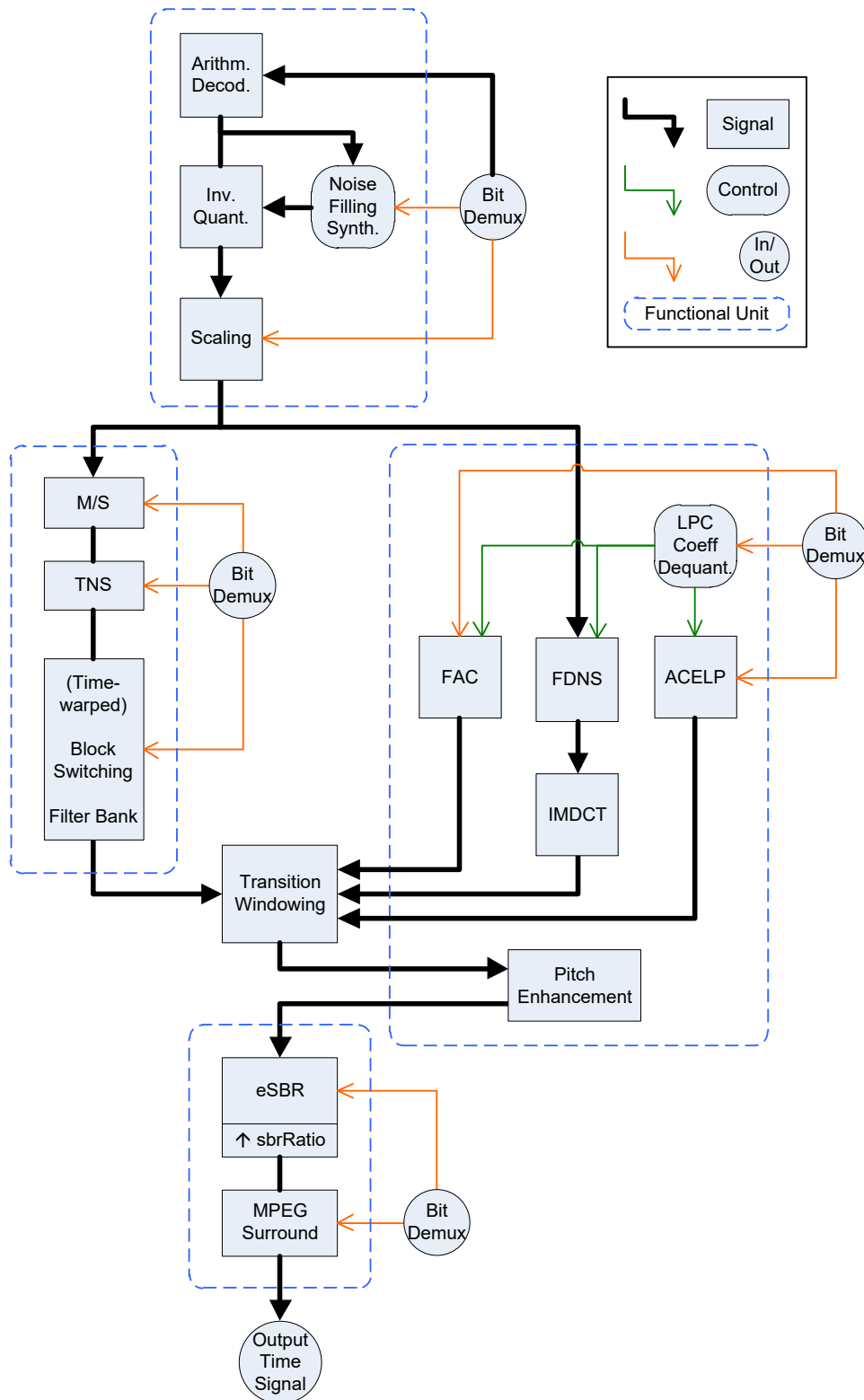


Figure 1 — Simplified block diagram of the typical USAC decoder configuration

4.2 Overview of the decoder tools

The input to the bitstream payload demultiplexer tool is the MPEG-D USAC bitstream payload. The demultiplexer separates the bitstream payload into the parts for each tool, and provides each of the tools with the bitstream payload information related to that tool.

The outputs from the bitstream payload demultiplexer tool are:

- depending on the core coding type in the current frame, either:
 - the quantized and noiselessly coded spectra represented by:
 - scalefactor information;
 - arithmetically coded spectral lines;
 - or: linear prediction (LP) parameters together with an excitation signal represented by either:
 - quantized and arithmetically coded spectral lines (transform coded excitation, TCX) or;
 - ACELP coded time domain excitation;
- the spectral noise filling information (optional);
- the M/S decision information (optional);
- the temporal noise shaping (TNS) information (optional);
- the filterbank control information;
- the time unwarping (TW) control information (optional);
- the enhanced spectral bandwidth replication (eSBR) control information (optional);
- the MPEG Surround 2-1-2 (MPS212) control information (optional).

The scalefactor noiseless decoding tool takes information from the bitstream payload demultiplexer, parses that information, and decodes the Huffman and DPCM coded scalefactors.

The input to the scalefactor noiseless decoding tool is:

- the scalefactor information for the noiselessly coded spectra.

The output of the scalefactor noiseless decoding tool is:

- the decoded integer representation of the scalefactors.

The context adaptive arithmetic decoding tool performs the spectral noiseless decoding step. It takes information from the bitstream payload demultiplexer, parses that information, decodes the context adaptive arithmetically coded data, and reconstructs the quantized spectra.

The input to this noiseless decoding tool is:

- the noiselessly coded spectra.

The output of this noiseless decoding tool is:

- the quantized values of the spectra.

The inverse quantizer tool takes the quantized values for the spectra, and converts the integer values to the non-scaled, reconstructed spectra. This quantizer is a companding quantizer, whose companding factor depends on the chosen core coding mode.

The input to the inverse quantizer tool is:

- the quantized values for the spectra.

The output of the inverse quantizer tool is:

- the un-scaled, inversely quantized spectra.

The noise filling tool is used to fill spectral gaps in the decoded spectra, which occur when spectral value are quantized to zero, e.g., due to a strong restriction on bit demand in the encoder.

The inputs to the noise filling tool are:

- the un-scaled, inversely quantized spectra;
- noise filling parameters;
- the decoded integer representation of the scalefactors.

The outputs to the noise filling tool are:

- the un-scaled, inversely quantized spectral values for spectral lines which were previously quantized to zero;
- modified integer representation of the scalefactors.

The rescaling tool converts the integer representation of the scalefactors to the actual values, and multiplies the un-scaled inversely quantized spectra by the relevant scalefactors.

The inputs to the scalefactors tool are:

- the decoded integer representation of the scalefactors;
- the un-scaled, inversely quantized spectra.

The output from the scalefactors tool is:

- the scaled, inversely quantized spectra.

For an overview over the M/S tool, refer to ISO/IEC 14496-3:2019, 4.1.1.2.

For an overview over the temporal noise shaping (TNS) tool, refer to ISO/IEC 14496-3:2019, 4.1.1.2.

The filterbank/block switching tool applies the inverse of the frequency mapping that was carried out in the encoder. An inverse modified discrete cosine transform (IMDCT) is used for the filterbank tool. The IMDCT can be configured to support 96, 128, 192, 256, 384, 512, 768, or 1024 spectral coefficients.

The inputs to the filterbank tool are:

- the (inversely quantized) spectra;
- the filterbank control information.

The output(s) from the filterbank tool is (are):

- the time domain reconstructed audio signal(s).

The time-warped filterbank/block switching tool replaces the normal filterbank/block switching tool when the time warping mode is enabled. The filterbank is the same (IMDCT) as for the normal filterbank, but in addition the windowed time domain samples are mapped from the warped time domain to the linear time domain by time-varying resampling.

The inputs to the time-warped filterbank tools are:

- the inversely quantized spectra;
- the filterbank control information;
- the time-warping control information.

The output(s) from the filterbank tool is (are):

- the linear time domain reconstructed audio signal(s).

The enhanced SBR (eSBR) tool regenerates the highband of the audio signal. It is based on replication of the sequences of harmonics, truncated during encoding. It adjusts the spectral envelope of the generated high-band and applies inverse filtering, and adds noise and sinusoidal components in order to recreate the spectral characteristics of the original signal.

The input to the eSBR tool is:

- the quantized envelope data;
- control data;
- a time domain signal from the frequency domain core decoder or the ACELP/TCX core decoder.

The output of the eSBR tool is either:

- a time domain signal or;
- a QMF-domain representation of a signal, e.g., in case MPS212 is used.

The MPEG Surround 2-1-2 (MPS212) tool produces multiple signals from one input signal by applying a sophisticated upmix procedure to the input signal controlled by appropriate spatial parameters. In the USAC context MPS212 is used for coding a stereo signal, by transmitting parametric side information alongside a transmitted downmixed signal.

The input to the MPS212 tool is:

- a downmixed time domain signal or;
- a QMF-domain representation of a downmixed signal from the eSBR tool.

The output of the MPS212 tool is:

- a two-channel time domain signal.

The ACELP tool provides a way to efficiently represent a time domain excitation signal by combining a long term predictor (adaptive codebook codeword) with a pulse-like sequence (innovation codebook codeword). The reconstructed excitation is sent through an LP synthesis filter to form a time domain signal.

The input to the ACELP tool is:

- adaptive and innovation codebook indices;

- adaptive and innovation codes gain values;
- other control data;
- inversely quantized and interpolated LPC filter coefficients.

The output of the ACELP tool is:

- the time domain reconstructed audio signal.

The MDCT based TCX decoding tool is used to turn the weighted LP residual representation from an MDCT-domain back to the time domain and outputs a time domain signal in which weighted LP synthesis filtering has been applied. The IMDCT can be configured to support 256, 512, or 1024 spectral coefficients.

The input to the TCX tool is:

- the (inversely quantized) MDCT spectra;
- inversely quantized and interpolated LPC filter coefficients.

The output of the TCX tool is:

- the time domain reconstructed audio signal.

4.3 Combination of USAC with MPEG Surround and SAOC

The output of the USAC decoder can be further processed by MPEG Surround (MPS) (ISO/IEC 23003-1) or spatial audio object coding (SAOC) (ISO/IEC 23003-2). If the SBR tool in USAC is active, a USAC decoder can typically be efficiently combined with a subsequent MPS/SAOC decoder by connecting them in the QMF domain in the same way as it is described for HE-AAC in ISO/IEC 23003-1:2007, 4.4. If a connection in the QMF domain is not possible, they need to be connected in the time domain.

If MPS/SAOC side information is embedded into a USAC bitstream by means of the `usacExtElement` mechanism (with `usacExtElementType` being `ID_EXT_ELE_MPEGS` or `ID_EXT_ELE_SAOC`), the time-alignment between the USAC data and the MPS/SAOC data assumes the most efficient connection between the USAC decoder and the MPS/SAOC decoder. If the SBR tool in USAC is active and if MPS/SAOC employs a 64 band QMF domain representation (see ISO/IEC 23003-1:2007, 6.6.3), the most efficient connection is in the QMF domain. Otherwise, the most efficient connection is in the time domain. This corresponds to the time-alignment for the combination of HE-AAC and MPS as defined in ISO/IEC 23003-1:2007, 4.4, 4.5, and 7.2.1.

The additional delay introduced by adding MPS decoding after USAC decoding is given by ISO/IEC 23003-1:2007, 4.5 and depends on whether HQ MPS or LP MPS is used, and whether MPS is connected to USAC in the QMF domain or in the time domain.

4.4 Interface between USAC and systems

This subclause clarifies the interface between USAC and MPEG systems. Every access unit delivered to the audio decoder from the systems interface shall result in a corresponding composition unit delivered from the audio decoder to the systems interface, i.e., the compositor. This shall include start-up and shut-down conditions, i.e., when the access unit is the first or the last in a finite sequence of access units.

For an audio composition unit, the ISO/IEC 14496-1:2010, 7.1.3.5 composition time stamp (CTS) specifies that the composition time applies to the n -th audio sample within the composition unit. For USAC, the value of n is always 1. Note that this applies to the output of the USAC decoder itself. In the case that a USAC decoder is, for example, being combined with an MPS decoder as described in 4.3, the additional delay caused by the MPS decoding process (see 4.3 and ISO/IEC 23003-1:2007, 4.5) needs to be taken into account for the composition units delivered at the output of the MPS decoder.

4.4.1 Decoder behaviour

4.4.1.1 General decoding process

The decoder shall operate in such a way that the decoding of one access unit shall always and immediately produce one full composition unit of audio signal data (one audio frame with outputFrameLength number of samples).

The decoder shall not discard any audio samples. In particular the decoder shall make no assumptions about encoder delay and shall also not attempt to compensate assumed encoder processing delay by removing audio samples from the composition unit buffer.

Discarding of audio samples due to the presence of an EditListBox as described in Annex F is not part of the normative USAC decoder but shall be applied by the MPEG-4 systems infrastructure.

4.4.1.2 Initialization and re-initialization of the USAC decoder

Upon (re-)initialization, all decoder internal signal buffers shall be set to zero.

Due to the initialized state of the decoder internal buffers, the decoder output may contain "start-up samples" when decoding the first access units of a given compressed data stream.

These start-up samples are samples that do not have a direct relation to the audio input data and are typically zero-valued and may be discarded by the systems infrastructure.

The number of start-up samples to be discarded may for example be transmitted by means of the media_time field in the EditListbox in an ISO base media file format environment. Note that this shall be done by the encoder.

If a given USAC decoder implementation produces more than the minimum number of start-up samples (i.e., it creates additional decoder delay), the number of additional samples shall be reported by the decoder to the systems infrastructure. Systems infrastructure shall then correctly apply delay compensation or time-alignment.

4.4.1.3 Decoding process of access unit with audio pre-roll

The decoding process of access units with embedded audio pre-roll frames is identical to the above description.

The presence of audio pre-roll in the first access unit prepares the decoder internal signal buffers. This allows an encoder to produce a compressed data stream, that will cause the decoder output buffer to contain less or no start-up samples.

The decoding description when changing from one configuration to another while employing audio pre-roll is described in 7.18.3.3.

If a given decoder implementation produces additional start-up samples (additional decoder delay), then the flushing of the old configuration (FlushDecoder()) shall be increased by the same amount of samples. The signal crossfade shall be delayed accordingly. The decoder shall ensure that the number of additional start-up samples (additional decoder delay) does not change when switching to another stream in the adaptation set.

4.5 USAC profiles and levels

4.5.1 General

This subclause defines profiles and their levels for unified speech and audio coding.

Complexity units are defined to give an approximation of the decoder complexity in terms of processing power and RAM usage required for the decoding process. The approximated processing power is given in "processor complexity units" (PCU), specified in MOPS. The approximated RAM usage is given in "RAM complexity units" (RCU), specified in kWords (1000 words).

4.5.2 MPEG-4 HE AACv2 compatibility

Large parts of the USAC codec are inherited from the codec tools and structure subsumed in the MPEG-4 HE AAC v2 profile. A few of these tools have been adopted into USAC as is. Many more have been adopted into USAC and greatly enhanced in terms of performance, capability and flexibility. Others were substituted with tools which provide a range of advantages over their MPEG-4 counterparts. As a result, USAC retains all *functionalities and performance features* that the AAC family of technologies – AAC, HE AAC, HE AAC v2 – can provide. However, it does not adopt all *tools*.

If a decoder is intended to provide full AAC family functionality, including the legacy MPEG-4 AAC tools, all coding tools listed in Table 1 shall be considered.

The following tools listed in Table 1 are normatively referenced in USAC:

Table 1 — Summary of the location of and normative reference to the definitions of all AAC, HE-AAC and USAC coding tools as employed in the extended high efficiency AAC profile

Module	Tool	defined in ISO/IEC	subclause	USAC	AAC LC	SBR	PS
block switching	block switching	14496-3	4.6.11	X	X		
window shapes	AAC based	14496-3	4.6.11	X	X		
	additional USAC	23003-3	6.2.9.3	X			
filter bank	AAC based	14496-3	4.6.11	X	X		
	additional USAC	23003-3	7.9	X			
TNS	TNS	14496-3	4.6.9	X	X		
intensity/ coupling	intensity	14496-3	4.6.8.2	^a	X		
	coupling	14496-3	4.6.8.3		X		
perceptual noise synthesis	PNS	14496-3	4.6.13	^b	X		
	noise filling	23003-3	7.2	X			
MS	basic mid/side coding	14496-3	4.6.8.1	X	X		
	MDCT-based complex prediction	23003-3	7.7.2	X			
quantization	non-uniform	14496-3	4.6.1	X	X		
	uniform	23003-3	7.1	X			
spectral noiseless coding	Huffman	14496-3	4.6.3	^c	X		
	context adaptive arithmetic coding	23003-3	7.4	X			
SBR	base	14496-3	4.6.18	X		X	X
	enhanced	23003-3	7.5	X			
parametric stereo extension	parametric stereo	14496-3	8.6.4/8.A	^d			X
	MPEG Surround 2-1-2 (incl. residual coding)	23003-3	6.2.13	X			
ACELP	ACELP	23003-3	7.14	X			
frequency domain noise shaping	scale factor based	14496-3	4.6.2	X	X		
	LPC based (as part of MDCT based TCX)	23003-3	7.15	X			
^a Functionality of the AAC LC intensity tool is fully provided by the MDCT-based complex prediction tool of USAC. ^b Functionality of the PNS tool is largely provided by the noise filling tool of USAC. ^c Functionality of the AAC LC Huffman coding tool is fully provided by the context adaptive arithmetic coding tool of USAC. ^d Functionality of the parametric stereo tool is fully provided by the MPEG Surround 2-1-2 tool of USAC.							

4.5.3 Baseline USAC profile

In the baseline USAC profile the following coding tools shall not be employed:

- time warped filterbank;
- DFT based harmonic transposer in enhanced spectral band replication;
- fractional delay decorrelator in MPEG Surround for mono to stereo upmixing (MPS212).

Four different hierarchical levels are defined with increasing number of audio channels and increasing complexity. The definition of the four levels of the baseline USAC profile is given in Table 2.

Table 2 — Levels for the baseline USAC profile

Level	Max. channels	Max. sampling rate [kHz]	Max. PCU	Max. RCU
1	1	48	7	6
2	2	48	12	11
3	5.1	48	31	28
4	5.1	96	62	28

Furthermore, restrictions on the sampling rate apply for the USAC baseline profile. The sampling rate signalled as part of the `UsacConfig()` shall be one out of those listed in Table 3. These sampling rates are chosen such that they can conveniently be resampled to 44100 Hz and 48000 Hz, respectively.

Table 3 — Allowed sampling rates for the baseline USAC profile

Sampling rates [Hz] and <code>usacSamplingFrequencyIndex</code>			
88200	0x01	96000	0x00
70560	n/a	76800	n/a
58800	n/a	64000	0x02
44100	0x04	48000	0x03
35280	n/a	38400	0x12
29400	n/a	32000	0x05
22050	0x07	24000	0x06
17640	n/a	19200	0x17
14700	n/a	16000	0x08
11760	n/a	12800	0x1a
11025	0x0a	12000	0x09
8820	n/a	9600	0x1b
7350	0x0c	8000	0x0b

Furthermore the following requirements apply:

- The number of pre-roll frames, `numPreRollFrames`, in an `AudioPreRoll()` extension payload shall not exceed 3.
- Decoders conforming to the baseline USAC profile shall support the full decoding and correct handling of the `AudioPreRoll()` extension.

NOTE The number of pre-roll frames required for seamless operation of the audio codec can be lower than the above mentioned number. See B.26 for encoder implementation guidelines.

4.5.4 Extended high efficiency AAC profile

The extended HE AAC profile contains the audio object types 42 (USAC), 5 (SBR), 29 (PS) and 2 (AAC LC) as defined in ISO/IEC 14496-3. In order for a decoder to support the extended HE AAC profile it shall implement all modules listed in Table 1. For some tools specific restrictions apply as outlined in the following.

The extended HE AAC profile is compatible with the MPEG-4 high efficiency AAC v2 profile as defined in ISO/IEC 14496-3. It warrants decodability of HE AAC v2 profile compliant bitstreams by extended HE AAC profile decoders.

A number of hierarchical levels are defined with increasing number of audio channels and increasing complexity. All levels shall support all tools required by the baseline USAC profile. Support for additional tools is optional. The definition of the levels of the extended HE AAC profile is given in Table 4. All notes in Table 4 and all restrictions listed in the columns 2, 3, 4, and 5 ("Max. channels/object", "Max. AAC sampling rate, SBR not present [kHz]", "Max. AAC sampling rate, SBR present [kHz]", "Max. SBR sampling rate [kHz] (in/out)") of Table 4 apply only when decoding HE AAC v2 profile compliant bitstreams.

Table 4 — Levels for the extended HE AAC profile

Level ^a	Max. channels/object	Max. AAC sampling rate, SBR not present [kHz]	Max. AAC sampling rate, SBR present [kHz]	Max. SBR sampling rate [kHz] (in/out)	Max. PCU	Max. RCU	Max. PCU HQ/LP SBR ^e	Max. RCU HQ/LP SBR ^e
1	NA	NA	NA	NA	NA	NA	NA	NA
2	2	48	24	24/48	12	11	12	11
3	2	48	24/48 ^c	48/48 ^b	15	11	15	11
4	5	48	24/48 ^d	48/48 ^b	25	28	20	23
5	5	96	48	48/96	49	28	39	23
6	7	48	24/48 ^d	48/48	34	37	27	30
7	7	96	48	48/96	67	37	53	30

NOTE A Level 6 or 7 decoder is not required to decode a Level 5 stream.

^a Level 2, 3, 4, 6 and 7 extended HE AAC profile decoders implement the baseline version of the parametric stereo tool. A level 5 decoder shall not be limited to the baseline version of the parametric stereo tool.

^b For level 3, 4 and 6 decoders, it is mandatory to operate the SBR tool in downsampled mode if the sampling rate of the AAC core is higher than 24kHz. Hence, if the SBR tool operates on a 48kHz signal, the internal sampling rate of the SBR tool will be 96kHz, however, the output signal will be downsampled by the SBR tool to 48kHz.

^c If parametric stereo data are present the maximum AAC sampling rate is 24kHz, if parametric stereo data are not present the maximum AAC sampling rate is 48kHz.

^d For one or two channels the maximum AAC sampling rate, with SBR present, is 48kHz. For more than two channels the maximum AAC sampling rate, with SBR present, is 24kHz.

^e The PCU/RCU number are given for a decoder operating the LP SBR tool whenever applicable.

For the MPEG-4 audio object type 2 (AAC LC), mono or stereo mixdown elements are not permitted.

For MPEG-4 audio object types 2, 5, and 29 the following restrictions apply:

- An extended HE AAC profile decoder shall operate the HQ SBR tool for bitstreams containing parametric stereo data.
- For bitstreams not containing parametric stereo data, the extended HE AAC profile decoder may operate the HQ SBR tool, or the LP SBR tool.

- Only bitstreams consisting of exactly one AAC single channel element may contain parametric stereo data. Bitstreams containing more than one channel in the AAC part shall not contain parametric stereo data.

4.6 Combination of USAC with MPEG-D DRC

The output of the USAC decoder can be further processed by MPEG-D DRC (ISO/IEC 23003-4). If the SBR tool in USAC is active, a USAC decoder can typically be efficiently combined with a subsequent MPEG-D DRC decoder by connecting them in the QMF domain in the same way as it is described in ISO/IEC 23003-4. If a connection in the QMF domain is not possible they shall be connected in the time domain.

The MPEG-D DRC payload shall be embedded into a USAC bitstream by means of the `usacExtElement` mechanism, with `usacExtElementType` of type `ID_EXT_ELE_UNI_DRC`. The loudness metadata shall be embedded by means of the `usacConfigExt` mechanism with `usacConfigExtType` of type `ID_CONFIG_EXT_LOUDNESS_INFO`. The time-alignment between the USAC data and the MPEG-D DRC data assumes the most efficient connection between the USAC decoder and the MPEG-D DRC decoder. If the SBR tool in USAC is active, the most efficient connection is in the QMF domain. Otherwise, the most efficient connection is in the time domain. The DRC tool is operated in regular delay mode and the DRC frame size has the same duration as the USAC frame size. The same holds for the DRC sampling rate, which is synchronized to the USAC sampling rate.

The time resolution of the DRC tool is specified by ΔT_{min} in units of the audio sample interval. It is calculated as specified in ISO/IEC 23003-4. Specific values are provided as examples based on the following formula:

$$\Delta T_{min} = 2^M$$

The applicable exponent M is found by looking up the audio sample rate range that fulfils:

$$f_{s,min} \leq f_s < f_{s,max}$$

Table 5 — Lookup table for the exponent M

$f_{s,min}$ [Hz]	$f_{s,max}$ [Hz]	M
8000	16000	3
16000	32000	4
32000	64000	5
64000	128000	6

Given the codec frame size N_{Codec} (`==outputFrameLength`), the DRC frame size in units of DRC samples at a rate of ΔT_{min} is:

$$N_{DRC} = N_{Codec} 2^{-M}$$

For USAC, MPEG-D DRC offers mandatory decoding capability of up to four DRC subbands using the time-domain DRC filter bank. More DRC subbands can be supported by operating in the QMF-domain. DRC sets that contain more than four DRC subbands shall contain gain sequences that are all aligned with the QMF-domain used for SBR. If the SBR tool in USAC is active, MPEG-D DRC shall always operate in the QMF-domain. The gain sequences are all aligned with the QMF domain in that case.

If no additional filter bank is required for the application of multiband DRC gains, MPEG-D DRC doesn't introduce any additional decoding delay.

The `drcLocation` parameter shall be encoded according to Table 6.

Table 6 — Encoding of drcLocation parameter

drcLocation n	Payload
1	uniDrcConfig() / uniDrcGain() (see ISO/IEC 23003-4)
2	<i>reserved</i>
3	<i>reserved</i>
4	<i>reserved</i>

5 Syntax

5.1 General

The bitstream syntax shall be based on ISO/IEC 14496-3:2019, 4.4.

The USAC bitstream syntax is shown in 5.2.

5.2 Decoder configuration (UsacConfig)

Table 7 — Syntax of UsacConfig()

Syntax	No. of bits	Mnemonic
UsacConfig() {		
usacSamplingFrequencyIndex ;	5	bslbf
if (usacSamplingFrequencyIndex == 0x1f) {		
usacSamplingFrequency ;	24	uimsbf
}		
coreSbrFrameLengthIndex ;	3	uimsbf
channelConfigurationIndex ;	5	uimsbf
if (channelConfigurationIndex == 0) {		
UsacChannelConfig();		
}		
UsacDecoderConfig();		
if (usacConfigExtensionPresent ==1) {	1	uimsbf
UsacConfigExtension();		
}		
}		

Table 8 — Syntax of UsacChannelConfig()

Syntax	No. of bits	Mnemonic
<pre> UsacChannelConfig() { numOutChannels = escapedValue(5,8,16); for (i=0; i<numOutChannels; i++) { bsOutputChannelPos[i]; } } </pre>	5	uimsbf

Table 9 — Syntax of UsacDecoderConfig()

Syntax	No. of bits	Mnemonic
<pre> UsacDecoderConfig() { numElements = escapedValue(4,8,16) + 1; for (elemIdx=0; elemIdx<numElements; ++elemIdx) { usacElementType[elemIdx] switch (usacElementType[elemIdx]) { case: ID_USAC_SCE UsacSingleChannelElementConfig(sbrRatioIndex); break; case: ID_USAC_CPE UsacChannelPairElementConfig(sbrRatioIndex); break; case: ID_USAC_LFE UsacLfeElementConfig(); break; case: ID_USAC_EXT UsacExtElementConfig(); break; } } } </pre>	2	uimsbf
NOTE UsacSingleChannelElementConfig(), UsacChannelPairElementConfig(), UsacLfeElementConfig() and UsacExtElementConfig() signalled at position elemIdx refer to the corresponding elements in UsacFrame() at the respective position elemIdx.		

Table 10 — Syntax of UsacSingleChannelElementConfig()

Syntax	No. of bits	Mnemonic
<pre> UsacSingleChannelElementConfig(sbrRatioIndex) { UsacCoreConfig(); if (sbrRatioIndex > 0) { SbrConfig(); } } </pre>		

Table 11 — Syntax of UsacChannelPairElementConfig()

Syntax	No. of bits	Mnemonic
<pre> UsacChannelPairElementConfig(sbrRatioIndex) { UsacCoreConfig(); if (sbrRatioIndex > 0) { SbrConfig(); stereoConfigIndex; } else { stereoConfigIndex = 0; } if (stereoConfigIndex > 0) { Mps212Config(stereoConfigIndex); } } </pre>	2	uimsbf

Table 12 — Syntax of UsacLfeElementConfig()

Syntax	No. of bits	Mnemonic
<pre> UsacLfeElementConfig() { tw_mdct = 0; noiseFilling = 0; } </pre>		

Table 13 — Syntax of UsacCoreConfig()

Syntax	No. of bits	Mnemonic
UsacCoreConfig() { tw_mdct ; noiseFilling ; }	1 1	bslbf bsblf

Table 14 — Syntax of SbrConfig()

Syntax	No. of bits	Mnemonic
SbrConfig() { harmonicSBR ; bs_interTes ; bs_pvc ; SbrDfltHeader(); }	 1 1 1 	 bsblf bsblf bsblf

Table 15 — Syntax of SbrDfltHeader()

Syntax	No. of bits	Mnemonic
SbrDfltHeader()		
{		
dflt_start_freq;	4	uimsbf
dflt_stop_freq;	4	uimsbf
dflt_header_extra1;	1	uimsbf
dflt_header_extra2;	1	uimsbf
if (dflt_header_extra1 == 1) {		
dflt_freq_scale;	2	uimsbf
dflt_alter_scale;	1	uimsbf
dflt_noise_bands;	2	uimsbf
} else {		
dflt_freq_scale = 2;		
dflt_alter_scale = 1;		
dflt_noise_bands = 2;		
}		
if (dflt_header_extra2 == 1) {		
dflt_limiter_bands;	2	uimsbf
dflt_limiter_gains;	2	uimsbf
dflt_interpol_freq;	1	uimsbf
dflt_smoothing_mode;	1	uimsbf
} else {		
dflt_limiter_bands = 2;		
dflt_limiter_gains = 2;		
dflt_interpol_freq = 1;		
dflt_smoothing_mode = 1;		
}		
}		

Table 16 — Syntax of Mps212Config()

Syntax	No. of bits	Mnemonic
Mps212Config(stereoConfigIndex)		
{		
bsFreqRes;	3	uimsbf
bsFixedGainDMX	3	uimsbf
bsTempShapeConfig;	2	uimsbf
bsDecorrConfig;	2	uimsbf
bsHighRateMode;	1	uimsbf
bsPhaseCoding;	1	uimsbf
bsOttBandsPhasePresent;	1	uimsbf
if (bsOttBandsPhasePresent) {		^a
bsOttBandsPhase;	5	uimsbf
}		
if (bsResidualCoding) {		^b
bsResidualBands;	5	uimsbf
bsOttBandsPhase = max(bsOttBandsPhase,bsResidualBands);		
bsPseudoLr;	1	uimsbf
}		
if (bsTempShapeConfig == 2) {		
bsEnvQuantMode;	1	uimsbf
}		
}		
^a If bsOttBandsPhasePresent==0 bsOttBandsPhase is initialized according to Table 109.		
^b bsResidualCoding depends on stereoConfigIndex according to Table 77.		

Table 17 — Syntax of UsacExtElementConfig()

Syntax	No. of bits	Mnemonic
UsacExtElementConfig()		
{		
usacExtElementType = escapedValue(4,8,16);		
usacExtElementConfigLength = escapedValue(4,8,16);		
usacExtElementDefaultLengthPresent;	1	uimsbf
if (usacExtElementDefaultLengthPresent) {		
usacExtElementDefaultLength = escapedValue(8,16,0) + 1;		
} else {		
usacExtElementDefaultLength = 0;		
}		
usacExtElementPayloadFrag;	1	uimsbf
switch (usacExtElementType) {		
case ID_EXT_ELE_FILL:		
break;		
case ID_EXT_ELE_MPEGS:		
SpatialSpecificConfig();		
break;		
case ID_EXT_ELE_SAOC:		
SaocSpecificConfig();		
break;		

<pre> case ID_EXT_ELE_AUDIOPREROLL: /* No configuration element */ break; case ID_EXT_ELE_UNI_DRC: uniDrcConfig(); break; default: while (usacExtElementConfigLength--) { tmp; } break; } </pre>	a	8	uimbsf
^a The default entry for the usacExtElementType is used for unknown extElementTypes so that legacy decoders can cope with future extensions.			

Table 18 — Syntax of UsacConfigExtension()

Syntax	No. of bits	Mnemonic
<pre> UsacConfigExtension() { 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' */ } break; case ID_CONFIG_EXT_LOUDNESS_INFO: loudnessInfoSet(); break; case ID_CONFIG_EXT_STREAM_ID: streamId(); break; default: while (usacConfigExtLength[confExtIdx]--) { tmp; } break; } } } </pre>	8	uimbsf
	8	uimbsf

Table 19 — Syntax of escapedValue()

Syntax	No. of bits	Mnemonic
escapedValue(nBits1, nBits2, nBits3) { value; if (value == $2^{nBits1-1}$) { value += valueAdd; if (valueAdd == $2^{nBits2-1}$) { value += valueAdd; } } return value; }	nBits1 nBits2 nBits3	uimsbf uimsbf uimsbf

Table 20 — Syntax of streamId()

Syntax	No. of bits	Mnemonic
streamId() { streamIdentifier }	16	uimsbf

5.3 USAC bitstream payloads

5.3.1 Payloads for audio object type USAC

Table 21 — Syntax of UsacFrame(),
top level payload for audio object type USAC

Syntax	No. of bits	Mnemonic
UsacFrame() { usacIndependencyFlag; for (elemIdx=0; elemIdx<numElements; ++elemIdx) { switch (usacElementType[elemIdx]) { case: ID_USAC_SCE UsacSingleChannelElement(usacIndependencyFlag); break; case: ID_USAC_CPE UsacChannelPairElement(usacIndependencyFlag); break; case: ID_USAC_LFE UsacLfeElement(usacIndependencyFlag); break; case: ID_USAC_EXT UsacExtElement(usacIndependencyFlag); break; } } }	1	uimsbf

Table 22 — Syntax of UsacSingleChannelElement()

Syntax	No. of bits	Mnemonic
<pre> UsacSingleChannelElement(indepFlag) { UsacCoreCoderData(1, indepFlag); if (sbrRatioIndex > 0) { UsacSbrData(1, indepFlag); } } </pre>		

Table 23 — Syntax of UsacChannelPairElement()

Syntax	No. of bits	Mnemonic
<pre> UsacChannelPairElement(indepFlag) { if (stereoConfigIndex == 1) { nrCoreCoderChannels = 1; } else { nrCoreCoderChannels = 2; } UsacCoreCoderData(nrCoreCoderChannels, indepFlag); if (sbrRatioIndex > 0) { if (stereoConfigIndex == 0 stereoConfigIndex == 3) { nrSbrChannels = 2; } else { nrSbrChannels = 1; } UsacSbrData(nrSbrChannels, indepFlag); } if (stereoConfigIndex > 0) { Mps212Data(indepFlag); } } </pre>		

Table 24 — Syntax of UsacLfeElement()

Syntax	No. of bits	Mnemonic
<pre> UsacLfeElement(indepFlag) { fd_channel_stream(0,0,0,0, indepFlag); } </pre>		

Table 25 — Syntax of UsacExtElement()

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

Table 26 — Syntax of ics_info()

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

5.3.2 Subsidiary payloads

Table 27 — Syntax of UsacCoreCoderData()

Syntax	No. of bits	Mnemonic
<pre> UsacCoreCoderData(nrChannels, indepFlag) { for (ch=0; ch < nrChannels; ch++) { core_mode[ch]; } if (nrChannels == 2) { StereoCoreToolInfo(core_mode, indepFlag, stereoConfigIndex); } 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]; } fd_channel_stream(common_window, common_tw, tns_data_present[ch], noiseFilling, indepFlag); } } } </pre>	1 1	uimsbf a uimsbf b
^a Each channel shall have its own instance of lpd_channel_stream ^b Each channel shall have its own instance of fd_channel_stream		

Table 28 — Syntax of StereoCoreToolInfo()

Syntax	No. of bits	Mnemonic
<pre> StereoCoreToolInfo(core_mode, indepFlag, stereoConfigIndex) { if (core_mode[0] == 0 && core_mode[1] == 0) { tns_active; common_window; if (common_window) { ics_info(); common_max_sfb; if (common_max_sfb == 0) { if (window_sequence == EIGHT_SHORT_SEQUENCE) { max_sfb1; } else { max_sfb1; } } else { max_sfb1 = max_sfb; } } max_sfb_ste = max(max_sfb, max_sfb1); } } </pre>	1 1 1 4 6	uimsbf uimsbf uimsbf uimsbf uimsbf

ms_mask_present;	2	uimsbf
if (ms_mask_present == 1) {		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = 0; sfb < max_sfb_ste; sfb++) {		
ms_used[g][sfb];	1	uimsbf
}		
}		
if ((ms_mask_present == 3) && (stereoConfigIndex == 0)) {		
cplx_pred_data(max_sfb_ste, indepFlag);		
} else {		
alpha_q_re[] = {0};		
alpha_q_im[] = {0};		
}		
if (tw_mdct) {		
common_tw;	1	uimsbf
if (common_tw) {		
tw_data();		
}		
}		
if (tns_active) {		
if (common_window) {		
common_tns;	1	uimsbf
} else {		
common_tns = 0;		
}		
tns_on_lr;	1	uimsbf
if (common_tns) {		
tns_data();		
tns_data_present[0] = 0;		
tns_data_present[1] = 0;		
} else {		
tns_present_both;	1	uimsbf
if (tns_present_both) {		
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 29 — Syntax of `fd_channel_stream()`

Syntax	No. of bits	Mnemonic
<code>fd_channel_stream(common_window, common_tw, tns_data_present, noiseFilling, indepFlag)</code>		
{		
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) {		
if (!common_tw) {		
tw_data();		
}		
}		
scale_factor_data ();		
if (tns_data_present) {		
tns_data ();		
}		
ac_spectral_data(indepFlag);		
fac_data_present;	1	uimsbf
if (fac_data_present) {		
fac_length = (window_sequence==EIGHT_SHORT_SEQUENCE) ? ccfl/16 : ccfl/8;		
fac_data(1, fac_length);		
}		
}		

Table 30 — Syntax of `cplx_pred_data()`

Syntax	No. of bits	Mnemonic
<code>cplx_pred_data(max_sfb_ste, indepFlag)</code>		
{		
cplx_pred_all;	1	uimsbf
if (cplx_pred_all == 0) {		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = 0; sfb < max_sfb_ste; sfb += SFB_PER_PRED_BAND) {		
cplx_pred_used[g][sfb];	1	uimsbf
if ((sfb+1) < max_sfb_ste) {		
cplx_pred_used[g][sfb+1] = cplx_pred_used[g][sfb];		
}		
}		
}		
}		
else {		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = 0; sfb < max_sfb_ste; sfb++) {		
cplx_pred_used[g][sfb] = 1;		
}		
}		
}		

pred_dir;	1	uimsbf
complex_coef;	1	uimsbf
if (complex_coef) {		
if (indepFlag) {		
use_prev_frame = 0;		
} else {		
use_prev_frame;	1	uimsbf
}		
}		
if (indepFlag) {		
delta_code_time = 0;		
} else {		
delta_code_time;	1	uimsbf
}		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = 0; sfb < max_sfb_ste; sfb += SFB_PER_PRED_BAND) {		
if (cplx_pred_used[g][sfb]) {		
hcod_sf[dpcm_alpha_q_re[g][sfb]];	1..19	vlclbf
if (complex_coef) {		
hcod_sf[dpcm_alpha_q_im[g][sfb]];	1..19	vlclbf
}		
} else {		
alpha_q_im[g][sfb] = 0;		
dpcm_alpha_q_im[g][sfb] = 60;		
}		
}		
else {		
alpha_q_re[g][sfb] = 0;		
alpha_q_im[g][sfb] = 0;		
}		
}		
}		

Table 31 — Syntax of tw_data()

Syntax	No. of bits	Mnemonic
tw_data()		
{		
tw_data_present;	1	uimsbf
if (tw_data_present == 1) {		
for (i = 0 ; i < NUM_TW_NODES ; i++) {		
tw_ratio[i];	3	uimsbf
}		
}		
}		

Table 32 — Syntax of scale_factor_data()

Syntax	No. of bits	Mnemonic
<pre> scale_factor_data() { dpcm_sf[0][0] = 60; for (g = 0; g < num_window_groups; g++) { for (sfb = 0; sfb < max_sfb; sfb++) { if (g > 0 sfb > 0) { hcod_sf[dpcm_sf[g][sfb]]; } } } } </pre>	1..19	vlclbf

Table 33 — Syntax of tns_data()

Syntax	No. of bits	Mnemonic
tns_data() { for (w = 0; w < num_windows; w++) { n_filt[w]; if (n_filt[w]) { coef_res[w]; } for (filt = 0; filt < n_filt[w]; filt++) { length[w][filt]; order[w][filt]; if (order[w][filt]) { direction[w][filt]; coef_compress[w][filt]; for (i = 0; i < order[w][filt]; i++) { coef[w][filt][i]; } } } } }	 1..2 1 {4;6} {3;4} 1 1 2..4	 uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf

Table 34 — Syntax of `ac_spectral_data()`

Syntax	No. of bits	Mnemonic
<pre>ac_spectral_data(indepFlag) { if (indepFlag) { arith_reset_flag = 1; } else { arith_reset_flag; } for (win = 0; win < num_windows; win++) { arith_data(lg, arith_reset_flag && (win==0)); } }</pre>	1	uimsbf
^a num_windows indicates the number of windows in the current window_sequence. In case window_sequence is EIGHT_SHORT_SEQUENCE num_windows equals 8. In all other cases num_windows equals 1.		

Table 35 — Syntax of `lpd_channel_stream()`

Syntax	No. of bits	Mnemonic
<pre> lpd_channel_stream(indepFlag) { acelp_core_mode; lpd_mode; bpf_control_info core_mode_last; fac_data_present; first_lpd_flag = !core_mode_last; first_tcx_flag=TRUE; k = 0; if (first_lpd_flag) { last_lpd_mode = -1; } while (k < 4) { if (k==0) { if ((core_mode_last==1) && (fac_data_present==1)) { fac_data(0, ccfl/8); } } else { if ((last_lpd_mode==0 && mod[k]>0) (last_lpd_mode>0 && mod[k]==0)) { fac_data(0, ccfl/8); } } if (mod[k] == 0) { acelp_coding(acelp_core_mode); last_lpd_mode=0; k += 1; } else { tcx_coding(lg(mod[k]) , first_tcx_flag, indepFlag); last_lpd_mode=mod[k]; k += (1 << (mod[k]-1)); first_tcx_flag=FALSE; } } lpc_data(first_lpd_flag); if ((core_mode_last==0) && (fac_data_present==1)) { short_fac_flag; fac_length = short_fac_flag ? ccfl/16 : ccfl/8; fac_data(1, fac_length); } } </pre>	3 5 1 1 1 1 1	uimbsf uimbsf, a uimbsf uimbsf uimbsf b c uimbsf
^a lpd_mode defines the contents of the array <code>mod[]</code> as described in 6.2.10.2, Table 94. ^b <code>first_lpd_flag</code> is defined in 6.2.10.2. ^c The number of spectral coefficients, <code>lg</code>, depends on <code>mod[k]</code> according to Table 155.		

Table 36 — Syntax of `lpc_data()`

Syntax	No. of bits	Mnemonic
<pre> lpc_data(first_lpd_flag, mod[]) { lpc_set = 4; mode_lpc = get_mode_lpc(lpc_set); lpc_first_approximation_index[lpc_set] code_book_indices(lpc_set, nk_mode, 2); if (first_lpd_flag) { lpc_set = 0; mode_lpc = get_mode_lpc(lpc_set); if (mode_lpc == 0) { lpc_first_approximation_index[lpc_set] } code_book_indices(0, nk_mode, 2); } if (mod[0] != 3) { lpc_set = 2; mode_lpc = get_mode_lpc(lpc_set); if (mode_lpc == 0) { lpc_first_approximation_index[lpc_set] } code_book_indices(lpc_set, nk_mode, 2); } if (mod[0] < 2) { lpc_set = 1; mode_lpc = get_mode_lpc(lpc_set); if (mode_lpc == 0) { lpc_first_approximation_index[lpc_set] } if (mode_lpc != 1) { code_book_indices(lpc_set, nk_mode, 2); } } if (mod[2] < 2) { lpc_set = 3; mode_lpc = get_mode_lpc(lpc_set); if (mode_lpc == 0) { lpc_first_approximation_index[lpc_set] } code_book_indices(lpc_set, nk_mode, 2); } } </pre>	8 a 8 a 8 a 8 a 8 a	uimsbf uimsbf uimsbf uimsbf uimsbf
^a nk_mode is determined by the number of the currently decoded LPC Filter set, lpc_set, and the LPC quantization mode, mode_lpc, according to Table 148.		

Table 37 — Syntax of qn_data()

Syntax	No. of bits	Mnemonic
<pre> qn_data(nk_mode, no_qn) { switch (nk_mode) { case 1: for (k=0; k<no_qn; k++) { qn[k] if (qn[k] > 0) { qn[k] += 1 } } break; case 0: case 2: case 3: for (k=0; k<no_qn; k++) { qn_base qn[k] = qn_base + 2; } if (nk_mode == 2) { for (k=0; k<no_qn; k++) { if (qn[k] > 4) { qn[k] if (qn[k] > 0) { qn[k] += 4 } } } } } else { for (k=0; k<no_qn; k++) { if (qn[k] > 4) { qn_ext switch (qn_ext) { case 0: qn[k] = 5; break; case 1: qn[k] = 6; break; case 2: qn[k] = 0; break; default: qn[k] = qn_ext + 4; break; } } } } break; } } } </pre>	1..n 2 1..n 1..n	uclbf uimbsf uclbf uclbf

Table 38 — Syntax of `get_mode_lpc()`

Syntax	No. of bits	Mnemonic
<pre> get_mode_lpc(lpc_set) { switch (lpc_set) { case 4: mode_lpc=0; break; case 0: case 2: mode_lpc = binary_code break; case 1: switch (binary_code) { case '0₂': mode_lpc = 2; break; case '10₂': mode_lpc = 0; break; case '11₂': mode_lpc = 1; break; } break; case 3: switch (binary_code) { case '0₂': mode_lpc = 1; break; case '10₂': mode_lpc = 0; break; case '110₂': mode_lpc = 2; break; case '111₂': mode_lpc = 3; break; } break; } return mode_lpc; } </pre>	1 1..2 1..3	vlclbf vlclbf
NOTE The mapping of binary code to mode_lpc can also be deduced from Table 148.		

Table 39 — Syntax `code_book_indices ()`

Syntax	No. of bits	Mnemonic
<pre> code_book_indices(idx, nk_mode, no_qn) { qn_data(nk_mode, no_qn); for (k=0; k<no_qn; k++) { if (qn[k] > 4) { nk = (qn[k]-3)/2; n = qn[k] - nk*2; } else { nk = 0; n = qn[k]; } code_book_index[idx][k] kv[idx][k][0] kv[idx][k][1] kv[idx][k][2] kv[idx][k][3] kv[idx][k][4] } } </pre>	^a 4*n nk nk nk nk nk nk	uimbsf uimbsf uimbsf uimbsf uimbsf uimbsf

kv[idx][k][5]	nk	uimsbf
kv[idx][k][6]	nk	uimsbf
kv[idx][k][7]	nk	uimsbf
}		
}		

^a idx can take values from 0 to 4 in case the syntax element is used in context of lpc_data(). In case of the use in the context of fac_data() idx can take values from 0 to 7 or from 0 to 15 depending on fac_length.

Table 40 — Syntax of acelp_coding()

Syntax	No. of bits	Mnemonic
acelp_coding(acelp_core_mode)		
{		
mean_energy;	2	uimsbf
nb_subfr = coreCoderFrameLength/256		^a
for (sfr=0; sfr<nb_subfr ; sfr++) {		
if ((sfr==0) ((nb_subfr==4) && (sfr==2))) {		
acb_index[sfr];	9	uimsbf
} else {		
acb_index[sfr];	6	uimsbf
}		
ltp_filtering_flag[sfr];	1	bmsbf
switch (acelp_core_mode) {		
case 0		
icb_index[sfr];	20	uimsbf
break;		
case 1		
icb_index[sfr];	28	uimsbf
break;		
case 2		
icb_index[sfr];	36	uimsbf
break;		
case 3		
icb_index[sfr];	44	uimsbf
break;		
case 4		
icb_index[sfr];	52	uimsbf
break;		
case 5		
icb_index[sfr];	64	uimsbf
break;		
case 6		
icb_index[sfr];	12	uimsbf
break;		
case 7		
icb_index[sfr];	16	uimsbf
break;		
}		
gains[sfr];	7	uimsbf
}		

^a coreCoderFrameLength designates the core frame length in samples and is equal to either 1024 or 768. See also 6.1.1.2.

Table 41 — Syntax of `tcx_coding()`

Syntax	No. of bits	Mnemonic
<code>tcx_coding(lg, first_tcx_flag, indepFlag)</code>		
{		
noise_factor;	3	uimsbf
global_gain;	7	uimsbf
if (first_tcx_flag) {		
if (indepFlag) {		
arith_reset_flag = 1;		
} else {		
arith_reset_flag;	1	uimsbf
}		
}		
else {		
arith_reset_flag=0;		
}		
arith_data(lg, arith_reset_flag);		
}		

Table 42 — Syntax of `arith_data()`

Syntax	No. of bits	Mnemonic
<code>arith_data(lg, arith_reset_flag)</code>		
{		
c = arith_map_context(N, arith_reset_flag);		
for (i=0; i<lg/2; i++) {		
/* MSB decoding */		
c = arith_get_context (c,i,N);		
for (lev=esc_nb=0;;) {		
pki = arith_get_pk(c + (esc_nb<<17));		
acod_m [pki][m];	1..20	vlclbf
if (m != ARITH_ESCAPE)		
break;		
lev += 1;		
if ((esc_nb=lev)>7)		
esc_nb=7;		
}		
b = m>>2;		
a = m - (b<<2);		
/* ARITH_STOP symbol detection */		
if (m==0 && lev>0)		
break;		
/* LSB decoding */		
for (l=lev; l>0; l--) {		
lsbidx = (a==0)?1:((b==0)?0:2);		
acod_r [lsbidx][r];	1..20	vlclbf
a=(a<<1) (r&1);		
b=(b<<1) ((r>>1)&1);		
}		

<pre> x_ac_dec[2*i] = a; x_ac_dec[2*i+1] = b; arith_update_context(i,a,b); } arith_finish(x_ac_dec, i,N,l,g); /* Signs decoding */ for (i=0; i<lg; i++) { if (x_ac_dec[i] != 0) { s; if (s==0) { x_ac_dec[i] *= -1; } } } </pre>	1	uimsbf
--	----------	---------------

Table 43 — Syntax of fac_data()

Syntax	No. of bits	Mnemonic
<pre> fac_data(useGain, fac_length) { if (useGain) { fac_gain; } for (i=0; i<fac_length/8; i++) { code_book_indices (i, 1, 1); } } </pre>	7	uimsbf
NOTE 1 This value is encoded using a modified unary code, where qn=0 is represented by one "0" bit, and any value qn greater or equal to 2 is represented by qn-1 "1" bits followed by one "0" stop bit.		
NOTE 2 qn=1 cannot be signalled, because the codebook Q_1 is not defined.		

5.3.3 Payloads for enhanced SBR

Table 44 — Syntax of UsacSbrData()

Syntax	No. of bits	Mnemonic
<pre> UsacSbrData(numberSbrChannels, indepFlag) { if (indepFlag) { sbrInfoPresent = 1; sbrHeaderPresent = 1; } else { sbrInfoPresent; if (sbrInfoPresent) { sbrHeaderPresent; } else { sbrHeaderPresent = 0; } } if (sbrInfoPresent) { SbrInfo(); } if (sbrHeaderPresent) { </pre>	1	uimsbf
	1	uimsbf

<pre> sbrUseDfltHeader; if (sbrUseDfltHeader) { /* copy all SbrDfltHeader() elements dlft_xxx_yyy to bs_xxx_yyy */ } else { SbrHeader(); } } sbr_data(bs_amp_res, numberSbrChannels, indepFlag); } </pre>	1	uimsbf
---	----------	---------------

Table 45 — Syntax of SbrInfo

Syntax	No. of bits	Mnemonic
SbrInfo()		
{		
bs_amp_res;	1	uimsbf
bs_xover_band;	4	uimsbf
bs_sbr_preprocessing;	1	uimsbf
if (bs_pvc) {		
bs_pvc_mode;	2	uimsbf
}		
}		

Table 46 — Syntax of SbrHeader()

Syntax	No. of bits	Mnemonic
SbrHeader()		
{		
bs_start_freq;	4	uimsbf , ^a
bs_stop_freq;	4	uimsbf , ^a
bs_header_extra_1;	1	uimsbf
bs_header_extra_2;	1	uimsbf
if (bs_header_extra_1) {		^b
bs_freq_scale;	2	uimsbf
bs_alter_scale;	1	uimsbf
bs_noise_bands;	2	uimsbf
}		
if (bs_header_extra_2) {		^b
bs_limiter_bands;	2	uimsbf
bs_limiter_gains;	2	uimsbf
bs_interpol_freq;	1	uimsbf
bs_smoothing_mode;	1	uimsbf
}		
}		
^a bs_start_freq and bs_stop_freq shall define a frequency band that does not exceed the limits defined in 7.5.5 and ISO/IEC 14496-3:2019, 4.6.18.3.6.		
^a If this bit is not set the default values for the underlying data elements shall be used disregarded any previous value.		

Table 47 — Syntax of `sbr_data()`

Syntax	No. of bits	Mnemonic
<pre> sbr_data(bs_amp_res, numberSbrChannels, indepFlag) { switch (numberSbrChannels) { case 1: sbr_single_channel_element(bs_amp_res, bs_pvc_mode, indepFlag); break; case 2: sbr_channel_pair_element(bs_amp_res, indepFlag); break; } } </pre>		

Table 48 — Syntax of `sbr_single_channel_element()`

Syntax	No. of bits	Mnemonic
<pre> sbr_single_channel_element(bs_amp_res, bs_pvc_mode, indepFlag) { if (harmonicSBR) { if (sbrPatchingMode[0] == 0) { sbrOversamplingFlag[0]; if (sbrPitchInBinsFlag[0]) sbrPitchInBins[0]; else sbrPitchInBins[0] = 0; } else { sbrOversamplingFlag[0] = 0; sbrPitchInBins[0] = 0; } } sbr_grid(0, bs_pvc_mode); sbr_dtdf(0, bs_pvc_mode, indepFlag); sbr_invf(0); if (bs_pvc_mode==0) { sbr_envelope(0, 0, bs_amp_res); } else { pvc_envelope(indepFlag); } sbr_noise(0, 0); if (bs_add_harmonic_flag[0]) { sbr_sinusoidal_coding(0, bs_pvc_mode); } } </pre>	1 1 1 7 1	uimsbf uimsbf uimsbf uimsbf uimsbf

Table 49 — Syntax of sbr_channel_pair_element()

Syntax	No. of bits	Mnemonic
sbr_channel_pair_element(bs_amp_res, indepFlag)		
{		
if (bs_coupling==1) {	1	uimsbf
if (harmonicSBR) {		
if (sbrPatchingMode[0,1] == 0) {	1	uimsbf
sbrOversamplingFlag[0,1];	1	uimsbf
if (sbrPitchInBinsFlag[0,1])	1	uimsbf
sbrPitchInBins[0,1];	7	uimsbf
else		
sbrPitchInBins[0,1] = 0;		
} else {		
sbrOversamplingFlag[0,1] = 0;		
sbrPitchInBins[0,1] = 0;		
}		
}		
sbr_grid(0, 0);	a	
if (!bs_coupling) sbr_grid(1, 0);		
sbr_dtdf(0, 0, indepFlag);		
sbr_dtdf(1, 0, indepFlag);		
sbr_invf(0);		
if (!bs_coupling) sbr_invf(1);	a	
sbr_envelope(0,1, bs_amp_res);		
sbr_noise(0,1);		
sbr_envelope(1,1, bs_amp_res);		
sbr_noise(1,1);		
} else {		
if (harmonicSBR) {		
if (sbrPatchingMode[0] == 0) {	1	uimsbf
sbrOversamplingFlag[0];	1	uimsbf
if (sbrPitchInBinsFlag[0])	1	uimsbf
sbrPitchInBins[0];	7	uimsbf
else		
sbrPitchInBins[0] = 0;		
} else {		
sbrOversamplingFlag[0] = 0;		
sbrPitchInBins[0] = 0;		
}		
if (sbrPatchingMode[1] == 0) {	1	uimsbf
sbrOversamplingFlag[1];	1	uimsbf
if (sbrPitchInBinsFlag[1])	1	uimsbf
sbrPitchInBins[1];	7	uimsbf
else		
sbrPitchInBins[1] = 0;		
} else {		
sbrOversamplingFlag[1] = 0;		
sbrPitchInBins[1] = 0;		
}		
}		
sbr_grid(0, 0);		
sbr_grid(1, 0);		
sbr_dtdf(0,0, indepFlag);		

<pre> sbr_dtdf(1,0, indepFlag); sbr_invf(0); sbr_invf(1); sbr_envelope(0,0, bs_amp_res); sbr_envelope(1,0, bs_amp_res); sbr_noise(0,0); sbr_noise(1,0); } if (bs_add_harmonic_flag[0]) { sbr_sinusoidal_coding(0, 0); } if (bs_add_harmonic_flag[1]) { sbr_sinusoidal_coding(1, 0); } } </pre>	1 1	uimbsbf uimbsbf
---	---------------------------------	---

^a In coupling mode only the data for channel 0 is read. This data shall be copied to the same syntax element for channel 1.

Table 50 — Syntax of sbr_grid()

Syntax	No. of bits	Mnemonic
sbr_grid(ch, bs_pvc_mode)		
{		
if (bs_pvc_mode == 0) {		
switch (bs_frame_class [ch]) {	2	uimsbf
case FIXFIX		
bs_num_env[ch] = 2^ tmp ;	2	uimsbf , ^a
bs_freq_res [ch][0];	1	uimsbf
for (env = 1; env < bs_num_env[ch]; env++)		
bs_freq_res[ch][env] = bs_freq_res[ch][0];		
break;		
case FIXVAR		
bs_var_bord_1 [ch];	2	uimsbf
bs_num_env[ch] = bs_num_rel_1 [ch] + 1;	2	uimsbf
for (rel = 0; rel < bs_num_env[ch]-1; rel++)		
bs_rel_bord_1[ch][rel] = 2* tmp + 2;	2	uimsbf
ptr_bits = ceil (log (bs_num_env[ch] + 1) / log (2));		^b
bs_pointer [ch];	ptr_bits	uimsbf
for (env = 0; env < bs_num_env[ch]; env++)		
bs_freq_res [ch][bs_num_env[ch] - 1 - env];	1	uimsbf
break;		
case VARFIX		
bs_var_bord_0 [ch];	2	uimsbf
bs_num_env[ch] = bs_num_rel_0 [ch] + 1;	2	uimsbf
for (rel = 0; rel < bs_num_env[ch]-1; rel++)		
bs_rel_bord_0[ch][rel] = 2* tmp + 2;	2	uimsbf
ptr_bits = ceil (log (bs_num_env[ch] + 1) / log (2));		^b
bs_pointer [ch];	ptr_bits	uimsbf
for (env = 0; env < bs_num_env[ch]; env++)		
bs_freq_res [ch] [env];	1	uimsbf
break;		
case VARVAR		
bs_var_bord_0 [ch];	2	uimsbf

bs_var_bord_1[ch];	2	uimsbf
bs_num_rel_0[ch];	2	uimsbf
bs_num_rel_1[ch];	2	uimsbf
bs_num_env[ch] = bs_num_rel_0[ch] + bs_num_rel_1[ch] + 1;		^a
for (rel = 0; rel < bs_num_rel_0[ch]; rel++)		
bs_rel_bord_0[ch][rel] = 2* tmp + 2;	2	uimsbf
for (rel = 0; rel < bs_num_rel_1[ch]; rel++)		
bs_rel_bord_1[ch][rel] = 2* tmp + 2;	2	uimsbf
ptr_bits = ceil (log(bs_num_env[ch] + 1) / log (2));		^b
bs_pointer[ch];	ptr_bits	uimsbf
for (env = 0; env < bs_num_env[ch]; env++)		
bs_freq_res[ch][env];	1	uimsbf
break;		
}		
if (bs_num_env[ch] > 1) { bs_num_noise[ch] = 2; }		
else { bs_num_noise[ch] = 1; }		
} else {		
bs_noise_position[ch];	4	uimsbf
bs_var_len_hf[ch];	1,3	uimsbf
if (bs_noise_position[ch] == 0) {		
bs_num_env[ch] = 1;		
bs_num_noise[ch] = 1;		
bs_freq_res[ch][0] = 0;		
} else {		
bs_num_env[ch] = 2;		
bs_num_noise[ch] = 2;		
for (env = 0; env < bs_num_env[ch]; env++) {		
bs_freq_res[ch][env] = 0;		
}		
}		
}		
}		

^a bs_num_env is restricted according to 7.5.1.3.

^b The division (/) is a float division without rounding or truncation.

Table 51 — Syntax of sbr_envelope()

Syntax	No. of bits	Mnemonic
<pre> sbr_envelope(ch, bs_coupling, bs_amp_res) { amp_res = bs_amp_res; if (bs_frame_class[ch] == FIXFIX && bs_num_env[ch] == 1) { amp_res = 0; } if (bs_coupling) { if (ch) { t_huff = t_huffman_env_bal_3_0dB; f_huff = f_huffman_env_bal_3_0dB; } else { t_huff = t_huffman_env_bal_1_5dB; f_huff = f_huffman_env_bal_1_5dB; } } } </pre>		

```

    } else {
        if (amp_res) {
            t_huff = t_huffman_env_3_0dB;
            f_huff = f_huffman_env_3_0dB;
        } else {
            t_huff = t_huffman_env_1_5dB;
            f_huff = f_huffman_env_1_5dB;
        }
    }
} else {
    if (amp_res) {
        t_huff = t_huffman_env_3_0dB;
        f_huff = f_huffman_env_3_0dB;
    } else {
        t_huff = t_huffman_env_1_5dB;
        f_huff = f_huffman_env_1_5dB;
    }
}

for (env = 0; env < bs_num_env[ch]; env++) {
    if (bs_df_env[ch][env] == 0) {
        if (bs_coupling && ch) {
            if (amp_res)
                bs_data_env[ch][env][0] = bs_env_start_value_balance;           5           uimsbf
            else
                bs_data_env[ch][env][0] = bs_env_start_value_balance;           6           uimsbf
        } else {
            if (amp_res)
                bs_data_env[ch][env][0] = bs_env_start_value_level;             6           uimsbf
            else
                bs_data_env[ch][env][0] = bs_env_start_value_level;             7           uimsbf
        }
        for (band = 1; band < num_env_bands[bs_freq_res[ch][env]]; band++)
            bs_data_env[ch][env][band] = sbr_huff_dec(f_huff, bs_codeword);    1..18
    } else {
        for (band = 0; band < num_env_bands[bs_freq_res[ch][env]]; band++)
            bs_data_env[ch][env][band] = sbr_huff_dec(t_huff, bs_codeword);    1..18
    }
    if (bs_interTes) {
        bs_temp_shape[ch][env];           1           uimsbf
        If (bs_temp_shape[ch][env]) {
            bs_inter_temp_shape_mode[ch][env];           2           uimsbf
        }
    }
}
}

```

^a num_env_bands[bs_freq_res[ch][env]] is derived from the header according to ISO/IEC 14496-3:2019, 4.6.18.3 and is named **n**.

^b sbr_huff_dec() is defined in ISO/IEC 14496-3:2019, 4.A.6.1.

Table 52 — Syntax of sbr_dtdf()

Syntax	No. of bits	Mnemonic
<pre> sbr_dtdf (ch, bs_pvc_mode, indepFlag) { if (bs_pvc_mode == 0) { if (indepFlag) { bs_df_env[ch][0] = 0 } else { bs_df_env[ch][0]; } for (env = 1; env < bs_num_env[ch]; env++) { bs_df_env[ch][env]; } } if (indepFlag) { bs_df_noise[ch][0] = 0 } else { bs_df_noise[ch][0]; } for (noise = 1; noise < bs_num_noise[ch]; noise++) { bs_df_noise[ch][noise]; } } </pre>	1 1 1 1	

Table 53 — Syntax of sbr_sinusoidal_coding()

Syntax	No. of bits	Mnemonic
<pre> sbr_sinusoidal_coding(ch, bs_pvc_mode) { for (n = 0; n < num_high_res[ch]; n++) bs_add_harmonic [ch][n]; if (bs_pvc_mode != 0) { bs_sinusoidal_position = 31; bs_sinusoidal_position_flag; if (bs_sinusoidal_position_flag == 1) bs_sinusoidal_position; } } </pre>	1 1 5	uimsbf

Table 54 — Syntax of pvc_envelope

Syntax	No. of bits	Mnemonic
pvc_envelope(indepFlag)		
{		
divMode;	3	uimsbf
nsMode;	1	uimsbf
if (divMode<=3) {		
num_length = divMode;		
if (indepFlag) {		
reuse_pvcID = 0;		
} else {		
reuse_pvcID;	1	uimsbf
}		
if (reuse_pvcID) {		
pvcID[0]=pvcID[-1];		
} else {		
pvcID[0];	7	uimsbf
}		
k=1;		
if (num_length) {		
sum_length=0;		
for (i=0; i<num_length; i++) {		
if (sum_length >= 13) {		
length;	1	uimsbf
} else if (sum_length >= 11) {		
length;	2	uimsbf
} else if (sum_length >= 7) {		
length;	3	uimsbf
} else {		
length;	4	uimsbf
}		
length += 1;		
sum_length += length;		
for (j=1; j<length; j++, k++) {		
pvcID[k] = pvcID[k-1];		
}		
pvcID[k++];	7	uimsbf
}		
}		
for (; k<16; k++) {		
pvcID[k]=pvcID[k-1];		
}		
} else {		
switch (divMode) {		
case 4:		
num_grid_info=2;		
fixed_length=8;		
break;		
case 5:		
num_grid_info=4;		
fixed_length=4;		
break;		