
**Information technology — MPEG audio
technologies —**

**Part 1:
MPEG Surround**

*Technologies de l'information — Technologies audio MPEG —
Partie 1: Ambiance MPEG*



Reference number
ISO/IEC 23003-1:2007(E)

AVA Documents
Document Sharing Hub
© ISO/IEC 2007

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 2007

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

Contents

Page

Foreword.....	v
Introduction	vi
1 Scope	1
2 Normative references	1
3 Terms and definitions.....	1
3.1 Definitions	1
3.2 Notation	4
3.3 Operations	5
3.4 Constants	5
3.5 Variables	6
4 MPEG Surround overview.....	9
4.1 Introduction	9
4.2 Basic structure	10
4.3 Tools and functionality.....	11
4.4 Inter-connection of MPEG Surround with audio coders.....	15
4.5 Delay and synchronization	15
4.6 Downmix gain.....	17
4.7 MPEG Surround Profiles and Levels	17
5 Syntax	20
5.1 Payloads for Spatial Audio Coding.....	20
5.2 Definition	39
6 Decoding process.....	56
6.1 Compressed data stream decoding and de-quantization.....	56
6.2 Enhanced Matrix Mode of MPEG Surround	78
6.3 Time / frequency transforms	81
6.4 MPEG Surround processing overview	84
6.5 Calculation of pre-matrix M1, mix-matrix M2 and post-matrix M3.....	109
6.6 Decorrelators.....	133
6.7 Subband Domain Temporal Processing (STP)	137
6.8 Guided Envelope Shaping (GES)	143
6.9 Residual coding	148
6.10 Low Power MPEG Surround decoding	161
6.11 MPEG Surround Binaural coder	174
7 Transport of Spatial Audio Side Information	194
7.1 Overview	194
7.2 Transport and Signalling in an MPEG Environment	194
7.3 Transport of MPEG Surround over PCM channels	198
Annex A (normative) Tables.....	204
A.1 Huffman Tables	204
A.2 Miscellaneous Tables	225
Annex B (normative) HRTF filter conversion (multi-slot).....	243
B.1 Determination of a multi-slot QMF representation of the time domain HRTF filters	243
Annex C (informative) HRTF filter conversion (single-slot).....	246
C.1 Determination of single slot (parametric) QMF representation of the HRTF filters	246
Annex D (informative) Reverberation	248
Annex E (informative) HRTF filter complexity reduction.....	249

Annex F (informative) Encoder	250
F.1 Overview	250
F.2 Time-frequency transform	250
F.3 Framing	250
F.4 Two channels to one channel module (R-OTT)	251
F.5 Three channels to two channels module (R-TTT)	252
F.6 Residual coding	254
F.7 Parameter quantization and Coding	255
F.8 Additional features	255
F.9 MPEG Surround over PCM channels encoding	259
Annex G (informative) Guidelines for transport in non-MPEG environments	261
G.1 Transport in conjunction with existing transmission systems	261
G.2 Self-synchronizing spatial audio bitstream	261
G.3 Usage of S3AC transport format	262
Annex H (informative) Scalable Channel Decoding for Reduced Speaker Configurations	263
H.1 Introduction	263
H.2 Scalable Channel Decoding for Arbitrary Tree Structure	263
H.3 Scalable Channel Decoding for Existing Tree Structure	268
Annex I (informative) Multichannel Visualization Tools for MPEG Surround	276
I.1 Overview	276
I.2 Visual object estimation	276
Annex J (informative) Patent statement	280

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.

ISO/IEC 23003-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

ISO/IEC 23003 consists of the following parts, under the general title *Information technology — MPEG audio technologies*:

— *Part 1: MPEG Surround*

Introduction

The following MPEG International Standard describes the coding of multi-channel signals based on a downmixed signal of the original multi-channel signal, and associated spatial parameters. It offers lowest possible data rate for coding of multi-channel signals, as well as an inherent mono or stereo downmix signal included in the data stream. Hence, a mono or stereo signal can be expanded to multi-channel by a very small additional data overhead. Furthermore, the International Standard describes binaural decoding of the MPEG Surround stream, enabling a surround sound experience over headphones. The International Standard furthermore defines an Enhanced Matrix Mode that enables a multi-channel upmix from a stereo signal without any spatial parameters.

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.

The 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 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 the ISO and IEC. Information may be obtained from the companies listed in Annex J.

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

Information technology — MPEG audio technologies —

Part 1: MPEG Surround

1 Scope

This International Standard describes the MPEG Surround standard (Spatial Audio Coding, SAC), that is capable of re-creating N channels based on $M < N$ transmitted channels, and additional control data. In the preferred modes of operating the spatial audio coding system, the M channels can either be a single mono channel or a stereo channel pair. The control data represents a significant lower data rate than required for transmitting all N channels, making the coding very efficient while at the same time ensuring compatibility with both M channel devices and N channel devices.

This International Standard incorporates a number of tools enabling a number of features that allow for broad application of the International Standard. A key feature is the ability to scale the spatial image quality gradually from very low spatial overhead towards transparency. Another key feature is that the compatible decoder input can be made compatible to existing matrix surround technologies. All tools are grouped to cover certain profiles.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 13818-7, *Information technology — Generic coding of moving pictures and associated audio information — Part 7: Advanced Audio Coding (AAC)*

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

3 Terms and definitions

3.1 Definitions

For the purpose of this document, the following terms and definitions apply.

3.1.1

channel

input or output audio channel corresponding to a specific speaker, as given by Table 1 — and illustrated in Figure 1

Table 1 — Channel abbreviation and loudspeaker position

Channel abbreviation	Loudspeaker position
L	Left Front
R	Right Front
C	Center Front
LFE	Low Frequency Enhancement
Ls	Left Surround
Rs	Right Surround
Lc	Left Front Center
Rc	Right Front Center
Lsr	Rear Surround Left
Rsr	Rear Surround Right
Cs	Rear Center
Lsd	Left Surround Direct
Rsd	Right Surround Direct
Lss	Left Side Surround
Rss	Right Side Surround
Lw	Left Wide Front
Rw	Right Wide front
Lv	Left Front Vertical Height
Rv	Right Front Vertical Height
Cv	Center Front Vertical Height
Lvr	Left Surround Vertical Height Rear
Rvr	Right Surround Vertical Height Rear
Cvr	Center Vertical Height Rear
Lvss	Left Vertical Height Side Surround
Rvss	Right Vertical Height Side Surround
Ts	Top Center Surround
LFE2	Low Frequency Enhancement 2

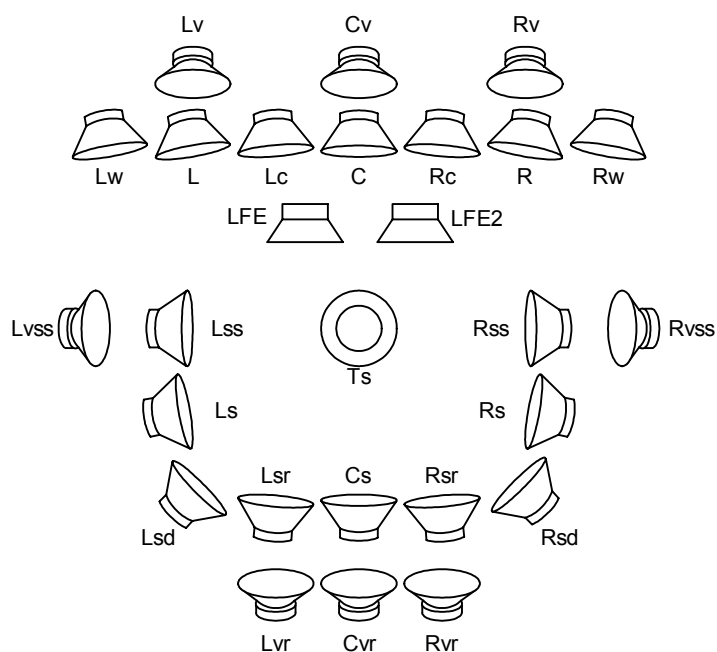


Figure 1 — Loudspeaker positions

3.1.2**Channel Level Difference****CLD**

energy difference between two channels

3.1.3**CLD band**

one or more hybrid subbands for which a single CLD parameter applies

3.1.4**Channel Prediction Coefficient****CPC**

prediction coefficient used for re-creating three channels from two channels

3.1.5**CPC band**

one or more hybrid subbands for which a single CPC parameter applies

3.1.6**hybrid filterbank**

hybrid filter bank structure, consisting of a QMF bank and oddly modulated Nyquist filter banks, used to transform time domain signals into hybrid subband samples

3.1.7**hybrid filtering**

filtering step on a QMF subband signal resulting in multiple hybrid subbands

NOTE The resulting hybrid subbands can be non-consecutive in frequency.

3.1.8**hybrid subband**

subband obtained after hybrid filtering of a QMF subband

NOTE The hybrid subband can have the same time/frequency resolution as a QMF subband.

3.1.9**Inter Channel Correlation****ICC**

correlation or coherence between two channels

3.1.10**ICC band**

one or more hybrid subbands for which a single ICC parameter applies

3.1.11**NA**

Not Applicable

3.1.12**M-N-M configuration**

configuration of the spatial audio coding system that re-creates M channels from N downmixed channel and the corresponding spatial parameters, e.g. 5-1-5 configuration or 5-2-5 configuration

3.1.13**OTT box**

conceptual one-to-two box that takes one channel as input and produces two channels as output

3.1.14**parameter band**

one or more hybrid subbands applicable to one parameter

3.1.15

parameter band border

parameter band delimiter, expressed as a specific hybrid subband

3.1.16

parameter time slot

specific time slot for which the parameter is defined

3.1.17

parameter set

parameters associated with a specific parameter time slot

3.1.18

parameter subset

parameters associated with a specific parameter time slot and a specific OTT box or TTT box

3.1.19

processing band

one or more hybrid subbands defining the finest frequency resolution that could be controlled by the parameters

3.1.20

QMF bank

bank of complex exponentially modulated filters

3.1.21

QMF subband

subband obtained after QMF filtering of a time-domain signal, without any additional hybrid filtering stage

3.1.22

SAC

Spatial Audio Coder

3.1.23

SAC frame

time segment to which processing is applied according to the data conveyed in the corresponding SpatialFrame() syntax element

3.1.24

time segment

group of consecutive time slots

3.1.25

time slot

finest resolution in time for SAC time borders

NOTE One time slot equals one subsample in the hybrid QMF domain.

3.1.26

TTT box

conceptual two-to-three box that takes two channels as input and produces three channels as output

3.2 Notation

The description of the Spatial Audio Coder uses the following notation:

- Vectors are indicated by bold lower-case names, e.g. **vector**.
- Matrices (and vectors of vectors) are indicated by bold upper-case single letter names, e.g. **M**.

- Variables are indicated by italic, e.g. *variable*.
- Functions are indicated as *func*(*x*).
- Real numbers are denoted by *R*
- Complex numbers are denoted by *C*

For equations in the text, normal mathematical interpretation is assumed (no rounding or truncation unless explicitly stated). For flowcharts, normal pseudo-code interpretation is assumed, with no rounding or truncation unless explicitly stated.

3.3 Operations

3.3.1 Scalar operations

X^* is the complex conjugate of X .

$y = \lfloor x \rfloor$ represents rounding down to the nearest integer, i.e., the largest integer number that is less than x .

or $y = \lceil x \rceil$ represents rounding up to the nearest integer, i.e., the smallest integer number that is not less than x .

$y = INT(x)$ represents truncation to integer (only keep the integer part), i.e., conversion to the integer number with the same sign as x and with an absolute value smaller than or equal to the absolute value of x .

$y = \log_2(x)$ is the base-2 logarithm of x .

$y = \log_{10}(x)$ is the base-10 logarithm of x .

$y = \min(...)$ the minimum value in the argument list.

$y = \max(...)$ the maximum value in the argument list.

$y = \text{mod}(x, z)$ is the modulo operation $y = (x - n \cdot z)$ where $n = \text{ceil}(x/z) - 1$ defined for $z \neq 0$.

$y = \text{round}(x)$ represents rounding to the nearest integer. Halfway cases are rounded away from zero.

$y = \text{sign}(x)$ the sign of x , hence defined as -1 for negative values of x , 1 for positive values and 0 for 0.

3.3.2 Vector operations

$y = \text{sort}(x)$. y is equal to the sorted vector x , where the elements of x are sorted in ascending order.

$y = \text{length}(x)$. y is the number of elements of the vector x .

3.4 Constants

ε A constant to avoid division by zero, e.g. 96 dB below maximum signal input.

3.5 Variables

$\mathbf{a}^m(l)$	aliasing condition vector defined for every parameter time slot l and all QMF subbands m that are the last subband (highest in frequency) within a parameter band.
ch	is the current audio channel.
$\mathbf{D}_{\text{ATD}}$	is the three dimensional matrix holding arbitrary tree data, i.e. mapped CLD data, for every OTT box, every parameter set, and M_{proc} bands, for the arbitrary tree.
$\mathbf{D}_{\text{CLD}}$	is the three dimensional matrix holding the dequantized, and mapped CLD data for every OTT box, every parameter set, and M_{proc} bands.
$\mathbf{D}_{\text{ICC}}$	is the three dimensional matrix holding the dequantized, and mapped ICC data for every OTT or TTT box, every parameter set, and M_{proc} bands.
$\mathbf{D}_{\text{CPC}_1}$	is the three dimensional matrix holding the dequantized, and mapped first CPC data for every TTT box, every parameter set, and M_{proc} bands.
$\mathbf{D}_{\text{CPC}_2}$	is the three dimensional matrix holding the dequantized, and mapped second CPC data for every TTT box, every parameter set, and M_{proc} bands.
$\mathbf{D}_{\text{CLD}_1}$	is the three dimensional matrix holding the dequantized, and mapped first CLD data for every TTT box, every parameter set, and M_{proc} bands.
$\mathbf{D}_{\text{CLD}_2}$	is the three dimensional matrix holding the dequantized, and mapped second CLD data for every TTT box, every parameter set, and M_{proc} bands.
$\mathbf{D}_{\text{YYY}}^{\text{Q}}$	is a three dimensional matrix similar to $\mathbf{D}_{\text{YYY}}$, the data is dequantized as for $\mathbf{D}_{\text{YYY}}$, however it has M_{par} bands of data. YYY can be any of CLD, ICC, CPC_1, CPC_2, CLD_1, CLD_2 or ATD.
envRatio_X	is a vector with GES envelope data for each channel X .
F_s	is the sampling frequency of the Spatial Audio Tool.
$\mathbf{G}$	is a three dimensional matrix holding the dequantized and mapped gain correction data for all input channels, parameter set, and M_{proc} bands.
$\mathbf{G}^{\text{Q}}$	is a three dimensional matrix holding the dequantized gain correction data for all input channels, parameter set, and M_{par} bands.
$\text{idxXXX}(.,.)$	is a three dimensional matrix holding the Huffman and delta decoded indices. XXX can be any of CLD, ICC, CPC_1, CPC_2, CLD_1, CLD_2 or ATD.
K	number of hybrid subbands, 71.
K_c	number of complex QMF subbands for Low Power MPEG Surround, $K_c = 8$.
L	number of parameter sets.

M_{proc}	is the number of processing bands, 28.
M_{proc}^c	number of complex processing bands for Low Power MPEG Surround, 12.
M_{par}	is the number of parameter bands signalled by <i>bsFreqRes</i> .
M_{QMF}	is the number of QMF subbands depending on sampling frequency as defined in subclause 6.3.3.
$\mathbf{m}_{\text{resPar}}$	is a vector with the number of parameter bands that each residual cover.
$\mathbf{m}_{\text{resProc}}$	is a vector with the number of processing bands that each residual cover.
$\mathbf{m}_{\text{tttLowProc}}$	is a vector with the number of processing bands for the low range in the TTT boxes.
$\mathbf{m}_{\text{tttHighProc}}$	is a vector with the number of processing bands for the high range in the TTT boxes.
$\mathbf{M}_1^{n,m}$	is the time and frequency variant pre- matrix, defined for all time slots n and all hybrid subbands m .
$\mathbf{M}_2^{n,m}$	is the time and frequency variant mix-matrix, defined for all time slots n and all hybrid subbands m .
$\mathbf{r}^m(l)$	weighted correlation sum based on the input downmix signal, defined for every parameter time slot l and all QMF subbands m that have an adjoining parameter border, used for Low Power MPEG surround.
<i>reset</i>	a variable (in the encoder and the decoder) set to one if certain data elements have changed from the previous frame, otherwise set to zero.
$\mathbf{S}_{\text{proc}}$	a matrix indicating for every parameter set and processing band if smoothing is applied.
$\mathbf{s}_{\text{delta}}$	a vector indicating for every time-slot the smoothing filter coefficient.
$\mathbf{t}$	is of length L and contains parameter time slots for all CLD, ICC, and CPC parameter sets in the current frame.
$\mathbf{Tree}(ch, ,)$	a 3 dimensional matrix, which for each input channel to the Arbitrary Tree have a column for each output signal of the sub-tree indexing the OTT modules the input signal must pass before the output is reached.
$\mathbf{Tree}_{\text{sign}}(ch, ,)$	a 3 dimensional matrix, which for each input channel to the Arbitrary Tree have a column for each output signal of the sub-tree indicating whether the upper (1) or the lower (-1) output of an OTT module should be followed to reach the output signal.
$\mathbf{Tree}_{\text{depth}}(ch, ,)$	a matrix which for each input channel to the Arbitrary Tree have the number of OTT modules that are passed for every output channel.
$\mathbf{Tree}_{\text{outChan}}(ch)$	is a vector with <i>numOutChan</i> elements and each element contain the number of output channels for each Arbitrary Sub-tree.

$\mathbf{v}^{n,m}$	is a vector with the hybrid subband output from the pre gain matrix $\mathbf{M}_1^{n,m} \mathbf{x}^{n,m}$, defined for all time slots n and all hybrid subbands m .
$\mathbf{w}^{n,m}$	is a vector with the hybrid subband output from the decorrelators, the pre-gain matrix and residuals, defined for all time slots n and all hybrid subbands m .
$\mathbf{w}_{\text{diffuse}}^{n,m}$	is a vector with the hybrid subband output from the decorrelators, defined when temporal shaping is used, defined for all time slots n and all hybrid subbands m .
$\mathbf{w}_{\text{direct}}^{n,m}$	is a vector with the hybrid subband output from the decorrelators, the pre-gain matrix and residuals, defined when temporal shaping is used, defined for all time slots n and all hybrid subbands m .
$\mathbf{x}^{n,m}$	is a vector with the hybrid subband input signals (down-mix and residuals), defined for all time slots n and all hybrid subbands m .
$\mathbf{y}^{n,m}$	is a vector with the output hybrid subband signals, which are feed into the hybrid synthesis filter banks, defined for all time slots n and all hybrid subbands m .
$\mathbf{y}_{\text{diffuse}}^{n,m}$	is a vector with the output hybrid subband signals for the diffuse part of the output signal, which is defined when temporal processing is applied, defined for all time slots n and all hybrid subbands m .
$\mathbf{y}_{\text{direct}}^{n,m}$	is a vector with the output hybrid subband signals for the direct part of the output signal, which is defined when temporal processing is applied, defined for all time slots n and all hybrid subbands m .

4 MPEG Surround overview

4.1 Introduction

MPEG Surround is based on a principle called Spatial Audio Coding (SAC). Spatial Audio Coding is a multi-channel compression technique incorporating a backwards compatible downmix, and exploiting the perceptual inter-channel redundancy in multi-channel audio signals to achieve higher compression rates. In order to efficiently represent the spatial image of the multi-channel signal, the MPEG Surround encoder employs a number of spatial parameters:

- Channel level Differences (CLD) describing level differences between two channels,
- Inter channel Correlation / Coherences (ICC) describing the amount of correlation or coherence between two channels and
- Channel Prediction Coefficients (CPC) enabling the recreation of a third channel out of two channels by means of prediction.

During encoding, spatial cues are extracted from the multi-channel audio signal and a downmix is generated. Any number of channels can be used for the downmix, provided that it is less than that used for the original audio signal. The MPEG Surround system supports multi-channel signals of up to 27 channels.

All parameters are efficiently quantized and coded into the spatial bitstream. The quantization process is flexible to allow for a gradual quality versus bit-rate trade-off. For the extreme case where no spatial parameters are employed, the spatial decoder supports an Enhanced Matrix Mode, in which the parameters are derived directly from the down-mix. In addition, support for residual coding is included, that allows coding of the difference between the multi-channel signal as reproduced by the parametric model and the original. Hence, the MPEG Surround system, includes several tools to scale gracefully on the rate/distortion curve, as outlined by the following figure.

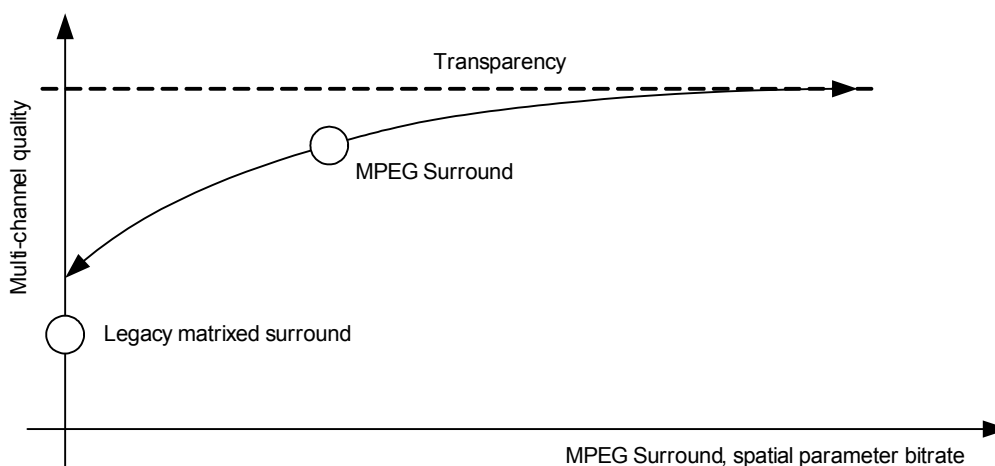


Figure 2 — MPEG Surround bit-rate quality scaling

The downmix can be compressed and transmitted without the need to update existing coders and infrastructures. The spatial cues, or spatial side information, are transmitted in a low bitrate side channel, e.g. the ancillary data portion of the downmix bit-stream.

For most audio productions both a stereo as well as a 5.1 multi-channel downmix is produced from the original multi-track recording by an audio engineer. Naturally, the internal (i.e. automated) downmix produced by the MPEG Surround encoder can differ significantly from the external (artistic) stereo downmix signal as intended by the audio engineer. The MPEG Surround encoder can therefore operate in two modes, "Internal Downmix Mode" or "External Downmix Mode". In the External Downmix Mode the artistic external downmix is transmitted instead of the internal downmix. The difference between this artistic stereo downmix and the automated stereo downmix signal, required by the decoder for optimal multi-channel reconstruction, is coded as part of the spatial side information stream either in a parametric fashion for low bitrate applications or as a wave-form coded difference signal, by means of residual coding.

On the decoder side, a multi-channel up-mix is created from the transmitted downmix signal and spatial side information. In this respect, the Spatial Audio Coding concept can be used as a pre- and post-processing step to upgrade existing systems. Figure 3 illustrates this for a 5.1 original with a stereo downmix. The MPEG Surround decoder supports two different modes, "Normal Mode" in which the spatial parameters obtained from the data stream is used for decoding, and a "Enhanced Matrix Mode" in which the decoder operates on the downmix signal only, estimating the required parameters from the received downmix.

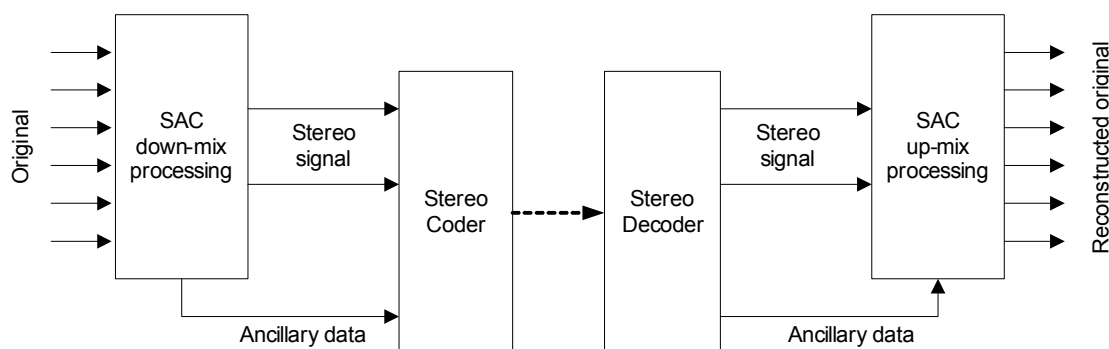


Figure 3 — Overview of the MPEG Surround decoder outlined for a stereo downmix

The spatial decoder combines the spatial parameters with the downmix channels into a multichannel output signal. Further, the MPEG Surround downmix can be provided in two different modes, "Normal Mode" and "Matrix Compatible Mode". If Matrix Compatible Mode is used, the MPEG Surround encoder provides a downmix signal representation that is suitable for playback via legacy matrix surround decoders.

4.2 Basic structure

The basic structure of the MPEG Surround system is shown in Figure 4. The data stream is de-quantized and decoded in the "Bit-stream decoder" modules, as outlined in subclause 6.1. The decoded spatial data is input to the modules calculating the pre-matrix and the mix-matrix as outlined in subclause 6.5. The residual coding data is input to the residual signal decoder as outlined in subclause 6.9. The input signal is analyzed by the QMF hybrid filterbank as outlined in subclause 6.3. The subband signal vector is multiplied with the pre-matrix as outlined in subclause 6.4. Part of the resulting signal vector is input to decorrelator units as outlined in subclause 6.6. The residual signals are also combined with the input signal vector as outlined in subclause 6.4. The mix matrix is applied according to subclause 6.4, and the temporal shaping is applied according to subclause 6.7 and 6.8. The output signal is synthesized according to subclause 6.3.

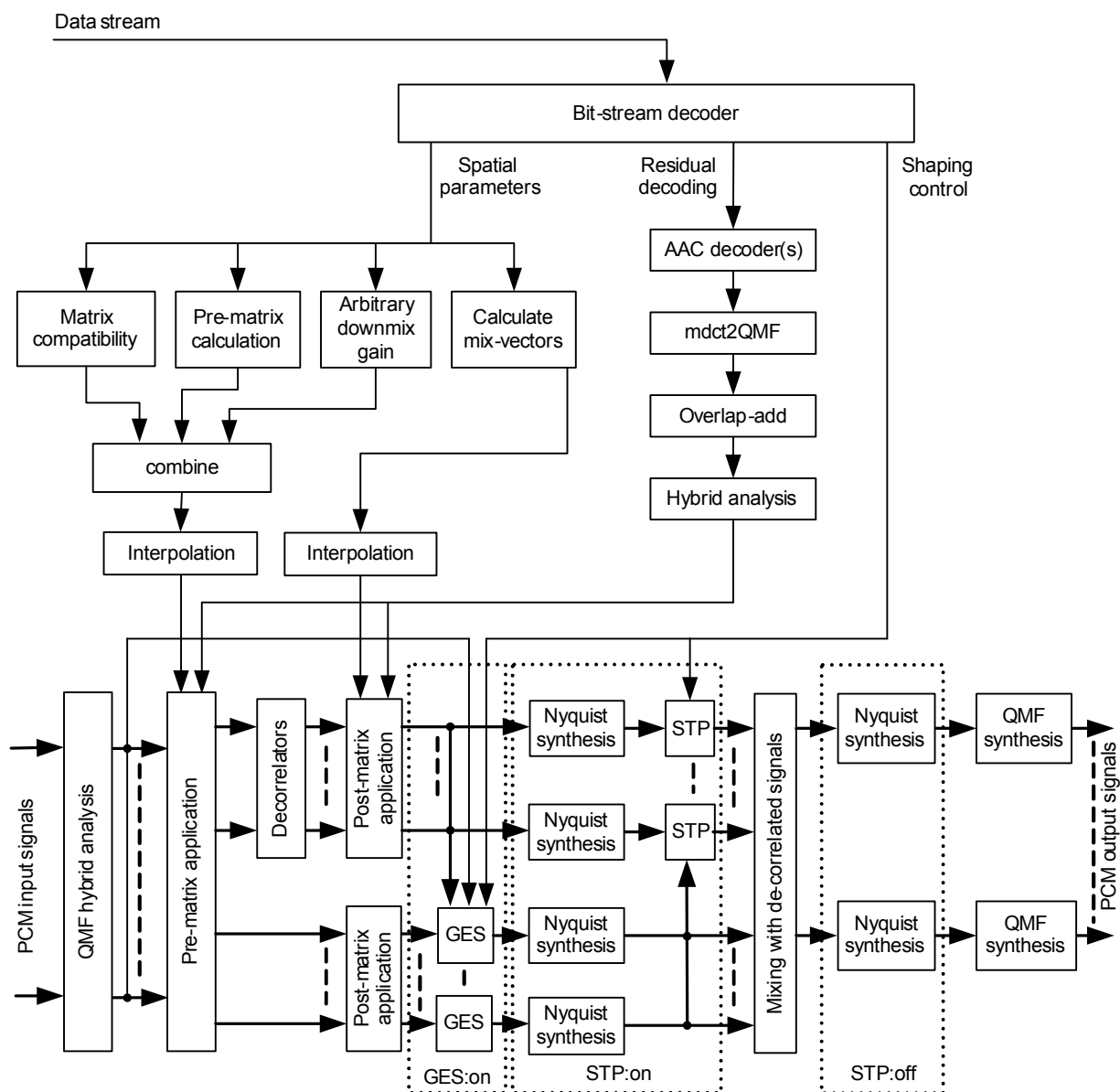


Figure 4 — Overview of the MPEG Surround decoder

4.3 Tools and functionality

4.3.1 General MPEG Surround tools

4.3.1.1 Introduction

The MPEG Surround system incorporates a number of tools that allow for flexible complexity and/or quality trade-off, as well as a diverse set of functionality. In the following subclauses some key-features of MPEG Surround are briefly outlined.

4.3.1.2 Residual coding

As the de-correlated signal is based on certain model assumptions, it is an incomplete representation that allows to scale towards transparent quality for increasing spatial bit-rate. By means of residual coding of the de-correlated signal this issue is addressed. For certain frequency regions where the de-correlated signal is not sufficient, the de-correlated signal is substituted by a residual waveform-type representation. One or more residual signals are coded using the MPEG-2 AAC Low Complexity syntax.

4.3.1.3 Temporal Shaping

Guided envelope shaping (GES) and Subband domain Temporal Processing (STP) are two tools that specifically address the preservation of the temporal structure in the output signal.

4.3.1.4 Enhanced Matrixed Mode of MPEG Surround

The MPEG Surround system includes an enhanced matrixed mode that creates a multi-channel signal based on the downmix without the transmission of a MPEG Surround bitstream. The parameters required in the MPEG Surround decoder are estimated from the received downmix signal. The basic principle is outlined in the following figure.

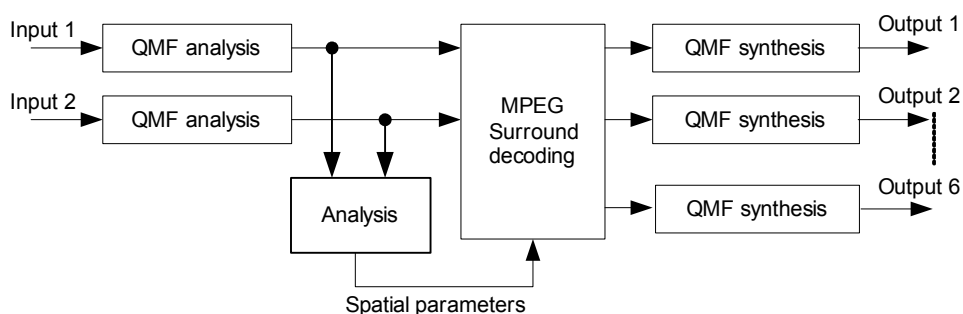


Figure 5 — Enhanced Matrix Mode of MPEG Surround

4.3.1.5 External downmix

The MPEG Surround system supports the use of external downmixes. The MPEG Surround encoder analyzes the difference between the internal downmix created by the MPEG Surround encoder and the external downmix. The difference is compensated for at the MPEG Surround decoder side. The basic principle is outlined in the following figure.

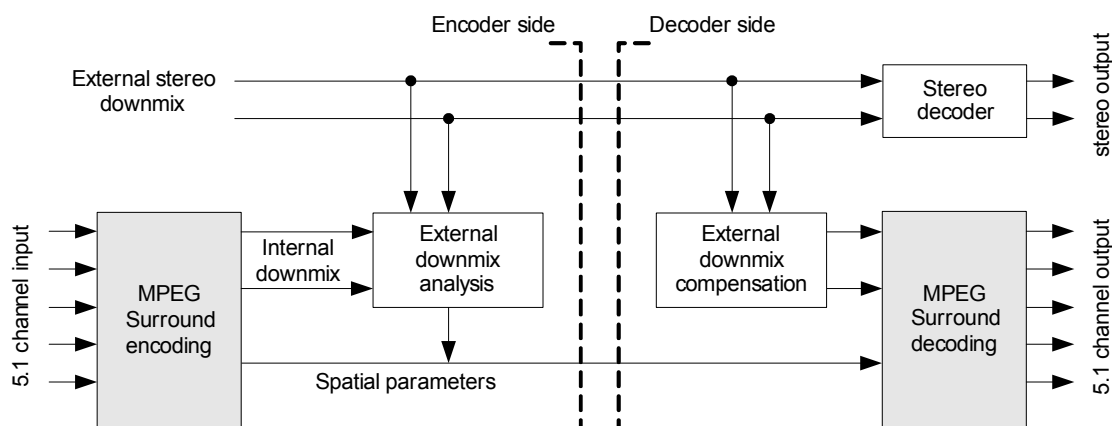


Figure 6 — Support for external downmixes within MPEG Surround

4.3.1.6 Matrix inversion (Matrix Compatible Mode)

In order to supply the highest level of backwards compatibility with legacy systems, the MPEG Surround system supports matrixed encoded downmixes. The MPEG Surround encoder can create a stereo downmix that is matrixed encoded, and can thus be decoded by legacy matrixed surround decoders. The MPEG Surround decoder, will invert the matrixed encoded downmix, and produce the multi-channel signal based on the inverted downmix and the spatial parameters, without any degradation in quality due to the matrixed encoded downmix. The basic principle is outlined in the following figure.

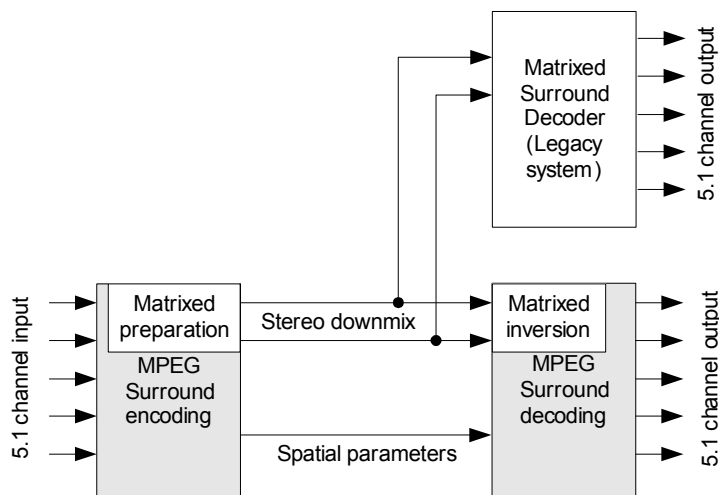


Figure 7 — Matrixed compatible MPEG Surround

4.3.1.7 Binaural and 3D stereo decoding

The MPEG Surround system can be operated in a binaural / 3D stereo mode. This enables a multi-channel impression over headphones by means of Head Related Transfer Function (HRTF) filtering. Two modes of operation can be distinguished, 3D stereo and binaural decoding. In the former, the binauralization is done on the MPEG Surround encoder side, and hence no decoder processing is required for stereo listening. For

normal MPEG Surround decoding, the binaural encoded downmix is inverted similarly to the inversion of the matrixed compatible downmix. The basic principle is outlined in the following figure.

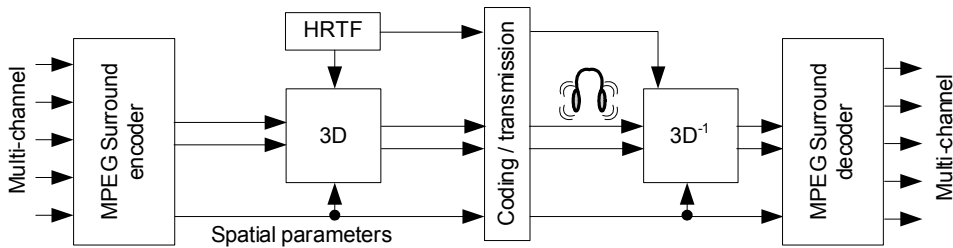


Figure 8 — 3D stereo decoding of MPEG Surround

For binaural decoding on the decoder side, the MPEG Surround downmix and spatial parameters are used in combination with HRTF filters supplied to the decoder. There are two modes of operation, a parametric approach, for lowest complexity, and a filtering approach for highest quality. The basic principle is outlined in the following figure.

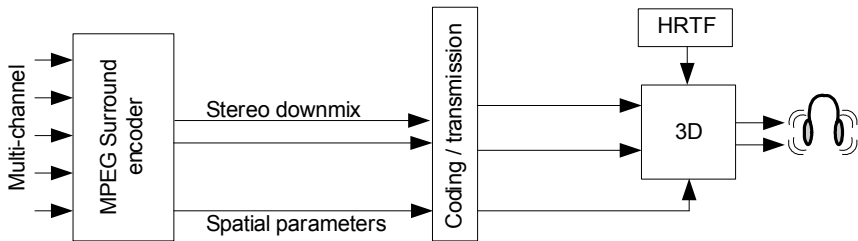


Figure 9 — Binaural decoding of MPEG Surround

4.3.2 High Quality and Low Power MPEG Surround

The MPEG Surround decoder can be implemented in a High Quality version and a Low Power version. The main difference are outlined by the following Table 2, and given in detail in subclause 6.10.

Table 2 — Outline of difference between the High Quality and the Low Power MEG Surround system

Tool or functionality	High Quality version	Low Power version
Filterbank	Complex valued QMF	Partially complex valued QMF
Residual coding	Supported over the entire frequency range	Only supported over the complex valued part of the frequency range
Binaural and 3D stereo	Parametric and HQ filtering supported	Only parametric HRTF filtering supported
Decorrelators		
Aliasing reduction	Not Applicable	Tool operates on the real-valued part of the frequency range

Both versions operate on the same data stream, albeit with different output signals.

4.4 Inter-connection of MPEG Surround with audio coders

The MPEG Surround decoder can be interfaced in either the time-domain or the QMF domain. In general, the MPEG Surround coder interfaces to the downmix channels by means of a 64 bands QMF domain frequency representation, identical to that standardized in ISO/IEC 14496-3 for High Quality Parametric Audio Coding. In the case the spatial coder is combined with HE-AAC, this QMF representation is directly available as an intermediate signal in the HE-AAC coder. In combination with alternative core coders, additional QMF analysis or synthesis modules, as defined in ISO/IEC 14496-3, are required for the spatial decoder and encoder, respectively. A Low Power MPEG Surround decoder cannot be connected in the QMF domain with a High Quality HE-AAC decoder, and vice versa a High Quality MPEG Surround decoder cannot be connected in the QMF domain with a Low Power HE-AAC decoder. For both combinations they shall be combined in the time-domain.

The 64 bands QMF representation of the downmix channels is further refined into hybrid subbands and grouped into parameter bands. By means of a pre-matrix operation on the hybrid subbands, generation of de-correlated signals and mixing, through application of the spatial parameters, multichannel hybrid subband signals are generated. In the last step, these are transformed to the time domain through an inverse QMF hybrid synthesis procedure resulting in the multi-channel time domain signal representation.

4.5 Delay and synchronization

The MPEG Surround decoder introduces a delay when processing the time domain signal from a downmix decoder. Two cases can be distinguished, connecting the MPEG Surround decoder with an arbitrary downmix coder (including High Efficiency AAC) via the time domain, and connecting the MPEG Surround decoder with a High Efficiency AAC decoder directly in the QMF domain. The direct connection in the QMF domain is not possible in case of upsampled or downsampled operation of MPEG Surround, as defined in subclause 6.3.3.

The MPEG Surround decoder introduces a total delay of 1281 samples for the High Quality decoder and 1601 samples for the Low Power decoder, as shown in Figure 10. The analysis filterbank as outlined in subclause 6.3.2 introduces a delay corresponding to 704 samples in the time domain. This is divided between 320 samples for the QMF analysis filtering and 384 samples for the Nyquist filtering. The synthesis filtering as outlined in subclause 6.3.2 introduces a delay corresponding to 257 samples in the time domain. This is divided into 0 samples for the hybrid synthesis filtering and 257 samples for the QMF synthesis filtering.

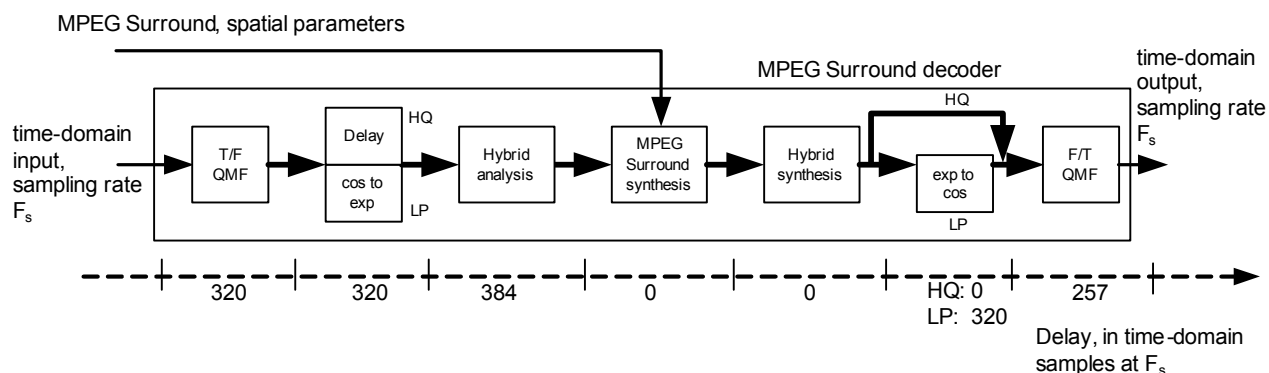


Figure 10 — Delay for the different parts of the MPEG Surround decoder

If the MPEG Surround decoder is connected with an arbitrary downmix coder (including High Efficiency AAC) via the time domain, as shown in Figure 11, the additional delay introduced by the MPEG Surround decoding process will be as outlined above.

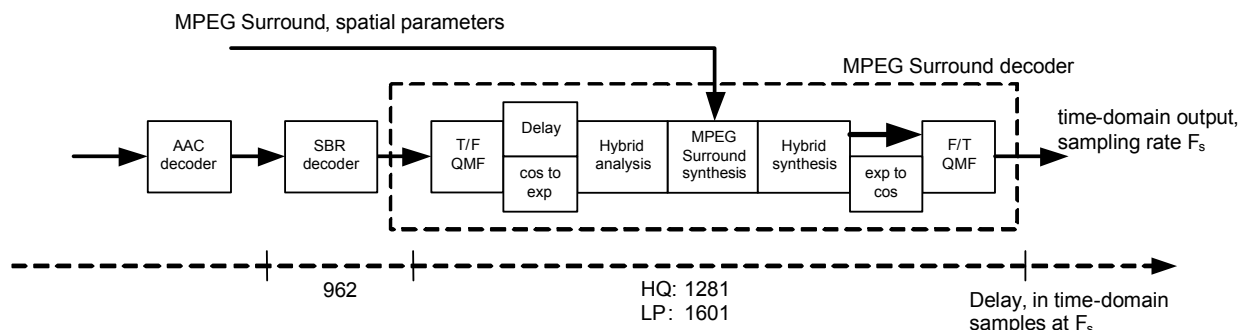


Figure 11 — Delay when connecting MPEG Surround in the time-domain for arbitrary core codec (including HE-AAC)

If the MPEG Surround decoder is directly connected with a High Efficiency AAC decoder via the QMF domain, as shown in Figure 12, the only additional delay is caused by the delay from the real to complex converter in the case of Low Power MPEG Surround and corresponding delay in case of High Quality MPEG Surround. This is because the QMF synthesis filtering is the same for High Efficiency AAC and MPEG Surround, and because the look-ahead of 384 samples needed for the Nyquist filtering is already available in the SBR tool of High Efficiency AAC, as described in ISO/IEC 14496-3 subclause 8.A.3.

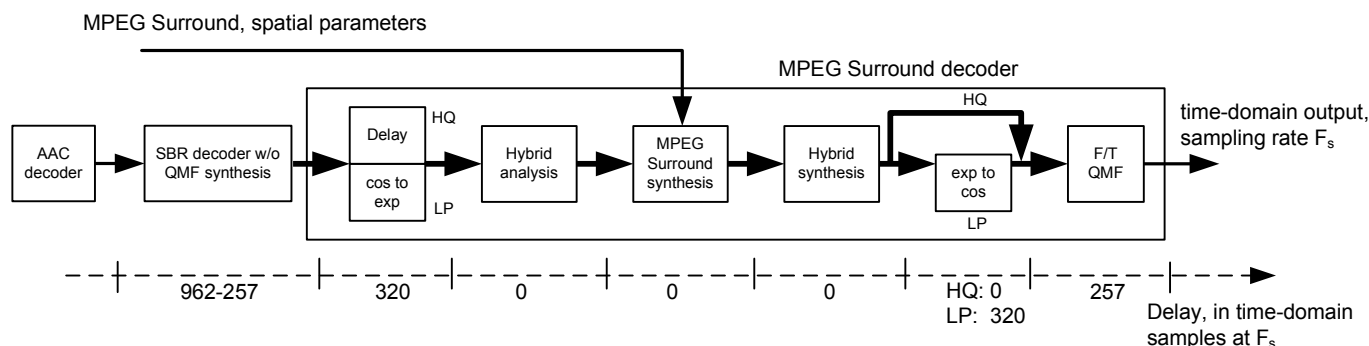


Figure 12 — Delay when connecting MPEG Surround with HE-AAC in the QMF-domain

Transmission of MPEG Surround side information with respect to transmission of the coded downmix signal is done in such a manner that there is no need to delay the downmix signal before it is processed by the MPEG Surround decoder. This means that MPEG Surround data is conveyed such that it is available when needed by the MPEG Surround decoding process. The temporal relationship between downmix data and spatial data is defined in clause 6. Note that special consideration is required if an MPEG Surround decoder and an HE-AAC decoder are connected in the time domain while a connection in the QMF domain would have been possible. In this case, the spatial parameters have to be delayed by 961 time samples, which is the sum of 257 samples for HE-AAC QMF synthesis and 704 samples for MPEG Surround QMF and Nyquist analysis.

In the case of Enhanced Matrix mode operation of the Low Power MPEG Surround decoder, the delay is the same as described above. In the case of Enhanced Matrix mode operation of the High Quality MPEG Surround decoder, the delay is 320 time-domain samples less than described above because the delay corresponding to the real to complex converter is not present.

In the case of single-slot parametric binaural decoding, the delay is the same as described above for both High Quality and Low Power MPEG Surround. In the case of multi-slot HRTF convolution approach, an additional delay is introduced depending on the initial delay of the HRTF used.

All delay figures given in this subclause up to now assume that MPEG Surround is operated in its normal mode using 64 band QMF banks. In the case of upsampled or downsampled operation of MPEG Surround, as defined in subclause 6.3.3, all delay figures measured in samples have to be multiplied by 2 (for upsampled operation) or multiplied by 0.5 (for downsampled operation).

4.6 Downmix gain

The downmix gain is applied to the mono or stereo downmix in the time domain prior to the QMF analysis. If an MPEG Surround decoder is directly connected to the output of an HE-AAC decoder in the QMF domain, the downmix gain is applied to the QMF domain input of MPEG Surround instead. The surround gain is applied to each surround channel (i.e., left surround and right surround for a 5.1 configuration) in the PCM domain after QMF synthesis. The LFE gain is applied to the LFE channel (as present in a 5.1 configuration) in the time domain after QMF synthesis.

4.7 MPEG Surround Profiles and Levels

4.7.1 Introduction

This subclause defines profiles and their levels for MPEG Surround.

Complexity units are defined to give an approximation of the decoder complexity in terms of processing power and RAM usage required for the MPEG Surround 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).

4.7.2 Baseline MPEG Surround Profile

The Baseline MPEG Surround Profile comprises the tools as indicated in the following table.

Table 3 — MPEG Surround Tools relation to the Baseline MPEG Surround Profile

MPEG Surround Tool	Included in the Baseline MPEG Surround Profile
Upmix (M1,M2 and M3, as defined by OTT and TTT modules, including decorrelators)	X
Parameterized External Downmix compensation (subclause 6.5.2.3.3)	X
Matrix compatibility (subclause 6.5.2.4)	X
Enhanced Matrix Mode decoding	X
Temporal Shaping	X
Residual Coding	X
Residual coding based External Downmix compensation (subclause 6.5.2.3.4)	
Binaural decoding (parametric and filtering)	X
3D Audio decoding	X
Low Power Decoding (as alternative to High Quality decoding)	X

Five different hierarchical levels are defined, which allow for different numbers of input and output channels, for different ranges of sampling rates, and for a different bandwidth of the residual signal decoding. The level of the decoder must be equal to or larger than the level of the bitstream in order to ensure proper decoding. In addition, decoders of level 1 and 2 are capable of decoding all bitstreams of level 2 and 3, though at a possibly slightly reduced quality due to the limitations of the decoder. The quality and format of the output of an MPEG Surround decoder furthermore depends on the specific decoder configuration, as detailed below. However, these decoder configuration aspects are completely orthogonal to the different levels of this profile.

An MPEG Surround decoder can be implemented in two different alternative versions:

- High Quality decoder (specified throughout the present document).
- Low Power decoder (difference to the HQ decoder specified in subclause 6.10).

An MPEG Surround decoder can be operated in two different modes, and support for both modes is mandatory for all levels:

- Normal mode (specified throughout the present document).
In this mode, the decoder takes a downmix signal and a side information bitstream as input and generates a multi-channel output signal (or a binaural output signal).
- Enhanced matrix mode (differences to the normal mode specified in subclause 6.2).
In this mode, the decoder takes a stereo downmix signal as input, e.g. a matrix surround stereo downmix, and generates a multi-channel output signal (or a binaural output signal).

An MPEG Surround decoder can provide three different output configurations, and support for all three configurations is mandatory for all levels except for level 1, where multi-channel output is not mandatory:

- Multi-channel output (specified throughout the present document).
Typically a 5.1 channel configuration for loudspeaker presentation. Also more complex configurations like 7.1 channels and beyond are supported by some levels.
- Stereo output (specified in subclause 6.4.7.3).
2-channel stereo output emulating a downmix of the multi-channel signal. This is only applicable in case of a 515 tree configuration because for a 525 tree configuration, the downmix is already a 2-channel stereo signal.
- Binaural output (specified in clause 7).
Stereo output for headphones presentation, generated by integrated binaural rendering based on an HRTF data set provided by the user to the decoder.

The definition of the five levels of the Baseline MPEG Surround Profile is given in Table 4.

Table 4 — Levels of the Baseline MPEG Surround profile

Level	Tree configurations	Max. number output channels	Max. sampling rate [kHz]	Max. bandwidth residual coding [QMF bands]	Max. PCU High Quality decoder	Max. RCU High Quality decoder	Max. PCU Low Power decoder	Max. RCU Low Power decoder
1	515, 525 (Note 1) (Note 4)	2.0	48	0 (Note 2)	12	5	6	4
2	515, 525 (Note 4)	5.1	48	0 (Note 2)	25	15	12	11
3	515, 525 (Note 4)	5.1	48	64 (Note 3)	25	15	12	11
4	515, 525, 757, 727 (Note 4)	7.1	48	64 (Note 3)	34	21	17	15
5	515, 525, 757, 727, plus arbitrary tree extension	32 incl. LFE	96	64 (Note 3)	123 (max. 70 at 48 kHz sampling)	61 (max. 38 at 48 kHz sampling)	80 (max. 44 at 48 kHz sampling)	53 (max. 32 at 48 kHz sampling)

Note 1: This level provides a 2-channel stereo output.

Note 2: Residual coding data, if present in the bitstream, is not utilized, hence the residual decoding tool is not required.

Note 3: A low power decoder utilizes only residual coding data for the first 8 QMF bands, corresponding to approximately 2.7 kHz bandwidth.

Note 4: Arbitrary tree extension data, if present, is not utilized.

The following applies to all levels of the Baseline MPEG Surround Profile:

- Support of stereo output for 515 (subclause 6.4.7.3) is mandatory for all levels.
- Support of “Enhanced matrix mode” (subclause 6.2) is mandatory for all levels, and also in combination with binaural decoding.
- Support of binaural decoding in “Single slot parametric low complexity approach” (subclause 6.11.4.1 and subclause 6.11.4.2.2) and 3DaudioMode (subclause 6.11.5) is mandatory for all levels.
- Support for “Multi-slots HRTF convolution approach” (subclause 6.11.4.2.3) is mandatory for all levels of a high quality decoder (since this mode is not applicable in case of a low power decoder), and the complexity figures assume that a set of HRTF filters with a length of 128 samples (level 1) or 512 samples (levels 2 to 5) in the time domain is used.
- The complexity figures assume that normal decorrelators and not fractional delay decorrelators are used (subclause 6.6.2.3).
- The complexity figures do not include the complexity of the decoding process for the downmix signal itself that is input to the MPEG Surround decoder.
- When an MPEG Surround decoder is connected to an HE-AAC downmix decoder in the QMF domain, the combined PCU and RCU values need to be corrected to take into account that QMF synthesis and analysis filtering of the downmix signals is removed. A High Quality QMF bank has 1.5 PCU and 0.5 RCU, and a Low Power QMF bank has 1.0 PCU and 0.5 RCU. For a 2-channel downmix coded using the HE-AAC Profile at level 2 combined with the Baseline MPEG Surround Profile at level 2, the resulting complexity is:
 - High Quality: $9+25-2*2*1.5 = 28$ PCU $10+15-2*2*0.5 = 23$ RCU
 - Low Power: $7+12-2*2*1.0 = 15$ PCU $8+11-2*2*0.5 = 17$ RCU

5 Syntax

5.1 Payloads for Spatial Audio Coding

Table 5 — Syntax of SpatialSpecificConfig()

Syntax	No. of bits	Mnemonic
SpatialSpecificConfig() {		
bsSamplingFrequencyIndex ;	4	uimsbf
if (bsSamplingFrequencyIndex == 0xf) {		
bsSamplingFrequency ;	24	uimsbf
}		
bsFrameLength ;	7	uimsbf
bsFreqRes ;	3	uimsbf
bsTreeConfig ;	4	uimsbf
bsQuantMode ;	2	uimsbf
bsOnelcc ;	1	uimsbf
bsArbitraryDownmix ;	1	uimsbf
bsFixedGainSur ;	3	uimsbf
bsFixedGainLFE ;	3	uimsbf
bsFixedGainDMX ;	3	uimsbf
bsMatrixMode ;	1	uimsbf
bsTempShapeConfig ;	2	uimsbf
bsDecorrConfig ;	2	uimsbf
bs3DaudioMode ;	1	uimsbf
for (i=0; i<numOttBoxes; i++) {		Note 1
OttConfig(i);		
}		
for (i=0; i<numTttBoxes; i++) {		Note 1
TttConfig(i);		
}		
if (bsTempShapeConfig == 2) {		
bsEnvQuantMode	1	uimsbf
}		
if (bs3DaudioMode) {		
bs3DaudioHRTFset ;	2	uimsbf
if (bs3DaudioHRTFset==0) {		
ParamHRTFset();		
}		
}		
ByteAlign();		
SpatialExtensionConfig();		
}		
Note 1: numOttBoxes and numTttBoxes are defined by Table 40 dependent on bsTreeConfig.		

Table 6 — Syntax of OttConfig()

Syntax	No. of bits	Mnemonic
OttConfig(i) { if (ottModelFe[i]) { bsOttBands[i]; } else { bsOttBands[i] = numBands; } }	5	Note 1 uimsbf Note 2
Note 1: ottModelFe[i] are defined by Table 40 dependent on bsTreeConfig. Note 2: numBands is defined in Table 39 and depends on bsFreqRes.		

Table 7 — Syntax of TttConfig()

Syntax	No. of bits	Mnemonic
TttConfig(i) { bsTttDualMode[i]; bsTttModeLow[i]; if (bsTttDualMode[i]) { bsTttModeHigh[i]; bsTttBandsLow[i]; bsTttBandsHigh[i] = numBands; } else { bsTttBandsLow[i] = numBands; } }	1 3 3 5	uimsbf uimsbf uimsbf uimsbf Note 1 Note 1
Note 1: numBands is defined in Table 39 and depends on bsFreqRes.		

Table 8 — Syntax of ParamHRTFset()

Syntax	No. of bits	Mnemonic
ParamHRTFset() { bsHRTFfreqRes; bsHRTFasymmetric; for (hc=0; hc<HRTFnumChan; hc++) { for (hb =0; hb <HRTFnumBands; hb ++) { bsHRTFlevelLeft[hc][hb]; } if (bsHRTFasymmetric) { for (hb =0; hb <HRTFnumBands; hb ++) { bsHRTFlevelRight[hc][hb]; } } } bsHRTFphase[hc]; if (bsHRTFphase[hc]) { for (hb =0; hb <HRTFnumPhase; hb ++) { bsHRTFphaseLR[hc][hb]; } } }	3 1 6 6 1 6	uimsbf uimsbf Note 1 Note 2 uimsbf Note 2 uimsbf uimsbf Note 3 uimsbf

<pre> } bsHRTFicc[hc]; if (bsHRTFicc[hc]) { for (hb = 0; hb < HRTFnumBands; hb++) { bsHRTFiccLR[hc][hb]; } } } } </pre>	1	uimbsbf
	3	Note 2 uimbsbf
Note 1: HRTFnumChan= 5. Note 2: HRTFnumBands is defined in Table 53 and depends on bsHRTFfreqRes. Note 3: HRTFnumPhase is defined in Table 53 and depends on bsHRTFfreqRes.		

Table 9 — Syntax of SpatialExtensionConfig()

Syntax	No. of bits	Mnemonic
<pre> SpatialExtensionConfig() { sacExtNum = 0; while (BitsAvailable() >= 8) { bsSacExtType; sacExtType[sacExtNum] = bsSacExtType; sacExtNum++; cnt = bsSacExtLen; if (cnt==15) { cnt += bsSacExtLenAdd; } if (cnt==15+255) { cnt += bsSacExtLenAddAdd; } bitsRead = SpatialExtensionConfigData(bsSacExtType) nFillBits = 8*cnt-bitsRead; bsFillBits; } } </pre>	4 4 8 16 nFillBits	Note 1 uimbsbf uimbsbf uimbsbf uimbsbf Note 2 bslbf
Note 1: The function BitsAvailable() returns the number of bits available to be read. Note 2: SpatialExtensionConfigData() returns the number of bits read.		

Table 10 — Syntax of SpatialExtensionConfigData(0)

Syntax	No. of bits	Mnemonic
<pre> SpatialExtensionConfigData(0) { bsResidualSamplingFrequencyIndex; bsResidualFramesPerSpatialFrame; for (i=0; i<numOttBoxes+numTttBoxes; i++) { ResidualConfig(i); } } </pre>	4 2	uimbsbf uimbsbf Note 1
Note 1: numOttBoxes and numTttBoxes are defined by Table 40 and depend on bsTreeConfig.		

Table 11 — Syntax of ResidualConfig()

Syntax	No. of bits	Mnemonic
ResidualConfig(i)		
{		
bsResidualPresent [i];	1	uimsbf
if (bsResidualPresent[i]) {		
bsResidualBands [i];	5	uimsbf
}		
}		

Table 12 — Syntax of SpatialExtensionConfigData(1)

Syntax	No. of bits	Mnemonic
SpatialExtensionConfigData(1)		
{		
bsArbitraryDownmixResidualSamplingFrequencyIndex ;	4	uimsbf
bsArbitraryDownmixResidualFramesPerSpatialFrame ;	2	uimsbf
bsArbitraryDownmixResidualBands ;	5	uimsbf
}		

Table 13 — Syntax of SpatialExtensionConfigData(2)

Syntax	No. of bits	Mnemonic
SpatialExtensionConfigData(2)		
{		
TreeConfig();		
}		

Table 14 — Syntax of TreeConfig()

Syntax	No. of bits	Mnemonic
TreeConfig()		
{		
numOutChanAT = 0;		
numOttBoxesAT = 0;		
for (ch = 0; ch < numOutChan)		Note 1
tmpOpen = 1;		
idx = 0;		
while (tmpOpen > 0) {		
bsOttBoxPresent [ch][idx];	1	uimsbf
if (bsOttBoxPresent[ch][idx]) {		
numOttBoxesAT = numOttBoxesAT+1;		
tmpOpen = tmpOpen+1;		
}		
else {		
numOutChanAT = numOutChanAT + 1;		
tmpOpen = tmpOpen-1;		
}		
idx = idx + 1;		
}		
}		

for (i=0; i<numOttBoxesAT; i++) {		
defaultATD[i] = bsOttDefaultCld ;	1	uimsbf
ottModelFeAT[i] = bsOttModelFe ;	1	uimsbf
if (ottModelFeAT[i]) {		
bsOttBandsAT[i] = bsOttBands ;	5	uimsbf
}		
else {		
bsOttBandsAT[i] = numBands;		Note 2
}		
}		
for (i=0; i<numOutChanAT; i++) {		
bsOutputChannelPos [i]	5	uimsbf
}		
}		
Note 1: numOutChan is defined in Table 40 and depends on bsTreeConfig.		
Note 2: numBands is defined in Table 39 and depends on bsFreqRes.		

Table 15 — Syntax of SpatialFrame()

Syntax	No. of bits	Mnemonic
SpatialFrame()		
{		
FramingInfo();		
bsIndependencyFlag ;	1	uimsbf
OttData();		
TttData();		
SmgData();		
TempShapeData();		
if (bsArbitraryDownmix != 0) {		
ArbitraryDownmixData();		
}		
ByteAlign();		
SpatialExtensionFrame();		
}		

Table 16 — Syntax of FramingInfo()

Syntax	No. of bits	Mnemonic
<pre> FramingInfo() { bsFramingType; bsNumParamSets; if (bsFramingType) { for (ps=0; ps<numParamSets; ps++) { if(ps==0){ bsParamSlot[0]; else{ bsDiffParamSlot[ps]; bsParamSlot[ps] = bsParamSlot[ps-1] + bsDiffParamSlot[ps] + 1; } } } } } </pre>	1 3 nBitsParamSlot(0) nBitsParamSlot(ps)	uimbsbf uimbsbf Note 1 uimbsbf Note 2 uimbsbf Note 2
Note 1: numParamSets is defined by numParamSets = bsNumParamSets + 1. Note 2: nBitsParamSlot(ps) is defined according to $\text{nBitsParamSlot}(0) = \text{ceil}(\log_2(\text{numSlots} - \text{numParamSets} + 1))$ and $\text{nBitsParamSlot}(ps) = \text{ceil}(\log_2(\text{numSlots} - \text{numParamSets} + ps - \text{bsParamSlot}[ps-1]))$ for $0 < ps < \text{numParamSets}$.		

Table 17 — Syntax of OttData()

Syntax	No. of bits	Mnemonic
<pre> OttData() { for (i=0; i<numOttBoxes; i++) { EcData(CLD, i, 0, bsOttBands[i]); } if (bsOneIcc) { EcData(ICC, 0, 0, numBands); } else { for (i=0; i<numOttBoxes; i++) { if (!ottModeLfe[i]) { EcData(ICC, i, 0, bsOttBands[i]); } } } } </pre>		Note 1 Note 2 Note 1 Note 1
Note 1: numOttBoxes and ottModeLfe[i] are defined by Table 40 and depends on bsTreeConfig. Note 2: numBands is defined in Table 39 and depends on bsFreqRes.		

Table 18 — Syntax of TttData ()

Syntax	No. of bits	Mnemonic
<pre> TttData() { tttOff = numOttBoxes; for (i=0; i<numTttBoxes; i++) { if (bsTttModeLow[i] < 2) { EcData(CPC, tttOff+4*i, 0, bsTttBandsLow[i]); EcData(CPC, tttOff+4*i+1, 0, bsTttBandsLow[i]); EcData(ICC, tttOff+4*i, 0, bsTttBandsLow[i]); } else { EcData(CLD, tttOff+4*i, 0, bsTttBandsLow[i]); EcData(CLD, tttOff+4*i+1, 0, bsTttBandsLow[i]); } if (bsTttDualMode[i]) { if (bsTttModeHigh[i] < 2) { EcData(CPC, tttOff+4*i+2, bsTttBandsLow[i], bsTttBandsHigh[i]); EcData(CPC, tttOff+4*i+3, bsTttBandsLow[i], bsTttBandsHigh[i]); EcData(ICC, tttOff+4*i+2, bsTttBandsLow[i], bsTttBandsHigh[i]); } else { EcData(CLD, tttOff+4*i+2, bsTttBandsLow[i], bsTttBandsHigh[i]); EcData(CLD, tttOff+4*i+3, bsTttBandsLow[i], bsTttBandsHigh[i]); } } } } </pre>		Note 1
Note 1: numOttBoxes is defined by Table 40 and depends on bsTreeConfig.		

Table 19 — Syntax of SmgData()

Syntax	No. of bits	Mnemonic
<pre> SmgData() { for (ps=0; ps<numParamSets; ps++) { bsSmoothMode[ps]; if (bsSmoothMode[ps] >= 2) { bsSmoothTime[ps]; } if (bsSmoothMode[ps] == 3) { bsFreqResStrideSmg[ps]; dataBands = (numBands-1)/pbStride+1; for (pg=0; pg<dataBands; pg++) { bsSmgData[ps][pg]; } } } } </pre>	2 2 2 1	Note 1 uimsbf uimsbf uimsbf Note 2 uimsbf
Note 1: numParamSets is defined by numParamSets = bsNumParamSets + 1. Note 2: pbStride is defined in Table 70 and depends on bsFreqResStrideSmg[]. Furthermore the division shall be interpreted as ANSI C integer division. numBands is defined in Table 39 and depends on bsFreqRes.		

Table 20 — Syntax of TempShapeData()

Syntax	No. of bits	Mnemonic
<pre> TempShapeData() { if (bsTempShapeConfig != 0) { bsTempShapeEnable; if (bsTempShapeEnable) { for (ch=0; ch<numTempShapeChan; ch++) { bsTempShapeEnableChannel[ch]; } if (bsTempShapeConfig == 2) { EnvelopeReshapeHuff(bsTempShapeEnableChannel); } else if (bsTempShapeConfig == 3) { /* reserved */ } } } } </pre>	1 1	uimsbf Note 1 uimsbf
Note 1: numTempShapeChan is defined by Table 40 and depends on bsTempShapeConfig and bsTreeConfig.		

Table 21 — Syntax of EnvelopeReshapeHuff()

Syntax	No. of bits	Mnemonic
<pre> EnvelopeReshapeHuff() { for (ch=0; ch<numTempShapeChan; ch++) { if (bsTempShapeEnableChan[ch]) { numValRcvd = 0; while (numValRcvd < numTimeSlots) { (erVal, erLen) = 2Dhuff_dec(hcod2D_EnvRes, bsCodeW); for (k=numValRcvd; k<numValRcvd+erLen; k++) { bsEnvShapeData[ch][k] = erVal; } numValRcvd += erLen; } } } } </pre>	1..x	Note 1 Note 2 vlcibf
Note 1: numTempShapeChan is defined by Table 40 and depends on bsTempShapeConfig and bsTreeConfig. Note 2: numSlots is defined by numSlots = bsFrameLength + 1.		

Table 22 — Syntax of ArbitraryDownmixData()

Syntax	No. of bits	Mnemonic
<pre> ArbitraryDownmixData() { adOff = numOttBoxes+4*numTttBoxes; for (ic=0; ic<numInChan; ic++) { EcData(CLD, adOff+ic, 0, numBands); } } </pre>		Note 1
Note 1: numBands is defined in Table 39 and depends on bsFreqRes.		

Table 23 — EcData()

Syntax	No. of bits	Mnemonic
<pre> EcData(dataType, paramIdx, startBand, stopBand) { dataSets = 0; for (ps=0; ps<numParamSets; ps++) { bsXXXdataMode[paramIdx][ps]; if (bsXXXdataMode[paramIdx][ps] == 3) { dataSets++; } } setIdx = 0; while (setIdx < dataSets) { bsDataPairXXX[paramIdx][setIdx]; bsQuantCoarseXXX[paramIdx][setIdx]; bsFreqResStrideXXX[paramIdx][setIdx]; dataBands = (stopBand-startBand-1)/pbStride+1; EcDataPair(dataType, paramIdx, setIdx, dataBands, bsDataPairXXX[paramIdx][setIdx], bsQuantCoarseXXX[paramIdx][setIdx]); if (bsDataPairXXX[paramIdx][setIdx]) { bsXXXQuantCoarse[paramIdx][setIdx+1] = bsXXXQuantCoarse[paramIdx][setIdx]; bsFreqResStrideXXX[paramIdx][setIdx+1] = bsFreqResStrideXXX[paramIdx][setIdx]; } setIdx += bsDataPairXXX[paramIdx][setIdx]+1; } startBandXXX[paramIdx] = startBand; stopBandXXX[paramIdx] = stopBand; } </pre>	2 1 1 2	Note 1 Note 2 uimsbf uimsbf uimsbf Note 3
Note 1: XXX is to be replaced by the value of dataType (CLD, ICC, CPC, ATD). Note 2: numParamSets is defined by numParamSets = bsNumParamSets + 1. Note 3: pbStride is defined in Table 70 and depends on bsFreqResStride[[]]. Furthermore the division shall be interpreted as ANSI C integer division.		

Table 24 — Syntax of EcDataPair()

Syntax	No. of bits	Mnemonic
EcDataPair(dataType, paramIdx, setIdx, dataBands, pairFlag, coarseFlag)		Note 1
{		
mixedTimePair_flag = 0;		
bsPcmCodingXXX [paramIdx][setIdx];	1	uimsbf
bsPilotCodingXXX[paramIdx][setIdx] = 0;		
if (bsPcmCodingXXX[paramIdx][setIdx] && dataBands>4) {		
bsPilotCodingXXX [paramIdx][setIdx];	1	uimsbf
}		
if (bsPcmCoding[paramIdx][setIdx] && !bsPilotCodingXXX[paramIdx][setIdx]) {		
if (coarseFlag) {		Note 2
numQuantSteps = numQuantStepsXXXCoarse;		
Else {		Note 2
numQuantSteps = numQuantStepsXXXFine;		
}		
aaDataPair = GroupedPcmData(dataType, pairFlag,		
numQuantSteps, dataBands);		
}		
else {		
allowDiffTimeBack = (!bsIndependencyFlag) (setIdx>0);		
(aaDataPairMsbDiff, aPgOffset, pilot, mixedTimePair_flag) =		
DiffHuffData(dataType, pairFlag,		
allowDiffTimeBack, bsPilotCodingXXX[paramIdx][setIdx], dataBands);		
aaDataPairLsb[0] = LsbData(dataType, coarseFlag, dataBands);		
if (pairFlag) {		
aaDataPairLsb[1] = LsbData(dataType, coarseFlag, dataBands);		
}		
}		
/* copy information read by EcDataPair() and its subfunctions		
into non-ambiguous variables for later delta decoding etc. */		
bsDiffTypeXXX[paramIdx][setIdx] = bsDiffType[0];		
bsDiffTimeDirectionXXX[paramIdx][setIdx] = bsDiffTimeDirection[0];		
bsXXXpilot[paramIdx][setIdx] = pilot;		
mixedTimePairXXX[paramIdx][setIdx] = mixedTimePair_flag;		
if (pairFlag) {		
bsDiffTypeXXX[paramIdx][setIdx+1] = bsDiffType[1];		
bsDiffTimeDirectionXXX[paramIdx][setIdx+1] = bsDiffTimeDirection[1];		
bsPcmCodingXXX[paramIdx][setIdx+1] =		
bsPcmCodingXXX[paramIdx][setIdx];		
bsXXXpilot[paramIdx][setIdx+1] = pilot;		
bsPilotCodingXXX[paramIdx][setIdx+1] =		
bsPilotCodingXXX[paramIdx][setIdx];		
mixedTimePairXXX[paramIdx][setIdx+1] = mixedTimePair_flag;		
}		
for (pg=0; pg<dataBands; pg++) {		
if (bsPcmCodingXXX[paramIdx][setIdx] &&		
!bsPilotCodingXXX[paramIdx][setIdx]) {		
bsXXXpcm[paramIdx][setIdx][pg] = aaDataPair[0][pg];		
else {		
bsXXXmsbDiff[paramIdx][setIdx][pg] = aaDataPairMsbDiff[0][pg];		
bsXXXlsb[paramIdx][setIdx][pg] = aaDataPairLsb[0][pg];		
}		
}		

```

    if (pairFlag) {
        if (bsPcmCodingXXX[paramIdx][setIdx+1] &&
            !bsPilotCodingXXX[paramIdx][setIdx+1]) {
            bsXXXpcm[paramIdx][setIdx+1][pg] = aaDataPair[1][pg];
        }
        else {
            bsXXXmsbDiff[paramIdx][setIdx+1][pg] = aaDataPairMsbDiff[1][pg];
            bsXXXlsb[paramIdx][setIdx+1][pg] = aaDataPairLsb[1][pg];
        }
    }
}

```

Note 1: XXX is to be replaced by the value of dataType. (CLD, ICC, CPC, ATD).

Note 2: numQuantStepsXXXCoarse and numQuantStepsXXXFine is defined in Table 71 and depends on dataType.

Table 25 — Syntax of GroupedPcmData()

Syntax	No. of bits	Mnemonic
GroupedPcmData(dataType, pairFlag, numQuantSteps, dataBands)		
{		
numPcmDataBands = dataBands;		
if (pairFlag) {		
numPcmDataBands *= 2;		
}		
for (i=0 ; i<numPcmDataBands ; i+= maxGrpLen) {		Note 1
grpLen = min(maxGrpLen, numPcmDataBands - i);		
nGrpBits = ceil(grpLen*log ₂ (numQuantSteps));		
bsPcmWord;	nGrpBits	uimbsf
for (j = 0; j < grpLen; j++) {		
idx = i + (grpLen-1) - j;		
pcmValue = bsPcmWord % numQuantSteps;		Note 2
if (pairFlag) {		
aaDataPair[idx%2][idx/2] = pcmValue;		Note 2
}		
else {		
aaDataPair[0][idx] = pcmValue;		
}		
bsPcmWord /= numQuantSteps;		
}		
}		
return (aaDataPair);		
}		

Note 1: maxGrpLen is defined in Table 72 and depends on numQuantSteps.

Note 2: % denotes the modulo operator (ANSI C integer math) and returns the remainder of the division.

Table 26 — Syntax of DiffHuffData()

Syntax	No. of bits	Mnemonic
<pre> DiffHuffData(dataType, pairFlag, allowDiffTimeBackFlag, pilotCodingFlag, dataBands) { mixedTimePair_flag = 0; bsDiffType[0] = DIFF_FREQ; bsDiffType[1] = DIFF_FREQ; if (pilotCodingFlag) { pilot = 1Dhuff_dec(hcodPilot_XXX, bsCodeW); } else { if (pairFlag allowDiffTimeBackFlag) { bsDiffType[0]; } if (pairFlag && ((bsDiffType[0] == DIFF_FREQ) allowDiffTimeBackFlag)) { bsDiffType[1]; } } bsCodingScheme; if (bsCodingScheme == HUFF_1D) { (aaHuffData[0]) = HuffData1D(dataType, aDiffType[0], pilotCodingFlag, dataBands); if (pairFlag) { (aaHuffData[1]) = HuffData1D(dataType, aDiffType[1], pilotCodingFlag, dataBands); } } else { /* HUFF_2D */ if (pairFlag) { bsPairing; } else { bsPairing = FREQ_PAIR; } if (bsPairing == FREQ_PAIR) { (aaHuffData[0]) = HuffData2DFreqPair(dataType, aDiffType[0], pilotCodingFlag, dataBands); if(pairFlag) { (aaHuffData[1]) = HuffData2DFreqPair(dataType, aDiffType[1], pilotCodingFlag, dataBands); } } else { /* TIME_PAIR */ (aaHuffData) = HuffData2DtimePair(dataType, aDiffType, pilotCodingFlag, dataBands); if (bsDiffType[0] != bsDiffType[1]) { mixedTimePair_flag = 1; } } } } </pre>	1..x 1 1 1 1	vlcibf Note1,Note2 Uimbsf Uimbsf Uimbsf Uimbsf

<pre> } } /* Inverse differential coding */ if ((bsDiffType[0] == DIFF_TIME) (bsDiffType[1] == DIFF_TIME)) { if (!allowDiffTimeBackFlag && (bsDiffType[0] == DIFF_TIME)) { bsDiffTimeDirection[0] = FORWARDS; } else if (!pairFlag (pairFlag && (bsDiffType[1] == DIFF_TIME))) { bsDiffTimeDirection[0] = BACKWARDS; } else { bsDiffTimeDirection[0]; } if (pairFlag) { bsDiffTimeDirection[1] = BACKWARDS; } } return (aaHuffData, aPgOffset, pilot, mixedTimePair_flag); } </pre>		
Note 1: XXX is to be replaced by the value of dataType (CLD, ICC or CPC, ATD).	1	Uimbsf
Note 2: 1Dhuff_dec() is defined in Annex A.1.		

Table 27 — Syntax of HuffData1D()

Syntax	No. of bits	Mnemonic
<pre> HuffData1D(dataType, diffType, pilotCodingFlag, dataBands) { pgOffset = 0; if (diffType == DIFF_FREQ &&!pilotCodingFlag) { aHuffData1D[0] = 1Dhuff_dec(hcodFirstBand_XXX, bsCodeW); pgOffset = 1; } for (i=pgOffset; i<dataBands; i++) { aHuffData1D[i] = 1Dhuff_dec(hcod1D_XXX_YY, bsCodeW); if (aHuffData1D[i] != 0) { bsSign; if (bsSign) { aHuffData1D[i] = -aHuffData1D[i]; } } } return (aHuffData1D); } </pre>		
	1..x	vlclbf Note1,3
	1..x	vlclbf Note1,2,3
	1	Uimbsf
Note 1: XXX is to be replaced by the value of dataType (CLD, ICC, CPC, ATD). Note 2: YY is to be replaced by "PC", "DF", or "DT", depending on the value of pilotCodingFlag and diffType. Note 3: 1Dhuff_dec() is defined in Annex A.1.		

Table 28 — Syntax of HuffData2DFreqPair()

Syntax	No. of bits	Mnemonic
HuffData2DFreqPair(dataType, diffType, pilotCodingFlag, dataBands)		
{		
LavIdx = 1Dhuff_dec(hcodLavIdx, bsCodeW);	1..3	Vlclbf Note 1
lav = lavTabXXX[LavIdx];		
pgOffset = 0;		
if (diffType == DIFF_FREQ &&!pilotCodingFlag) {		
aHuffData2D[0] = 1Dhuff_dec(hcodFirstBand_XXX, bsCodeW);	1..x	vlclbf Note6
pgOffset = 1;		
}		
escapeCode = hcod2D_XXX_YY_FP_LL_escape;		Note2,3,4,5
/* specific escape code belonging to this Huffman table */		
escCntr = 0;		
for (i=pgOffset; i<dataBands; i+=2) {		
(aTmp[0], aTmp[1]) = 2Dhuff_dec(hcod2D_XXX_YY_FP_LL, bsCodeW);	1..x	vlclbf , Note3,4,5,6
if (bsCodeWord != escapeCode) {		
aTmpSym = SymmetryData(aTmp);		
aHuffData2D[i] = aTmpSym[0];		
aHuffData2D[i+1] = aTmpSym[1];		
}		
else {		
aEscList[escCntr++] = i;		
}		
}		
if (escCntr > 0) {		
aaEscData = GroupedPcmData(dataType, 1, 2*lav+1, escCntr);		
for (i=0; i<escCntr; i++) {		
aHuffData2D[aEscList[i]] = aaEscData[0][i] - lav;		
aHuffData2D[aEscList[i]+1] = aaEscData[1][i] - lav;		
}		
}		
if ((dataBands-pgOffset) % 2) {		Note 7
aHuffData2D[dataBands-1] = 1Dhuff_dec(hcod1D_XXX_YY, bsCodeW);	1..x	vlclbf , Note3,4,6
if (aHuffData2D[dataBands-1] != 0) {		
bsSign ;	1	Uimbsf
if (bsSign) {		
aHuffData2D[dataBands-1] = -aHuffData2D[dataBands-1];		
}		
}		
}		
return (aHuffData2D);		
}		

Note 1: lavTabXXX is defined in Table 76.

Note 2: The escape code tables are defined in Table A.8, Table A.9, and Table A.10. For some Huffman tables no escape code is needed since all possible values are covered by the Huffman table.

Note 3: XXX is to be replaced by the value of dataType (CLD, ICC, CPC, ATD).

Note 4: YY is to be replaced by "PC", "DF", or "DT", depending on the value of pilotCodingFlag and diffType.

Note 5: LL is to be replaced by the value of lav.

Note 6: 1Dhuff_dec() and 2Dhuff_dec() are defined in Annex A.1.

Note 7: % denotes the modulo operator (ANSI C integer math) and returns the remainder of the division.

Table 29 — Syntax of HuffData2DTimePair()

Syntax	No. of bits	Mnemonic
HuffData2DTimePair(dataType, aDiffType, pilotCodingFlag, dataBands)		
{		
LavIdx = 1Dhuff_dec(hcodLavIdx, bsCodeW);	1..3	Vlclbf
lav = lavTabXXX[LavIdx];		Note 1
pgOffset = 0;		
if (((aDiffType[0] == DIFF_FREQ) (aDiffType[1] == DIFF_FREQ))		
&&!pilotCodingFlag) {		
aaHuffData2D[0][0] = 1Dhuff_dec(hcodFirstBand_XXX, bsCodeW);	1..x	vlclbf ,
		Note3,6
aaHuffData2D[1][0] = 1Dhuff_dec(hcodFirstBand_XXX, bsCodeW);	1..x	vlclbf ,
		Note3,6
pgOffset = 1;		
}		
escapeCode = hcod2D_XXX_YY_TP_LL_escape;		Note2,3,4,5
/* specific escape code belonging to this Huffman table */		
escCntr = 0;		
if ((aDiffType[0] == DIFF_TIME) (aDiffType[1] == DIFF_TIME)) {		
diffType = DIFF_TIME;		
}		
Else {		
diffType = DIFF_FREQ;		
}		
for (i=pgOffset; i<dataBands; i++) {		
(aTmp[0], aTmp[1]) = 2Dhuff_dec(hcod2D_XXX_YY_TP_LL, bsCodeW);	1..x	vlclbf ,
		Note3,4,5,6
if (bsCodeW != escapeCode) {		
aTmpSym = SymmetryData(aTmp);		
aaHuffData2D[0][i] = aTmpSym[0];		
aaHuffData2D[1][i] = aTmpSym[1];		
}		
else {		
aEscList[escCntr++] = i;		
}		
}		
}		

```

if ( escCntr > 0 ) {
    aaEscData = GroupedPcmData(dataType, 1, 2*lav+1, escCntr);
    for ( i=0; i<escCntr; i++ ) {
        aaHuffData2D[0][aEscList[i]] = aaEscData[0][i] - lav;
        aaHuffData2D[1][aEscList[i]] = aaEscData[1][i] - lav;
    }
}

return (aaHuffData2D);
}

```

Note 1: lavTabXXX is defined in Table 76.

Note 2: The escape code tables are defined in Table A.8, Table A.9, and Table A.10. For some Huffman tables no escape code is needed since all possible values are covered by the Huffman table.

Note 3: XXX is to be replaced by the value of dataType (CLD, ICC, CPC, ATD).

Note 4: YY is to be replaced by "PC", "DF", or "DT", depending on the value of pilotCodingFlag and diffType.

Note 5: LL is to be replaced by the value of lav.

Note 6: 1Dhuff_dec() and 2Dhuff_dec() are defined in Annex A.1.

Table 30 — Syntax of SymmetryData()

Syntax	No. of bits	Mnemonic
<pre> SymmetryData(aDataPair) { sumVal = aDataPair[0] + aDataPair[1]; diffVal = aDataPair[0] - aDataPair[1]; if (sumVal > lav) { aDataPair[0] = (2*lav+1) - sumVal; aDataPair[1] = - diffVal; } else { aDataPair[0] = sumVal; aDataPair[1] = diffVal; } if (aDataPair[0] + aDataPair[1] != 0) { bsSymBit[0]; if (bsSymBit[0]) { aDataPair[0] = - aDataPair[0]; aDataPair[1] = - aDataPair[1]; } } if (aDataPair[0] - aDataPair[1] != 0) { bsSymBit[1]; if (bsSymBit[1]) { tmpVal = aDataPair[0]; aDataPair[0] = aDataPair[1]; aDataPair[1] = tmpVal; } } } </pre>	1 1	uimsbf uimsbf

Table 31 — Syntax of LsbData()

Syntax	No. of bits	Mnemonic
<pre> LsbData(dataType, coarseFlag, dataBands) { for(i=0; i<dataBands; i++) { bsLsb = 0; if (dataType == CPC && !coarseFlag) { bsLsb; } aDataOut[i] =bsLsb; } return (aDataOut); } </pre>	1	uimsbf

Table 32 — Syntax of SpatialExtensionFrame()

Syntax	No. of bits	Mnemonic
<pre> SpatialExtensionFrame() { for (ec=0; ec<sacExtNum; ec++) { if (sacExtType[ec]<12) { cnt = bsSacExtLen; if (cnt==255) { cnt += bsSacExtLenAdd; } bitsRead = SpatialExtensionFrameData(sacExtType[ec]) nFillBits = 8*cnt-bitsRead; bsFillBits; } } } </pre>	8 16 nFillBits	uimbsbf uimbsbf Note 1 bslbf
Note 1: SpatialExtensionFrameData() returns the number of bits read.		

Table 33 — Syntax of SpatialExtensionFrameData(0)

Syntax	No. of bits	Mnemonic
SpatialExtensionFrameData(0)		
{		
ResidualData();		
}		

Table 34 — Syntax of ResidualData()

Syntax	No. of bits	Mnemonic
<pre> ResidualData() { for (i=0; i<numOttBoxes+numTttBoxes; i++) { if (bsResidualPresent[i]) { if (i<numOttBoxes) { for (ps=0; ps<numParamSets; ps++) { bslccDiffPresent[i][ps]; if (bslccDiffPresent[i][ps]) { for (pb=0; pb<bsResidualBands[i]; pb++) { lccDiff[i][ps][pb] = 1Dhuff_dec(hcod1D_ICC_Diff,bsCodeW); } } } } } tempExtraFrame=numSlots/(bsResidualFramesPerSpatialFrame+1); for (rf=0; rf<bsResidualFramesPerSpatialFrame; rf++) individual_channel_stream(0); if (window_sequence == EIGHT_SHORT_SEQUENCE) && ((tempExtraFrame == 18) (tempExtraFrame == 24) (tempExtraFrame == 30)) { individual_channel_stream(0); } } } </pre>	1 1..7	Note 2 Uimbsf vlc1bf Note 3 Note 4 Note 1 Note 5 Note 1
Note 1: individual_channel_stream(0) according to MPEG-2 AAC Low Complexity profile bitstream syntax described in subclause 6.3 of ISO/IEC 13818-7. Note 2: numParamSets is defined by numParamSets = bsNumParamSets + 1. Note 3: 1Dhuff_dec() is defined in Annex A.1. Note 4: numSlots is defined by numSlots = bsFrameLength + 1. Furthermore the division shall be interpreted as ANSI C integer division. Note 5: individual_channel_stream(0) determines the value of window_sequence.		

Table 35 — Syntax of SpatialExtensionFrameData(1)

Syntax	No. of bits	Mnemonic
<pre> SpatialExtensionDataFrame(1) { ArbitraryDownmixResidualData(); } </pre>		

Table 36 — Syntax of ArbitraryDownmixResidualData()

[illegible]

Table 37 — Syntax of SpatialExtensionFrameData(2)

Syntax	No. of bits	Mnemonic
SpatialExtensionFrameData(2)		
{		
ArbitraryTreeData();		
}		

Table 38 — Syntax of ArbitraryTreeData()

Syntax	No. of bits	Mnemonic
<pre> ArbitraryTreeData() { for (i=0; i<numOttBoxesAT; i++) { EcData(ATD, i, 0, bsOttBandsAT[i]); } } </pre>		Note 1
Note 1: numOttBoxesAT is defined by TreeConfig().		

5.2 Definition

ByteAlign() Up to 7 fill bits to achieve byte alignment with respect to the beginning of the syntactic element in which ByteAlign() occurs.

SpatialSpecificConfig() Syntactic element that contains the SAC configuration data (header).

bsSamplingFrequencyIndex

see ISO/IEC 14496-3, subclause 1.6.3.4.

bsSamplingFrequency see ISO/IEC 14496-3, subclause 1.6.3.3.

bsFrameLength Defines the number of time slots in a spatial frame according to:
 $\text{numSlots} = \text{bsFrameLength} + 1$.

bsFreqRes Defines the number of parameter bands according to:

Table 39 — bsFreqRes

bsFreqRes	numBands
0	Reserved
1	28
2	20
3	14
4	10
5	7
6	5
7	4

bsTreeConfig Defines the tree configuration according to:

Table 40 — bsTreeConfig

bsTreeConfig	Meaning
0	5151 configuration numOttBoxes = 5 defaultCld[0] = 1 defaultCld[1] = 1 defaultCld[2] = 0 defaultCld[3] = 0 defaultCld[4] = 1 defaultCld[5] = 0 ottModelFe[0] = 0 ottModelFe[1] = 0 ottModelFe[2] = 0 ottModelFe[3] = 0 ottModelFe[4] = 1 numTttBoxes = 0 numInChan = 1 numOutChan = 6 output channel ordering: L, R, C, LFE, Ls, Rs
1	5152 configuration numOttBoxes = 5 defaultCld[0] = 1 defaultCld[1] = 0 defaultCld[2] = 1 defaultCld[3] = 1 defaultCld[4] = 1 defaultCld[5] = 0 ottModelFe[0] = 0 ottModelFe[1] = 0 ottModelFe[2] = 1 ottModelFe[3] = 0 ottModelFe[4] = 0 numTttBoxes = 0 numInChan = 1 numOutChan = 6 output channel ordering: L, Ls, R, Rs, C, LFE
2	525 configuration numOttBoxes = 3 defaultCld[0] = 1 defaultCld[1] = 1 defaultCld[2] = 1 defaultCld[3] = 1 defaultCld[4] = 0 defaultCld[5] = 1 defaultCld[6] = 0 defaultCld[7] = 0 defaultCld[8] = 0 ottModelFe[0] = 1 ottModelFe[1] = 0 ottModelFe[2] = 0 numTttBoxes = 1 numInChan = 2 numOutChan = 6 output channel ordering: L, Ls, R, Rs, C, LFE

3	7271 configuration (5/2.1) numOttBoxes = 5 defaultCld[0] = 1 defaultCld[1] = 1 defaultCld[2] = 1 defaultCld[3] = 1 defaultCld[4] = 1 defaultCld[5] = 1 defaultCld[6] = 0 defaultCld[7] = 1 defaultCld[8] = 0 defaultCld[9] = 0 defaultCld[10] = 0 ottModeLfe[0] = 1 ottModeLfe[1] = 0 ottModeLfe[2] = 0 ottModeLfe[3] = 0 ottModeLfe[4] = 0 numTttBoxes = 1 numInChan = 2 numOutChan = 8 output channel ordering: L, Lc, Ls, R, Rc, Rs, C, LFE
4	7272 configuration (3/4.1) numOttBoxes = 5 defaultCld[0] = 1 defaultCld[1] = 1 defaultCld[2] = 1 defaultCld[3] = 1 defaultCld[4] = 1 defaultCld[5] = 1 defaultCld[6] = 0 defaultCld[7] = 1 defaultCld[8] = 0 defaultCld[9] = 0 defaultCld[10] = 0 ottModeLfe[0] = 1 ottModeLfe[1] = 0 ottModeLfe[2] = 0 ottModeLfe[3] = 0 ottModeLfe[4] = 0 numTttBoxes = 1 numInChan = 2 numOutChan = 8 output channel ordering: L, Lsr, Ls, R, Rsr, Rs, C, LFE

5	7571 configuration (5/2.1) numOttBoxes = 2 defaultCld[0] = 1 defaultCld[1] = 1 defaultCld[2] = 0 defaultCld[3] = 0 defaultCld[4] = 0 defaultCld[5] = 0 defaultCld[6] = 0 defaultCld[7] = 0 ottModelFe[0] = 0 ottModelFe[1] = 0 numTttBoxes = 0 numInChan = 6 numOutChan = 8 output channel ordering: L, Lc, Ls, R, Rc, Rs, C, LFE
6	7572 configuration (3/4.1) numOttBoxes = 2 defaultCld[0] = 1 defaultCld[1] = 1 defaultCld[2] = 0 defaultCld[3] = 0 defaultCld[4] = 0 defaultCld[5] = 0 defaultCld[6] = 0 defaultCld[7] = 0 ottModelFe[0] = 0 ottModelFe[1] = 0 numTttBoxes = 0 numInChan = 6 numOutChan = 8 output channel ordering: L, Lsr, Ls, R, Rsr, Rs, C, LFE
7...15	Reserved

bsQuantMode

Defines quantization and CLD energy-dependent quantization (EdQ) according to:

Table 41 — bsQuantMode

bsQuantMode	Meaning
0	default fine quantization for CLD, ICC, CPC
1	as above, but with first CLD EdQ curve
2	as above, but with second CLD EdQ curve
3	Reserved

bsOneIcc

Indicates if only a single left/right ICC parameter is conveyed common to all OTT boxes.

bsArbitraryDownmix

Indicates the presence of arbitrary down-mix gains according to Table 42.

Table 42 — bsArbitraryDownmix

bsArbitraryDownmix	Arbitrary down-mix gains
0	Not present
1	Present

bsFixedGainsSur Defines the gains used for the surround channels according to:

Table 43 — bsFixedGainsSur

bsFixedGainSur	Surround gain
0	1
1	$2^{(1/4)}$
2	$2^{(2/4)}$
3	$2^{(3/4)}$
4	$2^{(4/4)}$
5..7	reserved

bsFixedGainsLFE Defines the gains used for the LFE channels according to:

Table 44 — bsFixedGainsLFE

bsFixedGainLFE	LFE gain
0	1
1	$10^{(1/2)}$
2	$10^{(2/2)}$
3	$10^{(3/2)}$
4	$10^{(4/2)}$
5..7	reserved

bsFixedGainsDMX Defines the gains used for the downmix according to:

Table 45 — bsFixedGainsDMX

bsFixedGainDMX	Downmix gain
0	1
1	$2^{(1/4)}$
2	$2^{(2/4)}$
3	$2^{(3/4)}$
4	$2^{(4/4)}$
5	$2^{(5/4)}$
6	$2^{(6/4)}$
7	$2^{(8/4)}$

bsMatrixMode Indicates if a matrix compatible stereo downmix has been generated in the encoder according to:

Table 46 — bsMatrixMode

bsMatrixMode	Meaning
0	normal downmix
1	matrix compatible stereo downmix

bsTempShapeConfig Indicates operation mode of temporal shaping (STP or GES) in the decoder according to:

Table 47 — bsTempShapeConfig

bsTempShapeConfig	Meaning
0	do not apply temporal shaping
1	apply STP
2	apply GES
3	reserved

bsDecorrConfig Indicates operation mode of the decorrelator in the decoder according to:

Table 48 — bsDecorrConfig

bsDecorrConfig	Meaning
0	QMF split frequencies: 3, 15, 24, 65
1	QMF split frequencies: 3, 50, 65, 65
2	QMF split frequencies: 0, 15, 65, 65
3	Reserved

bs3AudioMode Indicates that the stereo downmix was 3D audio encoded and that inverse HRTF processing is to be applied.

bsEnvQuantMode Defines the quantization mode of the envelope shaping data according to:

Table 49 — bsEnvQuantMode

bsEnvQuantMode	Meaning
0	5-step quantizer
1	reserved

bs3AudioHRTFset Indicates the set of HRTF parameters according to the following table:

Table 50 — bs3AudioHRTFset

bs3AudioHRTFset	Meaning
0	set of HRTF parameters defined by ParamHRTFset()
1	reserved
2	reserved
3	reserved

OttConfig() Syntactic element that contains the configuration of an OTT box.

bsOttBands Defines the number of parameter bands $0 \leq pb < \text{bsOttBands}$ for which OTT information is present.

TttConfig() Syntactic element that contains the configuration of a TTT box.

bsTttDualMode Indicates if the TTT box operates in different modes for a low and high band range according to:

Table 51 — bsTttDualMode

bsTttDualMode	Meaning
0	same TTT mode for full band range (i.e. no separate high band range)
1	different TTT modes for low and high band ranges