
**Information technology — Coding of
audio-visual objects —**

**Part 3:
Audio**

**AMENDMENT 2: Parametric coding
for high-quality audio**

Technologies de l'information — Codage des objets audiovisuels —

Partie 3: Codage audio

*AMENDEMENT 2: Codage paramétrique pour le codage audio de
haute qualité*



Reference number
ISO/IEC 14496-3:2001/Amd.2:2004(E)

AVA Documents
Document Sharing Hub
© ISO/IEC 2004

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

© ISO/IEC 2004

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

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.

Amendment 2 to ISO/IEC 14496-3:2001 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Introduction

This document specifies the second Amendment to ISO/IEC 14496-3:2001. The document specifies the normative syntax of the 'Parametric Coding for High Quality Audio' tool SSC and the decoding process. An informative encoder description is given as well.

Information technology — Coding of audio-visual objects —

Part 3: Audio

AMENDMENT 2: Parametric coding for high-quality audio

In ISO/IEC 14496-3:2001, Introduction, add:

MPEG-4 SSC, (SinuSoidal Coding) is a parametric coding tool that is capable of full bandwidth high quality audio coding. The coding tool dissects a monaural or stereo audio signal into a number of different objects that each can be parameterized efficiently and encoded at a low bit-rate. These objects are transients: representing dynamic changes in the temporal domain, sinusoids: representing deterministic components, and noise: representing components that do not have a clear temporal or spectral localisation. The fourth object, that is only relevant for stereo input signals, captures the stereo image. As the signal is represented in a parametric domain, independent, high quality pitch and tempo scaling are possible at low computational cost.

Amendment subpart 1

In Part 3: Audio, Subpart 1, in subclause 1.3 Terms and Definitions, add:

270. **SSC**: SinuSoidal Coding.

and increase the index-number of subsequent entries.

In Part 3: Audio, Subpart 1, in subclause 1.5.1.1 Audio object type definition, replace table 1.1 with the table below:

Table 1.1 – Audio object definition

Tools/ Modules	Audio Object																																					
	Type	gain control	block switching	window shapes - standard	window shapes – AAC LD	filterbank - standard	filterbank – SSR	TNS	LTP	intensity	coupling	MPEG-2 prediction	PNS	MS	SIAQ	FSS	upsampling filter tool	quantisation&coding - AAC	quantisation&coding - TwinVQ	quantisation&coding - BSAC	AAC ER Tools	ER payload syntax	EP Tool 1)	CELP	Silence Compression	HVXC	HVXC 4kbs VR	SA tools	SASBF	MIDI	HILN	TTSI	SBR	SSC	Remark	Object Type ID		
Null																																					0	
AAC main			X	X		X		X		X	X	X	X	X					X																	2)	1	
AAC LC			X	X		X		X		X	X		X	X					X																		2	
AAC SSR	X	X	X	X			X	X		X	X		X	X					X																		3	
AAC LTP			X	X		X		X	X	X	X		X	X					X																	2)	4	
SBR																																		X			5	
AAC Scalable			X	X		X		X	X	X			X	X	X	X	X	X																		6)	6	
TwinVQ			X	X		X		X	X				X							X																	7	
CELP																								X													8	
HVXC																										X											9	
(Reserved)																																					10	
(Reserved)																																					11	
TTSI																																X					12	
Main synthetic																												X	X	X							3)	13
Wavetable synthesis																												X	X	X							4)	14
General MIDI																														X							15	
Algorithmic Synthesis and Audio FX																												X									16	
ER AAC LC			X	X		X		X		X			X	X				X			X	X	X														17	
(Reserved)																																					18	
ER AAC LTP			X	X		X		X	X	X			X	X				X			X	X	X													5)	19	
ER AAC scalable			X	X		X		X		X			X	X	X	X	X	X			X	X	X													6)	20	
ER TwinVQ			X	X		X		X						X					X			X	X														21	
ER BSAC			X	X		X		X		X			X	X						X		X	X														22	
ER AAC LD					X	X		X	X	X			X	X				X			X	X	X														23	
ER CELP																						X	X	X	X												24	
ER HVXC																						X	X			X	X										25	
ER HILN																						X	X								X						26	
ER Parametric																						X	X			X	X				X						27	
SSC																																			X		28	
(Reserved)																																					29	
(Reserved)																																					30	
(Reserved)																																					31	

In Part 3: Audio, Subpart 1, replace Table 1.2 (Audio Profiles definition) with the following table:

Table 1.2 – Audio Profiles definition

Audio Object Type	Main Audio Profile	Scalable Audio Profile	Speech Audio Profile	Synthetic Audio Profile	High Quality Audio Profile	Low Delay Audio Profile	Natural Audio Profile	Mobile Audio Internet-working Profile	AAC Profile	High Efficiency AAC Profile	Object Type ID
Null											0
AAC main	X						X				1
AAC LC	X	X			X		X		X	X	2
AAC SSR	X						X				3
AAC LTP	X	X			X		X				4
SBR										X	5
AAC Scalable	X	X			X		X				6
TwinVQ	X	X					X				7
CELP	X	X	X		X	X	X				8
HVXC	X	X	X			X	X				9
(reserved)											10
(reserved)											11
TTSI	X	X	X	X		X	X				12
Main synthetic	X			X							13
Wavetable synthesis											14
General MIDI											15
Algorithmic Synthesis and Audio FX											16
ER AAC LC					X		X	X			17
(reserved)											18
ER AAC LTP					X		X				19
ER AAC Scalable					X		X	X			20
ER TwinVQ							X	X			21
ER BSAC							X	X			22
ER AAC LD						X	X	X			23
ER CELP					X	X	X				24
ER HVXC						X	X				25
ER HILN							X				26
ER Parametric							X				27
SSC											28
(reserved)											29
(reserved)											30
(reserved)											31

In Part 3: Audio, Subpart 1, in subclause 1.6.2.1 *AudioSpecificConfig*, replace table 1.8 with the table below:

Table 1.8 – Syntax of *AudioSpecificConfig*()

Syntax	No. of bits	Mnemonic
<i>AudioSpecificConfig</i> ()		
{		
audioObjectType;	5	uimsbf
samplingFrequencyIndex;	4	uimsbf
if (<i>samplingFrequencyIndex</i> ==0xf)		
samplingFrequency;	24	uimsbf
channelConfiguration;	4	uimsbf
<i>sbrPresentFlag</i> = -1;		
if (<i>audioObjectType</i> == 5) {		
<i>extensionAudioObjectType</i> = <i>audioObjectType</i> ;		
<i>sbrPresentFlag</i> = 1;		
extensionSamplingFrequencyIndex;	4	uimsbf
if (<i>extensionSamplingFrequencyIndex</i> ==0xf)		
extensionSamplingFrequency;	24	uimsbf
audioObjectType;	5	uimsbf
}		
else {		
<i>extensionAudioObjectType</i> = 0;		
}		
if (<i>audioObjectType</i> == 1 <i>audioObjectType</i> == 2		
<i>audioObjectType</i> == 3 <i>audioObjectType</i> == 4		
<i>audioObjectType</i> == 6 <i>audioObjectType</i> == 7)		
<i> GASpecificConfig</i> ();		
if (<i>audioObjectType</i> == 8)		
<i> CelpSpecificConfig</i> ();		
if (<i>audioObjectType</i> == 9)		
<i> HvxcSpecificConfig</i> ();		
if (<i>audioObjectType</i> == 12)		
<i> TTSSpecificConfig</i> ();		
if (<i>audioObjectType</i> == 13 <i>audioObjectType</i> == 14		
<i>audioObjectType</i> == 15 <i>audioObjectType</i> ==16)		
<i> StructuredAudioSpecificConfig</i> ();		
if (<i>audioObjectType</i> == 17 <i>audioObjectType</i> == 19		
<i>audioObjectType</i> == 20 <i>audioObjectType</i> == 21		
<i>audioObjectType</i> == 22 <i>audioObjectType</i> == 23)		
<i> GASpecificConfig</i> ();		
if (<i>audioObjectType</i> == 24)		
<i> ErrorResilientCelpSpecificConfig</i> ();		
if (<i>audioObjectType</i> == 25)		
<i> ErrorResilientHvxcSpecificConfig</i> ();		
if (<i>audioObjectType</i> == 26 <i>audioObjectType</i> == 27)		
<i> ParametricSpecificConfig</i> ();		
if (<i>audioObjectType</i> == 17 <i>audioObjectType</i> == 19		
<i>audioObjectType</i> == 20 <i>audioObjectType</i> == 21		
<i>audioObjectType</i> == 22 <i>audioObjectType</i> == 23		
<i>audioObjectType</i> == 24 <i>audioObjectType</i> == 25		
<i>audioObjectType</i> == 26 <i>audioObjectType</i> == 27) {		
epConfig;	2	uimsbf
if (<i>epConfig</i> == 2 <i>epConfig</i> == 3) {		
<i> ErrorProtectionSpecificConfig</i> ();		
}		
if (<i>epConfig</i> == 3) {		

directMapping;	1	uimsbf
if (! directMapping) {		
/* tbd */		
}		
}		
}		
if (audioObjectType == 28)		
SSCSpecificConfig();		
if (extensionAudioObjectType != 5 &&		
bits_to_decode() >= 16) {		
syncExtensionType;	11	bslbf
if (syncExtensionType == 0x2b7) {		
extensionAudioObjectType;	5	uimsbf
if (extensionAudioObjectType == 5) {		
sbrPresentFlag;	1	uimsbf
If (sbrPresentFlag == 1) {		
extensionSamplingFrequencyIndex;	4	uimsbf
if (extensionSamplingFrequencyIndex == 0xf)		
extensionSamplingFrequency;	24	uimsbf
}		
}		
}		
}		
}		
}		

In Part 3: Audio, Subpart 1, in subclause 1.6.2.2.1 Overview, replace table 1.9 by the following table:

Table 1.9 – Audio Object Types

Audio Object Type	Object Type ID	definition of elementary stream payloads and detailed syntax	Mapping of audio payloads to access units and elementary streams
AAC MAIN	1	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
AAC LC	2	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
AAC SSR	3	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
AAC LTP	4	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
SBR	5	ISO/IEC 14496-3 subpart 4	
AAC scalable	6	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.3
TwinVQ	7	ISO/IEC 14496-3 subpart 4	
CELP	8	ISO/IEC 14496-3 subpart 3	
HVXC	9	ISO/IEC 14496-3 subpart 2	
TTSI	12	ISO/IEC 14496-3 subpart 6	
Main synthetic	13	ISO/IEC 14496-3 subpart 5	
Wavetable synthesis	14	ISO/IEC 14496-3 subpart 5	
General MIDI	15	ISO/IEC 14496-3 subpart 5	
Algorithmic Synthesis and Audio FX	16	ISO/IEC 14496-3 subpart 5	
ER AAC LC	17	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER AAC LTP	19	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER AAC scalable	20	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER Twin VQ	21	ISO/IEC 14496-3 subpart 4	
ER BSAC	22	ISO/IEC 14496-3 subpart 4	
ER AAC LD	23	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER CELP	24	ISO/IEC 14496-3 subpart 3	
ER HVXC	25	ISO/IEC 14496-3 subpart 2	
ER HILN	26	ISO/IEC 14496-3 subpart 7	
ER Parametric	27	ISO/IEC 14496-3 subpart 2 and 7	
SSC	28	ISO/IEC 14496-3 subpart 8	

Create Part 3: Audio, Subpart 8:

Subpart 8: Technical description of parametric coding for high quality audio

8.1 Scope

This part of ISO/IEC 14496 describes the MPEG-4 audio parametric coding scheme for compression of high quality audio. The short name is SSC (SinuSoidal Coding). At bit-rates around 24 kbit/s stereo and at a sampling rate of 44.1 kHz, the SSC coding scheme offers a quality that is interesting for a number of applications.

SSC employs four different tools that together parameterize an audio signal. These tools consist of transient modelling, sinusoidal modelling, noise modelling and stereo image modelling. One of the distinctive features of SSC is that it provides decoder support for independent tempo and pitch scaling at hardly any additional complexity.

Transient tool

The transient tool captures the highly dynamic events of the audio input signal. These events are efficiently modelled by means of a limited number of sinusoids that are shaped by means of an envelope.

Sinusoidal tool

The sinusoidal tool captures the deterministic events of the audio input signal. The slowly varying nature of sinusoidal components for typical audio signals is exploited by linking sinusoids over consecutive frames. By means of differential coding, the frequency, amplitude and phase parameters can be efficiently represented.

Noise tool

The noise tool captures the stochastic or non-deterministic events of the audio input signal. In the decoder, a white noise generator is used as excitation. A temporal and spectral envelope is applied to control the temporal and spectral properties of the noise in the audio signal.

Parametric stereo coding tool

The parametric stereo coding tool is able to capture the stereo image of the audio input signal into a limited number of parameters, requiring only a small overhead ranging from a few kbit/s for medium quality, up to about 9 kbit/s for higher quality. Together with a monaural downmix of the stereo input signal generated by the parametric stereo coding tool, the parametric stereo decoding tool is able to regenerate the stereo signal. It is a generic tool that in principle can operate in combination with any monaural coder. In Annex A of this document a normative description of the combination of HE-AAC with the parametric stereo coding tool is provided. SSC can also operate in dual mono mode. In that case the parametric stereo coding tool is not employed. The parametric stereo tool is intended for low bit-rates.

8.2 Terms and definitions

8.2.1

Frame

Basic unit that can be decoded on itself (file header information is required for general decoder settings).

8.2.2

Laguerre filter

Filter structure used in the noise analysis and synthesis.

8.2.3

Audio frame

Contains all data to decode an SSC-coded frame as a stand-alone unit (file header information is required for general decoder settings). For audio frames with `refresh_sinusoids==%1` and `refresh_noise==%1` the complete frame can always be reconstructed; otherwise it is possible in the case of random access that parts of the signal cannot be reconstructed (e.g. sinusoidal continuations, noise).

8.2.4

Sub-frame

Fine granularity within a frame.

8.2.5

f_s

The sampling frequency in Hertz.

8.2.6

Segment

An interval of samples that can be synthesized on the basis of the parameters that correspond to a sub-frame. The segment size is $2 \cdot S$ (see Table 8.11).

8.2.7

Window

A function that is used to weigh synthesized samples within a segment such that a valid synthesis is obtained.

8.2.8

LSF

Line Spectral Frequency.

8.2.9

Overlap and add

An additive method of combining overlapping intervals during signal synthesis.

8.2.10

Linking process

A method to keep track of sinusoidal components over time.

8.2.11

birth

The first component of a sinusoidal track.

8.2.12

Continuation

A sinusoidal track component that is not at the start or the end of a track.

8.2.13

Death

The last component of a sinusoidal track.

8.2.14

SMR

Signal-to-masking ratio.

8.2.15

Partial

Sinusoid of a limited duration.

8.2.16

IID

Inter-channel Intensity Differences.

8.2.17**IPD**

Inter-channel Phase Differences.

8.2.18**OPD**

Overall Phase Differences.

8.2.19**ICC**

Inter-channel Coherence.

8.3 Symbols and abbreviations**8.3.1 Arithmetic operators** $\lfloor x \rfloor$ Round x towards minus infinity $\lceil x \rceil$ Round x towards plus infinity.mod Modulus operator: $\text{mod}(x, y) = x - \left\lfloor \frac{x}{y} \right\rfloor y$. Defined only for positive values of x and y. $\Gamma(\alpha)$ Gamma distribution function, defined as $\Gamma(\alpha) = \int_0^{\infty} e^{-t} \cdot t^{\alpha-1} dt$.**8.3.2 Relation operators** $x?y:z$ If x is true then y else z.**8.3.3 Mnemonics**

The following mnemonics are defined to describe the different data types used in the coded bit-stream.

uimsbf Unsigned integer, most significant bit first.**simsbf** Signed integer, most significant bit first.**bslbf** Bitstream left bit first.**8.3.4 Ranges** $[0, 10]$ A number in the range of 0 up to and including 10. $[0, 10>$ A number in the range of 0 up to but excluding 10.**8.3.5 Number notation**

%X Binary number representation (e.g. %01111100).

\$X Hexadecimal number representation (e.g. \$7C).

X Numbers with no prefix use decimal representation (e.g. 124).

8.3.6 Definitions

S	Number of samples in a sub-frame (see Table 8.11).
L	Number of samples in a segment; $L = 2 \cdot S$.
<i>numQMFSlots</i>	Number of QMF subband samples per <i>ps_data()</i> element. For SSC, this parameter is fixed to 24.

8.4 Payloads for the audio object type SSC

8.4.1 Decoder configuration (SSCSpecificConfig)

Table 8.1 – Syntax of SSCSpecificConfig()

Syntax	Num. bits	Mnemonic
SSCSpecificConfig (channelConfiguration)		
{		
decoder_level	2	uimsbf
update_rate	4	uimsbf
synthesis_method	2	uimsbf
if (channelConfiguration != 1)		
{		
mode_ext	2	uimsbf
if ((channelConfiguration == 2) && (mode_ext == 1))		
{		
reserved	2	uimsbf
}		
}		
}		

8.4.2 SSC Bitstream Payload

Table 8.2 – Syntax of ssc_audio_frame()

Syntax	Num. bits	Mnemonic
ssc_audio_frame ()		
{		
ssc_audio_frame_header()		
ssc_audio_frame_data()		
}		

Table 8.3 – Syntax of ssc_audio_frame_header()

Syntax	Num. bits	Mnemonic
<pre> ssc_audio_frame_header () { refresh_sinusoids refresh_sinusoids_next_frame refresh_noise for (ch = 0; ch < nrof_channels; ch++) { s_nrof_continuations[0][ch] } n_nrof_den n_nrof_lsf freq_granularity amp_granularity phase_jitter_present if (phase_jitter_present == 1) { phase_jitter_percentage phase_jitter_band } } </pre>	1 1 1 Note 1 5 Note 1 2 2 1 2 2	uimbsf uimbsf uimbsf uimbsf uimbsf uimbsf uimbsf uimbsf uimbsf uimbsf
Note 1: See description of s_nrof_continuations and n_nrof_lsf in section 8.5.2.		

Table 8.4 – Syntax of ssc_audio_frame_data()

Syntax	Num. bits	Mnemonic
<pre> ssc_audio_frame_data() { for (sf = 0; sf < nrof_subframes; sf++) { for (ch = 0; ch < nrof_channels; ch++) { ssc_mono_subframe(sf,ch) if ((channelConfiguration == 2) && (mode_ext == 1) && (mod(sf+1,4)==0)) { ps_data() } } } } </pre>		

Table 8.5 – Syntax of ssc_mono_subframe()

Syntax	Num. bits	Mnemonic
<pre> ssc_mono_subframe (sf,ch) { subframe_transients(sf, ch) subframe_sinusoids(sf, ch) subframe_noise(sf, ch) } </pre>		

Table 8.6 – Syntax of subframe_transients()

Syntax	Num. bits	Mnemonic
<pre> subframe_transients (sf, ch) { t_transient_present[sf][ch] if (t_transient_present[sf][ch] == 1) { t_loc[sf][ch] if (t_type[sf][ch]==1) { t_b_par[sf][ch] t_chi_par[sf][ch] t_nrof_sin[sf][ch] t_nrof_sin[sf][ch]++ for (i = 0; i < t_nrof_sin[sf][ch]; i++) { t_freq[sf][ch][i] t_amp[sf][ch][i] t_phi[sf][ch][i] } } } } </pre>	1 Note 1 2 3 3 3 9 5 5	uimbsf uimbsf uimbsf uimbsf uimbsf uimbsf uimbsf uimbsf simsbf

Note 1: See description of t_loc in section 8.5.2.

Table 8.7 – Syntax of subframe_sinusoids()

Syntax	Num. bits	Mnemonic
<pre> subframe_sinusoids(sf, ch) { n = 0; p=0; q=0; /* Continuations */ if (sf > 0) { noc = 0; while (tmp_cont[ch][noc] > 0) { noc++;} s_nrof_continuations[sf][ch] = noc; } if ((refresh_sinusoids == 1) && (sf == 0)) { for (i = 0; i < s_nrof_continuations[sf][ch]; i++, n++) { s_cont[sf][ch][n] = ssc_huff_dec(huff_scont,bs_codeword); s_freq_coarse[sf][ch][n] = ssc_huff_dec(huff_sfreqc,bs_codeword); s_freq_fine[sf][ch][n] s_amp_coarse[sf][ch][n] = ssc_huff_dec(huff_sampca,bs_codeword); s_amp_fine[sf][ch][n] s_phi[sf][ch][n] if (s_cont[sf][ch][n] > 0) { s_adpcm_grid[sf][ch][n] = ssc_huff_dec(huff_sgrid,bs_codeword); s_delta_cont_freq_pha[sf+1][ch][p] p++; } if (s_cont[sf][ch][n] > 1) { s_delta_cont_freq_pha[sf+2][ch][q] q++; } } } else { for (i = 0; i < s_nrof_continuations[sf][ch]; i++, n++) { if (sf == 0) { s_cont[sf][ch][n] = ssc_huff_dec(huff_scont,bs_codeword); } else { s_cont[sf][ch][n] = tmp_cont[ch][n] - 1; } if (s_cont[sf][ch][n] > 0) { p++; } if (s_cont[sf][ch][n] > 1) { if ((refresh_sinusoids_next_frame == 0) (nrof_subframes-sf > 2)) { s_delta_cont_freq_pha[sf+2][ch][q] } q++; } s_delta_cont_amp[sf][ch][n] = ssc_huff_dec(huff_sampcr[amp_granularity],bs_codeword); } } } </pre>	2..5 7..25 0..3 3..16 0..3 5 3..7 2 2 2..5 2 1..15	bslbf bslbf simsbf bslbf simsbf simsbf bslbf uimsbf uimsbf bslbf uimsbf bslbf

<pre> } } /* Births */ s_nrof_births[sf][ch] = ssc_huff_dec(huff_nrofbirths,bs_codeword); if (s_nrof_births[sf][ch] > 0) { s_cont[sf][ch][n] = ssc_huff_dec(huff_scont,bs_codeword); s_freq_coarse[sf][ch][n] = ssc_huff_dec(huff_sfreqba,bs_codeword); s_freq_fine[sf][ch][n] s_amp_coarse[sf][ch][n] = ssc_huff_dec(huff_sampba,bs_codeword); s_amp_fine[sf][ch][n] s_phi[sf][ch][n] if (s_cont[sf][ch][n] > 0) { if ((refresh_sinusoids_next_frame == 0) (nrof_subframes-sf > 1)) { s_delta_cont_freq_pha[sf+1][ch][p] } p++; } if (s_cont[sf][ch][n] > 1) { if ((refresh_sinusoids_next_frame == 0) (nrof_subframes-sf > 2)) { s_delta_cont_freq_pha[sf+2][ch][q] } q++; } n++; for (i = 1; i < s_nrof_births[sf][ch]; i++, n++) { s_cont[sf][ch][n] = ssc_huff_dec(huff_scont,bs_codeword); s_delta_birth_freq_coarse[sf][ch][n] = ssc_huff_dec(huff_sfreqbr,bs_codeword); s_delta_birth_freq_fine[sf][ch][n] s_delta_birth_amp_coarse[sf][ch][n] = ssc_huff_dec(huff_sampbr,bs_codeword); s_delta_birth_amp_fine[sf][ch][n] s_phi[sf][ch][n] if (s_cont[sf][ch][n] > 0) { if ((refresh_sinusoids_next_frame == 0) (nrof_subframes-sf > 1)) { s_delta_cont_freq_pha[sf+1][ch][p] } p++; } if (s_cont[sf][ch][n] > 1) { if ((refresh_sinusoids_next_frame == 0) (nrof_subframes-sf > 2)) { s_delta_cont_freq_pha[sf+2][ch][q] } q++; } } } } /* Keep track of sinusoids that continue in next sub-frame(s) */ for (i = 0, k = 0; i < n; i++) { </pre>	<pre> 3..15 2..5 7..21 0..3 3..15 0..3 5 2 2 2..5 5..23 0..3 2..21 0..3 5 2 2 </pre>	<pre> bslbf bslbf bslbf bslbf bslbf bslbf uimsbf uimsbf bslbf bslbf simsbf bslbf simsbf simsbf uimsbf uimsbf </pre>
--	--	---

<pre> if (s_cont[sf][ch][i] > 0) { tmp_cont[ch][k] = s_cont[sf][ch][i]; k++; } }</pre>		
Note: The variables p, q are used as position indices for subframe+1 and subframe+2 respectively.		

Table 8.8 – Syntax of subframe_noise()

Syntax	Num. bits	Mnemonic
<pre> subframe_noise (sf, ch) { if ((refresh_noise == 1) && (sf == 0)) { n_laguerre[ch] n_laguerre_granularity[sf][ch] for (i = 0; i < n_nrof_den; i++) { n_lar_den_coarse[sf][ch][i] = ssc_huff_dec(huff_nlag,bs_codeword); if (n_laguerre_granularity[sf][ch]==1) { n_lar_den_fine[sf][ch][i] } } n_gain[sf][ch] n_lsf[sf][ch][0] = ssc_huff_dec(huff_nlsf,bs_codeword); for (i = 1; i < n_nrof_lsf; i++) { n_delta_lsf[sf][ch][i] = ssc_huff_dec(huff_nlsf,bs_codeword); } } else { if (mod(sf,2) == 0) { n_laguerre_granularity[sf][ch] for (i = 0; i < n_nrof_den; i++) { n_delta_lar_den_coarse[sf][ch][i] = ssc_huff_dec(huff_nlag,bs_codeword); if(n_laguerre_granularity[sf][ch]==1) { n_delta_lar_den_fine[sf][ch][i] } } } if (mod(sf,4) == 0) { n_delta_gain[sf][ch] = ssc_huff_dec(huff_ngain,bs_codeword); if (n_overlap_lsf == 1) { for (i = n_nrof_overlap_lsf; i < n_nrof_lsf; i++) { n_delta_lsf[sf][ch][i] = ssc_huff_dec(huff_nlsf,bs_codeword); } } else { n_lsf[sf][ch][0] = ssc_huff_dec(huff_nlsf,bs_codeword); for (i = 1; i < n_nrof_lsf; i++) { n_delta_lsf[sf][ch][i] = ssc_huff_dec(huff_nlsf,bs_codeword); } } } } } </pre>	2 1 1..18 2 7 2..9 2..9 1 1..18 2 1..12 1 2..9 2..9	uimbsbf uimbsbf bslbf simsbf uimbsbf bslbf bslbf uimbsbf bslbf simsbf bslbf uimbsbf bslbf bslbf

Table 8.9 – Syntax of ps_data()

Syntax	Num. bits	Mnemonic
ps_data()		
{		
if (enable_ps_header) {	1	uimsbf
if (enable_iid) {	1	uimsbf
iid_mode	3	uimsbf
nr_iid_par = nr_iid_par_tab[iid_mode]		
nr_ipdopd_par = nr_ipdopd_par_tab[iid_mode]		
}		
if (enable_icc) {	1	uimsbf
icc_mode	3	uimsbf
nr_icc_par = nr_icc_par_tab[icc_mode]		
}		
enable_ext	1	uimsbf
}		
frame_class	1	uimsbf
num_env_idx	2	uimsbf
num_env = num_env_tab[frame_class][num_env_idx]		
if (frame_class) {		
for (e=0 ; e<num_env ; e++) {		
border_position[e]	5	uimsbf
}		
}		
for (e=0 ; e<num_env ; e++) {		
if (enable_iid) {		
iid_dt[e]	1	uimsbf
iid_data()		
}		
}		
for (e=0 ; e<num_env ; e++) {		
if (enable_icc) {		
icc_dt[e]	1	uimsbf
icc_data()		
}		
}		
if (enable_ext) {		
cnt = ps_extension_size	4	uimsbf
if (cnt == 15)		
cnt += esc_count	8	uimsbf
num_bits_left = 8 * cnt		
while (num_bits_left > 7) {		
ps_extension_id	2	uimsbf
num_bits_left -= 2		
ps_extension(ps_extension_id, num_bits_left)		
}		
fill_bits	num_bits_left	
}		
}		
ps_extension(ps_extension_id, num_bits_left){		
if (ps_extension_id == 0) {		
if (enable_ipdopd) {	1	uimsbf

<pre> opd_par_df[e][b] = ssc_huff_dec(huff_opd_df,bs_codeword); } } } </pre>	1...5	bslbf
Note 1: ipd_bits and opd_bits represent the number of bits read by ipd_data() and opd_data() respectively. Note 2: the index iid_quant into huff_iid_df is obtained from Table 8.19.		

8.5 Semantics

8.5.1 SSCSpecificConfig

channelConfiguration – Channel configuration as defined in ISO/IEC-14496-3:edition 2001 subpart 1, paragraph 1.6.3.4.

decoder_level – Complexity bounds for decoder settings. A decoder that supports a certain level of complexity is not able to decode a bit-stream that is encoded according a higher level of complexity. This decoder is however able to decode a bit-stream that is encoded according to a lower level of complexity (see Table 8.10).

Table 8.10 - Decoder level

decoder_level	Level of complexity	max_nrof_sinusoids	max_nrof_den	#bits for s_nrof_continuations	#bits for n_nrof_lsf
00	Reserved	Na	Na	Na	Na
01	Medium	60	22	6	4
10	Reserved	Na	Na	Na	Na
11	Reserved	Na	Na	Na	Na

max_nrof_sinusoids - Maximum number of sinusoids that is allowed (sinusoids under Meixner transients not included).

max_nrof_den - Maximum value for n_nrof_den.

update_rate – Four bits indicating the sub-frame size S. Table 8.11 shows the relationship between update_rate and the sub-frame size S in samples.

Table 8.11 - Update rate

update_rate	S	update_rate	S
0000	Reserved	1000	Reserved
0001	Reserved	1001	Reserved
0010	Reserved	1010	Reserved
0011	Reserved	1011	Reserved
0100	384	1100	Reserved
0101	Reserved	1101	Reserved
0110	Reserved	1110	Reserved
0111	Reserved	1111	Reserved

synthesis_method – Two bits providing information on the preferred synthesis for the specific encoded program (see Table 8.12).

Table 8.12 - Synthesis method

Synthesis_method	Optimal synthesis
00	Overlap and Add
01	Reserved
10	Reserved
11	Reserved

mode_ext – In combination with channelConfiguration the mode_ext bits provide the full channel configuration. The number of bits is dependent on the channelConfiguration (see Table 8.13).

Table 8.13 - Channel configuration

channelConfiguration	# bits for mode_ext	nrof_channels
1	0	1
2	2	According to mode_ext
0, 3 ... 15	Na	Na

For channelConfiguration == 2, Table 8.14 applies:

Table 8.14 – Channel configuration in case channelConfiguration == 2

mode_ext	Full channel configuration
00	Dual mono (ch0=left, ch1=right)
01	Parametric Stereo
10	Reserved
11	Reserved

reserved – Two reserved bits; should be set to %0.

8.5.2 Decoding of SSC Bitstream Payload

ssc_audio_frame() – syntactic element that contains a single SSC frame

ssc_audio_frame_header() – syntactic element that contains the header data for a single SSC frame

ssc_audio_frame_data() – syntactic element that contains the data for a single SSC frame

ssc_huff_dec() – Huffman decoding procedure. See Annex B.

refresh_sinusoids – One bit indicating how sinusoidal continuations of the first sub-frame in a frame are encoded. If this bit equals %0, the continued track data is differentially coded with respect to the last sub-frame of the previous frame. If this bit equals %1, the continued track data in the first sub-frame of a frame are coded as absolute values.

refresh_sinusoids_next_frame – One bit providing an additional frame look ahead for the ADPCM decoding of sinusoidal parameters. If this bit is set to %1, the next frame is a refresh frame. In that case the bit refresh_sinusoids shall be set to %1 in the next frame.

refresh_noise – One bit indicating how noise parameters of the first sub-frame in a frame are encoded. If this bit equals %0, the noise parameters are differentially coded with respect to the last sub-frame of the previous frame. If this bit equals %1, the noise parameters in the first sub-frame of a frame are coded as absolute values.

s_nrof_continuations[sf][ch] – For sub-frame sf and channel ch, this value represents the number of continuations. In the case sf==0 the value of s_nrof_continuations is provided in the bit-stream. For the

remaining values of *sf*, the value of *s_nrof_continuations* is obtained implicitly as described in section 8.6.2.1. The number of bits required for *s_nrof_continuations*[0][*ch*] depends on the maximum number of allowed sinusoids, which is dependent of the decoder complexity, indicated by *decoder_level*. This relation is shown in Table 8.10.

n_nrof_den – Number of denominator LAR coefficients of the FIR filter for noise generation.

n_nrof_lsf – Number of LSF coefficients used for the generation of envelope for noise generation. The number of bits required for *n_nrof_lsf* depends on the decoder complexity, indicated by *decoder_level*. This relation is shown in Table 8.10.

freq_granularity – The granularity of the differentially or absolute coded frequency parameters used in *subframe_sinusoids()*. This parameter determines the number of bits to be read for the fine part of the frequency parameters.

amp_granularity – The granularity of the differentially or absolute coded amplitude parameters used in *subframe_sinusoids()*. This parameter determines the Huffman table to be used or the number of bits to be read for the fine part of the amplitude parameters.

phase_jitter_present – One bit to indicate presence of phase jitter parameters. If this bit equals %0, no phase jitter is present. If this bit equals %1, phase jitter is present.

phase_jitter_percentage – This is a two bit unsigned integer identifying a distance percentage. The full distance equals half a quantisation step. The maximum jitter applied to the frequency components is

$$\max_jitter = 2^{\text{freq_granularity}-1} \frac{\text{phase_jitter_percentage}+1}{2^2}.$$

phase_jitter_band – Two bits identifying the frequency representation level from which phase jitter must be applied. Table 8.15 provides the relation between *phase_jitter_band* and $f_{jitter,min}$.

Table 8.15 - Phase jitter band expressed in representation levels

phase_jitter_band	frequency representation level $f_{jitter,min}$
00	0
01	800
10	1600
11	2400

nrof_subframes – the number of sub-frames in one frame. This value is fixed to 8.

ssc_mono_subframe() – syntactic element that contains the data for a single SSC sub frame.

ps_data() – syntactic element that contains the parametric stereo data.

subframe_transients() – syntactic element that contains the transient data for a single SSC sub frame.

subframe_sinusoids() – syntactic element that contains the sinusoid data for a single SSC sub frame.

subframe_noise() – syntactic element that contains the noise data for a single SSC sub frame.

t_transient_present[sf][ch] – One bit indicating if a transient is present in sub-frame *sf*, channel *ch*. If *t_transient_present[sf][ch]==%1*, a transient is present. If *t_transient_present[sf][ch]==%0*, no transient is present.

t_loc[sf][ch] – Indication of the location of the transient in sub-frame sf of channel ch, indicated in the number of samples from the start of the sub-frame. The valid range for t_loc is [0,S>. The number of bits that is used to represent t_loc is calculated according to

$$\lceil \log_2(S) \rceil,$$

where S represents the sub-frame size in samples.

t_type[sf][ch] – Two bits to indicate the transient type of the transient in sub-frame sf of channel ch (see Table 8.16).

Table 8.16 - Transient types

t_type	Type
00	Step
01	Meixner
10	Reserved
11	Reserved

t_b_par[sf][ch] – For a transient of the Meixner type in sub-frame sf of channel ch, these 3 bits hold the value for the attack of the transient envelope, denoted as the ‘b-parameter’. Allowed values for t_b_par are [0, 1, 2, 3]. The remaining values are reserved. The value b is calculated as

$$b = t_b_par + 2.$$

t_chi_par[sf][ch] – For a transient of the Meixner type in sub-frame sf of channel ch, these 3 bits hold the value for the decay of the transient envelope, denoted as the ‘ξ-parameter’. Allowed values for t_chi_par are [0, 1, 2, 3]. The remaining values are reserved. The value ξ is tabulated in Table 8.17.

Table 8.17 - Quantised values for b and ξ

ξ		t_b_par			
		0	1	2	3
t_chi_par	0	0.9688	0.9685	0.9683	0.9681
	1	0.9763	0.9756	0.9750	0.9744
	2	0.9839	0.9827	0.9817	0.9807
	3	0.9914	0.9898	0.9884	0.9870

t_nrof_sin[sf][ch] – For a transient of the Meixner type in sub-frame sf of channel ch, these 3 bits represent the number of sinusoids that are present under the envelope. The number of sinusoids under the Meixner envelope is equal to the value in the stream plus one.

t_freq[sf][ch][i] – For a transient of the Meixner type in sub-frame sf of channel ch, these bits represent the frequency in radians of the i-th sinusoid under the transient envelope.

$$tf_q[i] = \frac{2\pi}{f_s} \frac{10^{\frac{t_freq[sf][ch][i]}{11.421.4}} - 1}{0.00437},$$

where tf_q represents the dequantized absolute frequency in radians.

t_amp[sf][ch][i] – For a transient of the Meixner type in sub-frame sf of channel ch, these bits represent the amplitude of the i-th sinusoid under the transient envelope.

$$ta_q[i] = ta_b^{2 \cdot t_amp[sf][ch][i]},$$

where ta_b represents the log quantisation base (maximum error=1.5dB), $ta_b = 1.1885$. ta_q represents the dequantized absolute amplitude.

t_phi[sf][ch][i] – For a transient of the Meixner type in sub-frame sf of channel ch, these bits represent the phase of the i-th sinusoid under the transient envelope. The decoded value is converted into a phase value in radians in the range $[-\pi, \pi]$ and is specified for the start of the transient.

$$tp_q[i] = 2 \cdot tp_e \cdot t_phi[sf][ch][i],$$

where tp_e represents the absolute phase error ($tp_e = \frac{\pi}{32}$) and tp_q represents the dequantized absolute phase (in radians). The allowed range for t_phi is $[-16, 15]$; the representation level +16 is represented by -16 (because $+\pi = -\pi$).

noc – Local variable that counts the number of continuations in the previous sub-frame.

tmp_cont[ch][noc] – Local array that contains a copy of the s_cont-parameters of the previous sub-frame, needed for parsing the stream correctly (extract number of continuations and keep track of how many sub-frames the sinusoidal track must be continued in the current frame).

s_cont[sf][ch][n] – For sub-frame sf and channel ch, this value indicates how many sub-frames component n will be continued in the current frame (if the component continues also in the next frame, one must be added to the number of sub-frames it continues in the current frame. If the value is 0 this indicates component n stops at sub-frame sf, which is called a death). The valid range for s_cont is $[0, 9]$.

s_freq_coarse[sf][ch][n] – For sub-frame sf and channel ch, this value represents the coarse frequency parameter of the n-th sinusoid.

s_freq_fine[sf][ch][n] – For sub-frame sf and channel ch, this signed integer represents a higher level of detail to the coarse frequency parameter. The number of bits to be read amounts to $(3 - \text{freq_granularity})$. The frequency representation level f_{rl} is the sum of coarse frequency, fine frequency scaled to the granularity grid.

$$f_{rl}[n] = s_freq_coarse[sf][ch][n] + s_freq_fine[sf][ch][n] \cdot 2^{\text{freq_granularity}}.$$

Phase jitter is only applied in combination with tempo and pitch scaling. If phase_jitter_present == %1 and $f_{rl} > f_{jitter,min}$, the phase jitter parameter is

$$f_{jitter} = \lfloor \max_jitter \cdot (2x - 1) + 0.5 \rfloor,$$

where x holds a random number, uniformly distributed between 0 and 1, generated for each frequency parameter in the sub-frame, matching the requirement above. The decoded value is converted into a dequantized absolute frequency value f_q in radians, using the following equation:

$$f_q[n] = \frac{2\pi}{f_s} \frac{10^{\frac{f_{rl}[n]}{91.2214}} - 1}{0.00437}.$$

s_amp_coarse[sf][ch][n] – For sub-frame sf and channel ch, this value represents the coarse amplitude parameter of the n-th sinusoid.

s_amp_fine[sf][ch][n] – For sub-frame sf and channel ch, this parameter represents a higher level of detail to the coarse amplitude parameter. The number of bits to be read amounts to $(3 - \text{amp_granularity})$. The amplitude representation level sa_{rl} is the sum of coarse amplitude, fine amplitude scaled to the granularity grid

$$sa_{rl}[n] = s_amp_coarse[sf][ch][n] + s_amp_fine[sf][ch][n] \cdot 2^{\text{amp_granularity}}.$$

The decoded value is converted into a dequantized linear amplitude value sa_q in the range $[1, 2^{15}-1]$ conforming to

$$sa_q[n] = sa_b^{2 \cdot sa_{rl}[n]}.$$

Where $sa_b = 1.0218$ is the log quantization base. Its value corresponds to a maximum error of 0.1875 dB.

s_phi[sf][ch][n] – For sub-frame sf and channel ch, this represents the phase parameter of the n-th sinusoid. This value is converted into a phase value in radians in the range $[-\pi, \pi]$ conforming to

$$sp_q[n] = 2 \cdot sp_e \cdot s_phi[sf][ch][n],$$

where sp_e represents the absolute phase error ($sp_e = \frac{\pi}{32}$) and sp_q represents the dequantized absolute phase (in radians) The allowed range for s_phi is $[-16, 15]$; the representation level +16 is represented by -16 (because $+\pi == -\pi$).

s_adpcm_grid[sf][ch][n] – For sub-frame sf and channel ch, this value represents the initial index into Table 8.30 as used in the ADPCM decoder for the n-th sinusoid. This table is used to decode the sinusoidal information.

s_delta_cont_freq_pha[sf][ch][n] – For sub-frame sf and channel ch, this value represent the representation levels for the n-th sinusoid, that serve as input to the ADPCM decoder. In order to compensate for the 2 sub-frames delay in the decoder, the representation levels are transmitted 2 sub-frames in advance. In the bit-stream syntax, future representation levels are indicated by indices sf+1 and sf+2, indicating the representation levels of the two sub-sequent sub-frames respectively. In the case sf+1 or sf+2 exceeds nrof_subframes, then the representation level is assigned to the next frame. In this case, the new number of the sub-frame in the next frame is (sf+1) - nrof_subframes or (sf+2) - nrof_subframes respectively.

s_delta_cont_amp[sf][ch][n] – For sub-frame sf and channel ch, this represents the differential amplitude parameter of the n-th sinusoid. This value is converted into a linear amplitude value in the range $[1, 2^{15}-1]$ conforming to

$$sa_{rl}[n] = sa_{rl,psf} + s_delta_cont_amp[sf][ch][n],$$

where sa_{rl} represents the amplitude representation level and $sa_{rl,psf}$ represents the amplitude representation level in the previous sub-frame. For dequantization of sa_{rl} into sa_q see s_amp_fine. In the case the amplitude granularity, amp_granularity of the current frame is different from that of the previous frame, prior to applying the differential encoded values, the granularity of the value of the previous frame is converted to the granularity of the current frame according to

$$sa_{rl,psf} = 2^{\text{amp_granularity}} \left\lfloor \frac{sa'_{rl,psf}}{2^{\text{amp_granularity}}} + 0.5 \right\rfloor,$$

where $sa'_{rl,psf}$ represents the amplitude representation level of the previous sub-frame and amp_granularity represents the granularity of the current sub-frame.

s_nrof_births[sf][ch] – For sub-frame sf and channel ch, this value represents the number of births. The allowed range is $[0, \text{max_nrof_sinusoids}-s_nrof_continuations[sf][ch]]$.

s_delta_birth_freq_coarse[sf][ch][n] – For sub-frame sf and channel ch, this value represents the differential, coarse frequency parameter of the n-th sinusoid.

s_delta_birth_freq_fine[sf][ch][n] – For sub-frame sf and channel ch, this represents a higher level of detail to the coarse differential frequency parameter. The number of bits to be read is (3 – freq_granularity). The delta frequency representation level df_{rl} is

$$df_{rl}[n] = s_delta_birth_freq_coarse[sf][ch][n] + s_delta_birth_freq_fine[sf][ch][n] \cdot 2^{\text{freq_granularity}}$$

The decoded value of the n-th sinusoid is converted into a frequency value in Hertz, using the frequency representation level of the previous birth f_{rl} of sub-frame sf (the (n-1)th sinusoid):

$$f_{rl}[n] = f_{rl}[n-1] + df_{rl}[n],$$

where f_{rl} represents the frequency representation level. Modification of f_{rl} due to phase jitter uses the same rules as stated under s_freq_fine. For dequantization of f_{rl} into f_q see also s_freq_fine.

s_delta_birth_amp_coarse[sf][ch][n] – For sub-frame sf and channel ch, this represents the differential, coarse amplitude parameter of the n-th sinusoid.

s_delta_birth_amp_fine[sf][ch][n] – For sub-frame sf and channel ch, this represents a higher level of detail to the coarse amplitude parameter. The number of bits to be read is (3 – amp_granularity). The delta amplitude representation level sda_{rl} is

$$sda_{rl}[n] = s_delta_birth_amp_coarse[sf][ch][n] + s_delta_birth_amp_fine[sf][ch][n] \cdot 2^{\text{amp_granularity}}$$

The decoded value for the n-th sinusoid is converted into a linear amplitude value, using the amplitude representation level of the previous birth sa_{rl} (the (n-1)th sinusoid):

$$sa_{rl}[n] = sa_{rl}[n-1] + sda_{rl}[n],$$

where sa_{rl} represents the amplitude representation level. For dequantization of sa_{rl} into sa_q see s_amp_fine.

n_laguerre[ch] – λ coefficient of the Laguerre filter for noise synthesis, see Table 8.18.

Table 8.18 - Possible values for λ

n_laguerre	λ
00	0
01	0.5
10	0.7
11	Reserved

n_laguerre_granularity[sf][ch] – 1 bit denoting quantisation accuracy of the Laguerre coefficients.

n_lar_den_coarse[sf][ch][i] – for sub-frame sf and channel ch this represents the denominator LAR coefficient number i.

n_lar_den_fine[sf][ch][i] – for sub-frame sf and channel ch this represents a higher level of detail to the coarse denominator LAR coefficient parameter. The representation level $nlar_{rl}$ is the sum of coarse denominator LAR and fine denominator LAR:

$$nlar_{rl}[i] = n_lar_den_coarse[sf][ch][i] + n_lar_den_fine[sf][ch][i]$$

and is converted to a LAR coefficient according to:

$$nlar_q[i] = nlar_{rl}[i] \cdot \Delta_{\text{LAR}}.$$

For the definition of Δ_{LAR} see section 8.6.3.3.1. Section 8.6.3.3.2 and 8.6.3.3.3 show how to convert LARs to reflection- or a-parameters.

n_gain[sf][ch] – For sub-frame sf and channel ch this value represents the gain coefficient. The gain representation level $ngain_{rl}$ is obtained as:

$$ngain_{rl} = n_gain[sf][ch]$$

n_lsf[sf][ch][i] – For sub-frame sf and channel ch this value represents the LSF coefficient number i. The allowed range for n_lsf is [0,255]. The dequantised LSF parameters $nlsf_q$ are obtained as

$$nlsf_q[i] = n_lsf[sf][ch][i] * \pi / 256 + \pi / 512$$

n_delta_lsf[sf][ch][i] – For sub-frame sf and channel ch this value represents the differential LSF coefficient number i. They are obtained using the following algorithm

for $i > 0$:

$$nlsf_q[i] = (nlsf_q[i-1] + n_delta_lsf[sf][ch][i]) * \pi / 256 + \pi / 512.$$

n_delta_lar_den_coarse[sf][ch][i] – for sub-frame sf and channel ch this represents the differential, denominator LAR coefficient number i.

n_delta_lar_den_fine[sf][ch][i] – for sub-frame sf and channel ch this represents a higher level of detail to the coarse denominator LAR coefficient parameter. The representation level $ndlar_{rl}$ is the sum of the differential coarse denominator LAR and differential fine denominator LAR:

$$ndlar_{rl}[i] = n_delta_lar_den_coarse[sf][ch][i] + n_delta_lar_den_fine[sf][ch][i]$$

and is converted to a LAR coefficient according to:

$$nlar_{rl}[i] = nlar_{rl,psf}[i] + ndlar_{rl}[i].$$

where $nlar_{rl}[i]$ and $nlar_{rl,psf}[i]$ represent the LAR representation level of the current and previous sub-frame respectively. Note that in the case the $n_laguerre_granularity$ changes from %1 to %0 going from sub-frame sf-1 to sf, the value $nlar_{rl,psf}[i]$ is first converted to the coarsest quantisation grid according to:

$$nlar_{rl,psf}[i] = 4 * \lfloor nlar'_{rl,psf}[i] / 4 + 0.5 \rfloor.$$

where $nlar'_{rl,psf}[i]$ represents the LAR representation level of the previous sub-frame. For further processing of $nlar_{rl}$ into reflection or a-parameters see sections 8.6.3.3.2 and 8.6.3.3.3.

n_delta_gain[sf][ch] – For sub-frame sf and channel ch this value represents the differential gain coefficient, and is converted to a representation level $ngain_{rl}$ according to:

$$ngain_{rl} = ngain_{rl,p4sf} + n_delta_gain[sf][ch],$$

where $ngain_{rl,p4sf}$ represents the gain representation level for sub-frame sf-4.

n_overlap_lsf – One bit indicating whether LSF coefficients overlap from the previous definition in channel ch.

enable_ps_header – One bit indicating whether PS header information is present. If set to %1, the PS header data configuring the PS decoder is transmitted. Otherwise, the latest configuration persists.

enable_iid – One bit denoting the presence of IID parameters. If enable_iid is set to %1, Inter-channel Intensity Difference (IID) parameters will be sent from this point on in the bit stream. If enable_iid==%0, no IID parameters will be sent from this point on in the bit stream.

iid_mode – The configuration of IID parameters (number of bands and quantisation grid, iid_quant) is determined by iid_mode. Eight different configurations for IID parameters are supported (see Table 8.19).

Table 8.19 - IID mode configurations

iid_mode	nr_iid_par_tab	nr_ipdopd_par_tab	iid_quant	Index range
0 (000)	10	5	0	-7...7
1 (001)	20	11		-7...7
2 (010)	34	17		-7...7
3 (011)	10	5	1	-15...15
4 (100)	20	11		-15...15
5 (101)	34	17		-15...15
6 (110)	reserved			
7 (111)	reserved			

If no IID data is sent in the bit-stream, all IID parameters are reset to 0 (i.e. index=0).

The default and the fine quantization grids for IID, iid_quant = %0 and iid_quant = %1 are as provided in Table 8.B.18 and Table 8.B.17 respectively.

Table 8.20 - Default quantization grid for IID.

Index	-7	-6	-5	-4	-3	-2	-1	0
IID [dB]	-25	-18	-14	-10	-7	-4	-2	0
Index	1	2	3	4	5	6	7	
IID [dB]	2	4	7	10	14	18	25	

Table 8.21 - Fine quantization grid for IID.

Index	-15	-14	-13	-12	-11	-10	-9	-8
IID [dB]	-50	-45	-40	-35	-30	-25	-22	-19
Index	-7	-6	-5	-4	-3	-2	-1	0
IID [dB]	-16	-13	-10	-8	-6	-4	-2	0
Index	1	2	3	4	5	6	7	8
IID [dB]	2	4	6	8	10	13	16	19
Index	9	10	11	12	13	14	15	
IID [dB]	22	25	30	35	40	45	50	

Note that the configuration of the Inter-channel Phase Difference (IPD) / Overall Phase Difference (OPD) parameters, is strictly coupled to the IID configuration. This is also illustrated in Table 8.19.

enable_icc – One bit denoting the presence of ICC parameters. If enable_icc is set to %1, Inter-channel Coherence (ICC) parameters will be sent from this point on in the bit stream. If enable_icc==%0, no ICC parameters will be sent from this point on in the bit stream.

icc_mode – The configuration of Inter-channel Coherence parameters (number of bands and quantisation grid) is determined by icc_mode. Eight different configurations are supported for IC parameters (see Table 8.22).

Table 8.22 - ICC mode configuration

icc_mode	nr_icc_par_tab	Index range	Mixing procedure
0 (000)	10	0...7	R _a
1 (001)	20	0...7	
2 (010)	34	0...7	
3 (011)	10	0...7	R _b
4 (100)	20	0...7	
5 (101)	34	0...7	
6 (110)	reserved		
7 (111)	reserved		

If no ICC data is sent in the bit-stream, all ICC parameters are reset to 1 (i.e. index=0). The default quantization grid for ICC is provided in Table 8.B.19.

Table 8.23 - Quantization grid for ICC.

index	0	1	2	3	4	5	6	7
ρ	1	0.937	0.84118	0.60092	0.36764	0	-0.589	-1

enable_ext – The PS extension layer is enabled using the enable_ext bit. If it is set to %1 the IPD and OPD parameters are sent. If it's disabled, i.e. %0, the extension layer is skipped.

frame_class – The frame_class bit determines whether the parameter positions of the current frame are uniformly spaced accross the frame (FIX_BORDERS: frame_class==%0) or they are defined using the positions described by border_position (VAR_BORDERS: frame_class==%1).

num_env_idx – The number of (sets of) parameters (envelopes) per frame is determined using num_env_idx. In case of fixed parameter spacing (frame_class==%0) and a variable parameter spacing (frame_class==%1) this relation is shown in Table 8.24.

num_env – Local variable denoting the number of stereo envelopes (sets of parameters). num_env==0 signals that no new stereo parameters are transmitted and that the last parameters in the previous ps_data() element shall be kept unchanged and applied to the current ps_data() element.

Table 8.24 - Number of parameter sets num_env as a function of num_env_idx in case of fixed and variable spacing.

num_env_idx	num_env_tab[frame_class][num_env_idx]	
	frame_class==0	frame_class==1
0	0	1
1	1	2
2	2	3
3	4	4

border_position[e] – In case of variable parameter spacing the parameter positions are determined by border_position[e]. It contains the QMF sample index n_e for parameter set e of the current ps_data() element.

iid_dt[e] – This flag describes for envelope index e, whether the IID parameters are coded differentially over time (iid_dt==%1) or over frequency (iid_dt==%0). In the case iid_mode is different from the previous envelope (e-1), iid_dt[e] shall have the value 0% forcing frequency differential coding.

iid_data() – syntactic element containing IID data.

icc_dt[e] – This flag describes for envelope index e, whether the ICC parameters are coded differentially over time (icc_dt==%1) or over frequency (icc_dt==%0). In the case icc_mode is different from the previous envelope (e-1), icc_dt[e] shall have the value 0% forcing frequency differential coding.

icc_data() – syntactic element containing ICC data.

cnt – Local variable denoting the number of bytes used for the ps_extension() element.

ps_extension_size – The length of the PS extension layer is ps_extension_size, measured in bytes. If the extension size makes use of the escape code (ps_extension_size == 15) the length of the extension layer is extended by an additional amount of bytes.

esc_count – In case the escape code (ps_extension_size == 15) is used, esc_count describes the additional length of the PS extension layer measured in bytes.

num_bits_left – Local variable describing the number of bits left for reading in the ps_extension() element.

ps_extension_id – The identification tag (version) of the PS extension layer is given by ps_extension_id. Currently only one version is supported (see Table 8.25).

Table 8.25 - Description of ps_extension_id

ps_extension_id	Version
00 (0)	v0
01 (1)	reserved
10 (2)	reserved
11 (3)	reserved

fill_bits – These fill_bits accomplish byte-alignment of the ps_extension() data.

enable_ipdopd – The application of IPD and OPD parameters in the bit-stream is denoted by enable_ipdopd. If set (enable_ipdopd==%1) IPD and OPD parameters are sent, if disabled (enable_ipdopd==%0) no IPD and OPD parameters are sent for the current frame in the bit-stream. The quantization grid for both IPD and OPD is provided in Table 8.26. If no IPD or OPD data is sent in the bit-stream, all IPD and OPD parameters are set to 0 (i.e. index=0).

Table 8.26 - Quantization grid for IPD/OPD.

Index	0	1	2	3	4	5	6	7
Representation level	0	$\frac{\pi}{4}$	$\frac{\pi}{2}$	$\frac{3\pi}{4}$	π	$\frac{5\pi}{4}$	$\frac{3\pi}{2}$	$\frac{7\pi}{4}$

ipd_dt[e] – This flag describes for envelope index e, whether the IPD parameters are coded differentially over time (ipd_dt==%1) or over frequency (ipd_dt==%0). In the case iid_mode is different from the previous envelope (e-1), ipd_dt[e] shall have the value 0% forcing frequency differential coding.

ipd_data() – syntactic element containing IPD data.

opd_dt[e] – This flag describes for envelope index e, whether the OPD parameters are coded differentially over time (opd_dt==%1) or over frequency (opd_dt==%0). In the case iid_mode is different from the previous envelope (e-1), opd_dt[e] shall have the value 0% forcing frequency differential coding.

opd_data() – syntactic element containing OPD data.

reserved_ps – This bit is reserved and has a value %0.

iid_par_dt[e][b] – In case of differential coding of IID parameters over time ($iid_dt[e]==1$), $iid_par_dt[e][b]$ describes the IID index difference with respect to the b^{th} parameter position for envelope $e-1$. If no previous parameter is available, $iid_par_dt[e][b]$ represents the IID index difference with respect to the decoded value 0 (i.e. index=0). The IID index $iid_par[e][b]$ is determined as:

$$iid_par[e][b] = iid_par[e-1][b] + iid_par_dt[e][b]$$

where $iid_par[e-1][b]$ represents the IID index of the previous envelope, $e-1$. The IID value $iid[b]$ is obtained by using $iid_par[e][b]$ as an index to Table 8.20 or Table 8.21, depending on iid_mode .

iid_par_df[e][b] – In case of differential coding of IID parameters over frequency ($iid_dt[e]==0$), $iid_par_df[e][b]$ describes the IID difference with respect to the $(b-1)^{th}$ parameter in envelope e . If no previous parameter is available, $iid_par_df[e][b]$ represents the IID difference with respect to the decoded value 0 (i.e. index=0). The IID index $iid_par[e][b]$ is determined as:

$$iid_par[e][0] = iid_par_df[e][0]$$

$$iid_par[e][b] = iid_par[e][b-1] + iid_par_df[e][b] \quad \text{for } b > 0$$

where $iid_par[e][b-1]$ represents the IID index of the previous IID value for envelope e . The IID value $iid[b]$ is obtained by using $iid_par[e][b]$ as an index to Table 8.20 or Table 8.21, depending on iid_mode .

icc_par_dt[e][b] – In case of differential coding of ICC parameters over time ($icc_dt[e]==1$), $icc_par_dt[e][b]$ describes the difference with respect to the b^{th} parameter position for envelope $e-1$. If no previous parameter is available, $icc_par_dt[e][b]$ represents the ICC difference with respect to the decoded value 1 (i.e. index=0). The ICC index $icc_par[e][b]$ is determined as:

$$icc_par[e][b] = icc_par[e-1][b] + icc_par_dt[e][b]$$

where $icc_par[e-1][b]$ represents the ICC index of the previous envelope, $e-1$. The ICC value $\rho[b]$ is obtained by using $icc_par[e][b]$ as an index to Table 8.23.

icc_par_df[e][b] – In case of differential coding of ICC parameters over frequency ($icc_dt[e]==0$), $icc_par_df[e][b]$ describes the ICC difference with respect to the $(b-1)^{th}$ parameter for envelope e . If no previous parameter is available, $icc_par_df[e][b]$ represents the ICC difference with respect to the decoded value 1 (i.e. index=0). The ICC index $icc_par[e][b]$ is determined as:

$$icc_par[e][0] = icc_par_df[e][0]$$

$$icc_par[e][b] = icc_par[e][b-1] + icc_par_df[e][b] \quad \text{for } b > 0$$

where $icc_par[e][b-1]$ represents the ICC index of the previous ICC value for envelope e . The ICC value $\rho[b]$ is obtained by using $icc_par[e][b]$ as an index to Table 8.23.

ipd_par_dt[e][b] – In case of differential coding of IPD parameters over time ($ipd_dt[e]==1$), $ipd_par_dt[e][b]$ describes the IPD difference with respect to the b^{th} parameter position for envelope e . If no previous parameter is available, $ipd_par_dt[e][b]$ represents the IPD difference with respect to the decoded value 0 (i.e. index=0). Note: for IPD parameters modulo-8 differential coding is applied. The IPD index $ipd_par[e][b]$ is determined as:

$$ipd_par[e][b] = \text{mod}(ipd_par[e-1][b] + ipd_par_dt[e][b], 8)$$

where $ipd_par[e-1][b]$ represents the IPD index of the previous envelope, $e-1$. The IPD value $ipd[b]$ is obtained by using $ipd_par[e][b]$ as an index to Table 8.26.

ipd_par_df[e][b] – In case of differential coding of IPD parameters over frequency ($ipd_dt[e]==0$), $ipd_par_df[e][b]$ describes the IPD difference with respect to the $(b-1)^{th}$ parameter for envelope e . If no previous parameter is available, $ipd_par_df[e][b]$ represents the IPD difference with respect to the decoded

value 0 (i.e. index=0). Note: for IPD parameters modulo-8 differential coding is applied. The IPD index $ipd_par[e][b]$ is determined as:

$$\begin{aligned} ipd_par[e][0] &= ipd_par_df[e][0] \\ ipd_par[e][b] &= \text{mod}(ipd_par[e][b-1] + ipd_par_df[e][b], 8) \quad \text{for } b > 0 \end{aligned}$$

where $ipd_par[e][b-1]$ represents the IPD index of the previous IPD value for envelope e . The IPD value $ipd[b]$ is obtained by using $ipd_par[e][b]$ as an index to Table 8.23.

opd_par_dt[e][b] – In case of differential coding of OPD parameters over time ($opd_dt[e]==\%1$), $opd_par_dt[e][b]$ describes the OPD difference with respect to the b^{th} parameter position for envelope ($e-1$). If no previous parameter is available, $opd_par_dt[e][b]$ represents the OPD difference with respect to the decoded value 0 (i.e. index=0). Note: for OPD parameters modulo-8 differential coding is applied. The OPD index $opd_par[e][b]$ is determined as:

$$opd_par[e][b] = \text{mod}(opd_par[e-1][b] + opd_par_dt[e][b], 8)$$

where $opd_par[e-1][b]$ represents the OPD index of the previous envelope, $e-1$. The OPD value $opd[b]$ is obtained by using $opd_par[e][b]$ as an index to Table 8.26.

opd_par_df[e][b] – In case of differential coding of OPD parameters over time ($opd_dt[e]==\%0$), $opd_par_df[e][b]$ describes the OPD difference with respect to the $(b-1)^{\text{th}}$ parameter position for envelope e . If no previous parameter is available, $opd_par_df[e][b]$ represents the OPD difference with respect to the decoded value 0 (i.e. index=0). Note: for OPD parameters modulo-8 differential coding is applied. The OPD index $opd_par[e][b]$ is determined as:

$$\begin{aligned} opd_par[e][0] &= opd_par_df[e][0] \\ opd_par[e][b] &= \text{mod}(opd_par[e][b-1] + opd_par_df[e][b], 8) \quad \text{for } b > 0 \end{aligned}$$

where $opd_par[e][b-1]$ represents the OPD index of the previous OPD value for envelope e . The OPD value $opd[b]$ is obtained by using $opd_par[e][b]$ as an index to Table 8.23.

8.5.3 Indexing of sub-frames

In the case differential coding is applied from one sub-frame to the next, a negative sub-frame index sf of frame k might be indexed. In case this occurs, the negative sub-frame shall be corrected according to

$$sf = sf + nrof_subframes.$$

Note that the sub-frame index thus obtained resides in frame $k-1$. Similarly, in the case that sf is larger than $nrof_subframes$, the sub-frame shall be corrected according to

$$sf = sf - nrof_subframes.$$

The sub-frame index thus obtained resides in frame $k+1$.

8.6 Decoding processes

The parametric stereo decoder is illustrated in Figure 8.1. After de-formatting the bit-stream, the monaural signal M is reconstructed as the combination of transients, sinusoids and noise. Subsequently the stereo parameters are used to reconstruct the left and right signal from the monaural encoded signal. For dual-mono and mono, the parametric stereo decoder is not used.

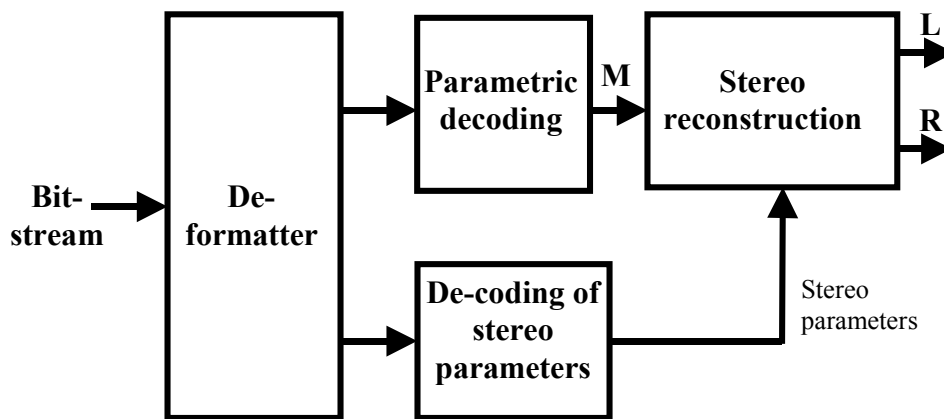


Figure 8.1 - Illustration of the integration of the stereo coding tools into the parametric decoder

The parametric decoder consists of three decoders: the transient decoder (8.6.1), the sinusoidal decoder (8.6.2) and the noise decoder (8.6.3). The decoded signal is obtained by summing the output of the three decoders per frame per channel. In the description of the decoders in the parametric decoder, the indexing of the sub-frame sf and channel ch is occasionally omitted for clarity.

8.6.1 Transients

Two types of transients are defined, a step transient and a transient of the Meixner type. The decoding of the step transient only involves the interpretation of the position. For the Meixner type, a parameterized envelope $g[n]$ and a number of sinusoids need to be decoded.

8.6.1.1 Step transient

A step transient does not generate a signal of its own (as the Meixner transient does), but it is used to modify the window shape for synthesizing sinusoidal and noise components (see sections 8.6.2.4 and 8.6.3).

8.6.1.2 Meixner transient

For the decoding of the Meixner transient, first the envelope needs to be generated. For the envelope the following parameters are required: the start position t_{loc} , the onset slope indicated by t_b_par (b parameter) and the decaying slope represented by t_chi_par (ξ parameter).

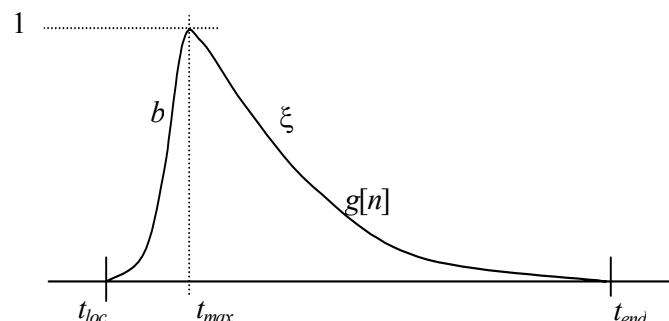


Figure 8.2 - The Meixner envelope is described by the function $g[n]$. The attack slope is controlled by the parameter b . The decay is controlled by the parameter ξ

The start time of the envelope, t_{loc} , is referred to as $n = 0$ for ease of explanation. This envelope $g[n]$ is generated according to

$$g[0] = \frac{(1 - \xi^2)^{\frac{b}{2}}}{a_{\max}},$$

$$g[n] = g[n-1] \cdot \xi \cdot \sqrt{\frac{b+n-1}{n}},$$

for $n=1$ up to and including t_{end} . The end position of the transient window t_{end} is defined below. The maximum $a_{\max}$ at position $t_{\max}$ is given by the approximations

$$t_{\max} = \frac{b-1}{-2 \cdot \log_e(\xi)},$$

$$a_{\max} = \sqrt{\frac{-2 \cdot \log_e(\xi)}{\Gamma(b)}} \cdot \left(\frac{b-1}{e}\right)^{\frac{b-1}{2}}.$$

These complex expressions, especially that for $a_{\max}$, have been evaluated for the allowed values of $t_{\text{b_par}}$ and $t_{\text{chi_par}}$, and are tabulated below.

Table 8.27 - $t_{\max}$ for all possible values of $t_{\text{b_par}}$ and $t_{\text{chi_par}}$.

$t_{\max}$		$t_{\text{b_par}}$			
		0	1	2	3
$t_{\text{chi_par}}$	0	15	30	45	59
	1	20	39	57	75
	2	30	56	79	100
	3	57	96	126	150

Table 8.28 - $a_{\max}$ for all possible values of $t_{\text{b_par}}$ and $t_{\text{chi_par}}$.

$a_{\max}$		$t_{\text{b_par}}$			
		0	1	2	3
$t_{\text{chi_par}}$	0	0.152713500109658	0.131630525645664	0.120142673294398	0.112550174511598
	1	0.132843681407528	0.115639700421076	0.106510539071702	0.100663024431527
	2	0.109279971016712	0.0971964875412947	0.0909719057150294	0.0872632874594248
	3	0.0797175749717262	0.0744985442180281	0.0723059623257423	0.0715041477354716

The transient needs to be synthesized until t_{end} . This position t_{end} is defined as the end of the second complete subsequent sub-frame after the transient position t_{loc} . This is illustrated in Figure 8.3.