
**Information technology — Generic coding
of moving pictures and associated audio
information —**

**Part 7:
Advanced Audio Coding (AAC)**

*Technologies de l'information — Codage générique des images
animées et du son associé —*

Partie 7: Codage du son avancé (AAC)

Reference number
ISO/IEC 13818-7:2006(E)

Documents
Document Sharing Hub
© ISO/IEC 2006



Not for Resale

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 2006

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
1.1 General.....	1
1.2 MPEG-2 AAC Tools Overview.....	1
2 Normative References	7
3 Terms and Definitions	7
4 Symbols and Abbreviations	14
4.1 Arithmetic Operators	14
4.2 Logical Operators	15
4.3 Relational Operators	15
4.4 Bitwise Operators	16
4.5 Assignment	16
4.6 Mnemonics	16
4.7 Constants	16
5 Method of Describing Bitstream Syntax	16
6 Syntax	18
6.1 Audio Data Interchange Format, ADIF.....	18
6.2 Audio Data Transport Stream, ADTS	19
6.3 Raw Data.....	21
7 Profiles and Profile Interoperability	33
7.1 Profiles	33
7.2 Profile Interoperability.....	35
8 Overall Data Structure	36
8.1 AAC Interchange Formats	36
8.2 Raw Data.....	41
8.3 Single Channel Element (SCE), Channel Pair Element (CPE) and Individual Channel Stream (ICS)	45
8.4 Low Frequency Enhancement Channel (LFE)	51
8.5 Program Config Element (PCE).....	51
8.6 Data Stream Element (DSE)	56
8.7 Fill Element (FIL).....	56
8.8 Extension Payload	57
8.9 Tables.....	61
8.10 Figures	70
9 Noiseless Coding.....	70
9.1 Tool Description.....	70
9.2 Definitions	71
9.3 Decoding Process.....	73
9.4 Tables.....	76
10 Quantization	76
10.1 Tool Description.....	76
10.2 Definitions	76
10.3 Decoding Process.....	76
11 Scalefactors.....	77
11.1 Tool Description.....	77

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.

ISO/IEC 13818-7 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This fourth edition cancels and replaces the third edition (ISO 13818-7:2004), which has been technically revised. It also incorporates the Technical Corrigendum ISO/IEC 13818-7:2004/Cor.1:2005.

ISO/IEC 13818 consists of the following parts, under the general title *Information technology — Generic coding of moving pictures and associated audio information*:

- *Part 1: Systems*
- *Part 2: Video*
- *Part 3: Audio*
- *Part 4: Conformance testing*
- *Part 5: Software simulation* [Technical Report]
- *Part 6: Extensions for DSM-CC*
- *Part 7: Advanced Audio Coding (AAC)*
- *Part 9: Extension for real time interface for systems decoders*
- *Part 10: Conformance extensions for Digital Storage Media Command and Control (DSM-CC)*
- *Part 11: IPMP on MPEG-2 systems*



Introduction

The standardization body ISO/IEC JTC 1/SC 29/WG 11, also known as the Moving Pictures Experts Group (MPEG), was established in 1988 to specify digital video and audio coding schemes at low data rates. MPEG completed its first phase of audio specifications (MPEG-1) in November 1992, ISO/IEC 11172-3. In its second phase of development, the MPEG Audio subgroup defined a multichannel extension to MPEG-1 audio that is backwards compatible with existing MPEG-1 systems (MPEG-2 BC) and defined an audio coding standard at lower sampling frequencies than MPEG-1, ISO/IEC 13818-3.

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

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 the 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 D.

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 D. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Information technology — Generic coding of moving pictures and associated audio information —

Part 7: Advanced Audio Coding (AAC)

1 Scope

1.1 General

This International Standard describes the MPEG-2 audio non-backwards compatible standard called MPEG-2 Advanced Audio Coding, AAC [1], a higher quality multichannel standard than achievable while requiring MPEG-1 backwards compatibility. This MPEG-2 AAC audio standard allows for ITU-R “indistinguishable” quality according to [2] at data rates of 320 kbit/s for five full-bandwidth channel audio signals.

The AAC decoding process makes use of a number of required tools and a number of optional tools. Table 1 lists the tools and their status as required or optional. Required tools are mandatory in any possible profile. Optional tools may not be required in some profiles.

Table 1 — AAC decoder tools

Tool Name	Required / Optional
Bitstream Formatter	Required
Noiseless Decoding	Required
Inverse quantization	Required
Rescaling	Required
M/S	Optional
Prediction	Optional
Intensity	Optional
Dependently switched coupling	Optional
TNS	Optional
Filterbank / block switching	Required
Gain control	Optional
Independently switched coupling	Optional

1.2 MPEG-2 AAC Tools Overview

The basic structure of the MPEG-2 AAC system is shown in Figure 1 and Figure 2. As is shown in Table 1, there are both required and optional tools in the decoder. The data flow in this diagram is from left to right, top to bottom. The functions of the decoder are to find the description of the quantized audio spectra in the bitstream, decode the quantized values and other reconstruction information, reconstruct the quantized spectra, process the reconstructed spectra through whatever tools are active in the bitstream in order to arrive at the actual signal spectra as described by the input bitstream, and finally convert the frequency domain spectra to the time domain, with or without an optional gain control tool. Following the initial reconstruction and scaling of the spectrum reconstruction, there are many optional tools that modify one or more of the spectra in order to provide more efficient coding. For each of the optional tools that operate in the spectral domain, the option to “pass through” is retained, and in all cases where a spectral operation is omitted, the spectra at its input are passed directly through the tool without modification.

The input to the bitstream demultiplexer tool is the MPEG-2 AAC bitstream. The demultiplexer separates the parts of the MPEG-AAC data stream into the parts for each tool, and provides each of the tools with the bitstream information related to that tool.

The outputs from the bitstream demultiplexer tool are:

- The sectioning information for the noiselessly coded spectra,
- The noiselessly coded spectra,
- The M/S decision information (optional),
- The predictor state information (optional),
- The intensity stereo control information and coupling channel control information (both optional),
- The temporal noise shaping (TNS) information (optional),
- The filterbank control information, and
- The gain control information (optional).

The noiseless decoding tool takes information from the bitstream demultiplexer, parses that information, decodes the Huffman coded data, and reconstructs the quantized spectra and the Huffman and DPCM coded scalefactors.

The inputs to the noiseless decoding tool are:

- The sectioning information for the noiselessly coded spectra, and
- The noiselessly coded spectra.

The outputs of the Noiseless Decoding tool are:

- The decoded integer representation of the scalefactors, and
- The quantized values for the spectra.

The inverse quantizer tool takes the quantized values for the spectra, and converts the integer values to the non-scaled, reconstructed spectra. This quantizer is a non-uniform quantizer.

The input to the Inverse Quantizer tool is:

- The quantized values for the spectra.

The output of the inverse quantizer tool is:

- The un-scaled, inversely quantized spectra.

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

The inputs to the rescaling tool are:

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

The output from the scalefactors tool is:

- The scaled, inversely quantized spectra.

The M/S tool converts spectra pairs from Mid/Side to Left/Right under control of the M/S decision information in order to improve coding efficiency.

The inputs to the M/S tool are:

- The M/S decision information, and
- The scaled, inversely quantized spectra related to pairs of channels.

The output from the M/S tool is:

- The scaled, inversely quantized spectra related to pairs of channels, after M/S decoding.

Note The scaled, inversely quantized spectra of individually coded channels are not processed by the M/S block, rather they are passed directly through the block without modification. If the M/S block is not active, all spectra are passed through this block unmodified.

The prediction tool reverses the prediction process carried out at the encoder. This prediction process re-inserts the redundancy that was extracted by the prediction tool at the encoder, under the control of the predictor state information. This tool is implemented as a second order backward adaptive predictor. The inputs to the prediction tool are:

- The predictor state information, and
- The scaled, inversely quantized spectra.

The output from the prediction tool is:

- The scaled, inversely quantized spectra, after prediction is applied.

Note If the prediction is disabled, the scaled, inversely quantized spectra are passed directly through the block without modification.

The intensity stereo tool implements intensity stereo decoding on pairs of spectra.

The inputs to the intensity stereo tool are:

- The inversely quantized spectra, and
- The intensity stereo control information.

The output from the intensity stereo tool is:

- The inversely quantized spectra after intensity channel decoding.

Note The scaled, inversely quantized spectra of individually coded channels are passed directly through this tool without modification, if intensity stereo is not indicated. The intensity stereo tool and M/S tool are arranged so that the operation of M/S and intensity stereo are mutually exclusive on any given scalefactor band and group of one pair of spectra.

The coupling tool for dependently switched coupling channels adds the relevant data from dependently switched coupling channels to the spectra, as directed by the coupling control information.

The inputs to the coupling tool are:

- The inversely quantized spectra, and
- The coupling control information.

The output from the coupling tool is:

- The inversely quantized spectra coupled with the dependently switched coupling channels.

Note The scaled, inversely quantized spectra are passed directly through this tool without modification, if coupling is not indicated. Depending on the coupling control information, dependently switched coupling channels might either be coupled before or after the TNS processing.

The coupling tool for independently switched coupling channels adds the relevant data from independently switched coupling channels to the time signal, as directed by the coupling control information.

The inputs to the coupling tool are:

- The time signal as output by the filterbank, and
- The coupling control information.

The output from the coupling tool is:

- The time signal coupled with the independently switched coupling channels.

Note The time signal is passed directly through this tool without modification, if coupling is not indicated.

The temporal noise shaping (TNS) tool implements a control of the fine time structure of the coding noise. In the encoder, the TNS process has flattened the temporal envelope of the signal to which it has been applied. In the decoder, the inverse process is used to restore the actual temporal envelope(s), under control of the TNS information. This is done by applying a filtering process to parts of the spectral data.

The inputs to the TNS tool are:

- The inversely quantized spectra, and
- The TNS information.

The output from the TNS block is:

- The inversely quantized spectra.

Note If this block is disabled, the inversely quantized spectra are passed through without modification.

The filterbank / block switching tool applies the inverse of the frequency mapping that was carried out in the encoder. An inverse modified discrete cosine transform (IMDCT) is used for the filterbank tool. The IMDCT can be configured to support either one set of 128 or 1024, or four sets of 32 or 256 spectral coefficients.

The inputs to the filterbank tool are:

- The inversely quantized spectra, and
- The filterbank control information.

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

- The time domain reconstructed audio signal(s).

When present, the gain control tool applies a separate time domain gain control to each of four frequency bands that have been created by the gain control PQF filterbank in the encoder. Then, it assembles four frequency bands and reconstructs the time waveform through the gain control tool's filterbank.

The inputs to the gain control tool are:

- The time domain reconstructed audio signal(s), and
- The gain control information.

The output(s) from the gain control tool is (are):

- The time domain reconstructed audio signal(s).

If the gain control tool is not active, the time domain reconstructed audio signal(s) are passed directly from the filterbank tool to the output of the decoder. This tool is used for the scalable sampling rate (SSR) profile only.

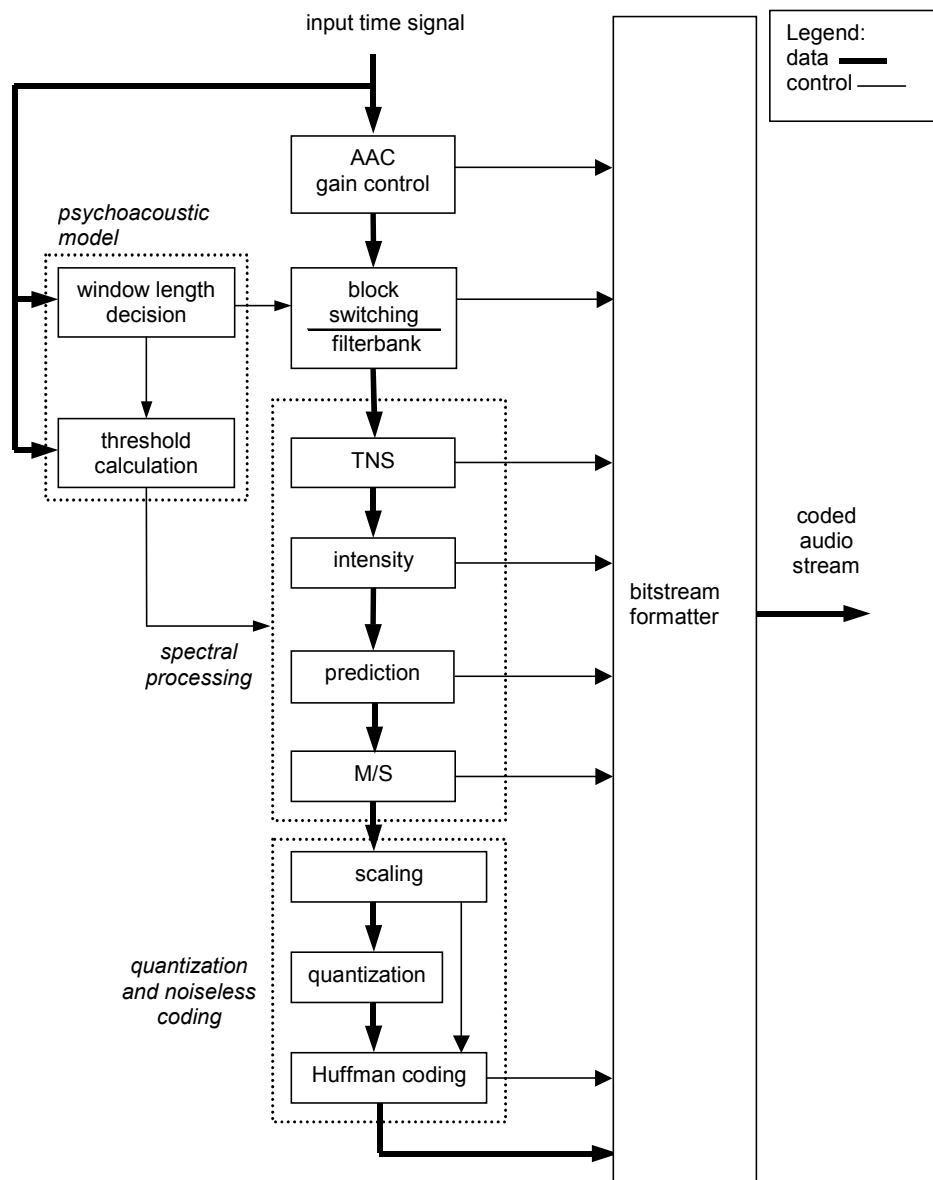


Figure 1 — MPEG-2 AAC Encoder Block Diagram

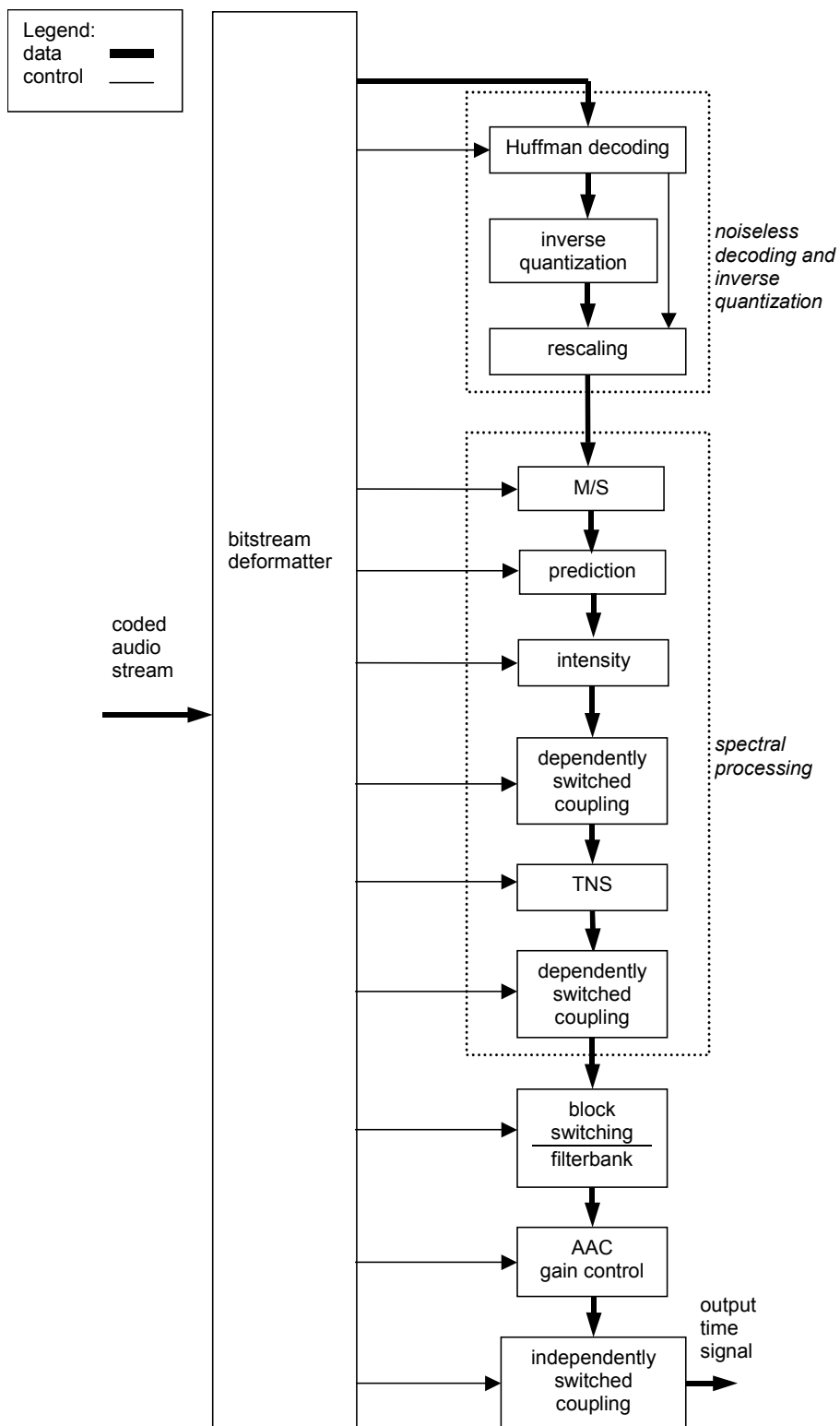


Figure 2 — MPEG-2 AAC Decoder Block Diagram

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 11172-3: *Information technology — Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s — Part 3: Audio*

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

ISO/IEC 13818-3: *Information technology — Generic coding of moving pictures and associated audio information — Part 3: Audio*

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

3 Terms and Definitions

For the purposes of this part of ISO/IEC 13818, the following definitions apply.

3.1

access unit

in the case of compressed audio, an audio access unit

3.2

alias

mirrored signal component resulting from sampling

3.3

analysis filterbank

filterbank in the encoder that transforms a broadband PCM audio signal into a set of spectral coefficients

3.4

ancillary data

part of the bitstream that might be used for transmission of ancillary data

3.5

audio access unit

for AAC, the smallest part of the encoded bitstream which can be decoded by itself, where decoded means "fully reconstructed sound"

NOTE Typically, this is a segment of the encoded bitstream starting after the end of the byte containing the last bit of one ID_END id_syn_ele() through the end of the byte containing the last bit of the next ID_END id_syn_ele.

3.6

audio buffer

buffer in the system target decoder (see ISO/IEC 13818-1) for storage of compressed audio data

3.7

bark

standard unit corresponding to one critical band width of human hearing

3.8

backward compatibility

newer coding standard is backward compatible with an older coding standard if decoders designed to operate with the older coding standard are able to continue to operate by decoding all or part of a bitstream produced according to the newer coding standard

3.9

bitrate

rate at which the compressed bitstream is delivered to the input of a decoder

3.10

bitstream

stream

ordered series of bits that forms the coded representation of the data

3.11

bitstream verifier

process by which it is possible to test and verify that all the requirements specified in this part of ISO/IEC 13818 are met by the bitstream

3.12

block companding

normalizing of the digital representation of an audio signal within a certain time period

3.13

byte aligned

bit in a coded bitstream is byte-aligned if its position is a multiple of 8-bits from either the first bit in the stream for the Audio Data Interchange Format (see 6.1) or the first bit in the syncword for the Audio Data Transport Stream Format (see 6.2)

3.14

byte

sequence of 8 bits

3.15

centre channel

audio presentation channel used to stabilize the central component of the frontal stereo image

3.16

channel

sequence of data representing an audio signal intended to be reproduced at one listening position

3.17

coded audio bitstream

coded representation of an audio signal

3.18

coded representation

data element as represented in its encoded form

3.19

compression

reduction in the number of bits used to represent an item of data

3.20

constant bitrate

operation in which the bitrate is constant from start to finish of the coded bitstream

3.21

CRC

Cyclic Redundancy Check to verify the correctness of data

3.22**critical band**

unit of bandwidth which represents the standard unit of bandwidth expressed in human auditory terms, corresponding to a fixed length on the human cochlea, approximately equal to 100 Hz at low frequencies and 1/3 octave at higher frequencies, above approximately 700 Hz

3.23**data element**

item of data as represented before encoding and after decoding

3.24**decoded stream**

decoded reconstruction of a compressed bitstream

3.25**decoder**

embodiment of a decoding process

3.26**decoding (process)**

process defined in this part of ISO/IEC 13818 that reads an input coded bitstream and outputs decoded audio samples

3.27**digital storage media****DSM**

digital storage or transmission device or system

3.28**discrete cosine transform****DCT**

either the forward discrete cosine transform or the inverse discrete cosine transform, an invertible, discrete orthogonal transformation

3.29**downmix**

matrixing of n channels to obtain less than n channels

3.30**editing**

process by which one or more coded bitstreams are manipulated to produce a new coded bitstream

NOTE Conforming edited bitstreams are defined in this part of ISO/IEC 13818.

3.31**encoder**

embodiment of an encoding process

3.32**encoding (process)**

process, not specified in ISO/IEC 13818, that reads a stream of input audio samples and produces a valid coded bitstream as defined in this part of ISO/IEC 13818

3.33**entropy coding**

variable length lossless coding of the digital representation of a signal to reduce statistical redundancy

3.34

Fast Fourier Transformation

FFT

fast algorithm for performing a discrete Fourier transform (an orthogonal transform)

3.35

filterbank

set of band-pass filters covering the entire audio frequency range

3.36

flag

variable which can take one of only the two values defined in this part of ISO/IEC 13818

3.37

forward compatibility

a newer coding standard is forward compatible with an older coding standard if decoders designed to operate with the newer coding standard are able to decode bitstreams of the older coding standard

3.38

frame

part of the audio signal that corresponds to audio PCM samples from an audio access unit

3.39

F_s

sampling frequency

3.40

Hann window

time function applied sample-by-sample to a block of audio samples before Fourier transformation

3.41

Huffman coding

specific method for entropy coding

3.42

hybrid filterbank

serial combination of subband filterbank and MDCT

3.43

IDCT

Inverse Discrete Cosine Transform

3.44

IMDCT

Inverse Modified Discrete Cosine Transform

3.45

intensity stereo

method of exploiting stereo irrelevance or redundancy in stereophonic audio programmes based on retaining at high frequencies only the energy envelope of the right and left channels

3.46

joint stereo coding

any method that exploits stereophonic irrelevance or stereophonic redundancy

3.47

joint stereo mode

mode of the audio coding algorithm using joint stereo coding

3.48**low frequency enhancement (LFE) channel**

limited bandwidth channel for low frequency audio effects in a multichannel system

3.49**main audio channels**

all channels represented by either `single_channel_element()`'s (see 8.2.1) or `channel_pair_element()`'s (see 8.2.1)

3.50**mapping**

conversion of an audio signal from time to frequency domain by subband filtering and/or by MDCT

3.51**masking**

property of the human auditory system by which an audio signal cannot be perceived in the presence of another audio signal

3.52**masking threshold**

function in frequency and time below which an audio signal cannot be perceived by the human auditory system

3.53**modified discrete cosine transform****MDCT**

transform which has the property of time domain aliasing cancellation

NOTE An analytical expression for the MDCT can be found in C.3.1.2.

3.54**M/S stereo**

method of removing imaging artefacts as well as exploiting stereo irrelevance or redundancy in stereophonic audio programmes based on coding the sum and difference signal instead of the left and right channels

3.55**multichannel**

combination of audio channels used to create a spatial sound field

3.56**multilingual**

presentation of dialogue in more than one language

3.57**non-tonal component**

noise-like component of an audio signal

3.58**Number of Considered Channels****NCC**

number of channels represented by the elements SCE, independently switched CCE and CPE, i.e. once the number of SCEs plus once the number of independently switched CCEs plus twice the number of CPEs, with respect to the naming conventions of the MPEG-AAC decoders and bitstreams, $NCC=A+I$

NOTE This number is used to derive the required decoder input buffer size (see 8.2.3).

3.59**Nyquist sampling**

sampling at or above twice the maximum bandwidth of a signal

3.60

padding

method to adjust the average length of an audio frame in time to the duration of the corresponding PCM samples, by conditionally adding a slot to the audio frame

3.61

parameter

variable within the syntax of this specification which may take one of a range of values. A variable which can take one of only two values is a flag or indicator and not a parameter

3.62

parser

functional stage of a decoder which extracts from a coded bitstream a series of bits representing coded elements

3.63

polyphase filterbank

set of equal bandwidth filters with special phase interrelationships, allowing for an efficient implementation of the filterbank

3.64

prediction error

difference between the actual value of a sample or data element and its predictor

3.65

prediction

use of a predictor to provide an estimate of the sample value or data element currently being decoded

3.66

predictor

linear combination of previously decoded sample values or data elements

3.67

presentation channel

audio channel at the output of the decoder

3.68

presentation unit

in the case of compressed audio, a decoded audio access unit

3.69

program

set of main audio channels, coupling_channel_element()'s (see 8.2.1), lfe_channel_element()'s (see 8.2.1), and associated data streams intended to be decoded and played back simultaneously

NOTE A program may be defined by default (see 8.5.3.1 and 8.5.3.3) or specifically by a program_config_element() (see 8.5.3.2). A given single_channel_element() (see 8.2.1), channel_pair_element() (see 8.2.1), coupling_channel_element(), lfe_channel_element() or data channel may accompany one or more programs in any given bitstream.

3.70

psychoacoustic model

mathematical model of the masking behaviour of the human auditory system

3.71

random access

process of beginning to read and decode the coded bitstream at an arbitrary point

3.72**reserved**

when used in the clauses defining the coded bitstream, indicates that the value may be used in the future for ISO/IEC defined extensions

3.73**sampling frequency** **F_s**

rate in Hertz which is used to digitize an audio signal during the sampling process

3.74**scalefactor**

factor by which a set of values is scaled before quantization

3.75**scalefactor band**

set of spectral coefficients which are scaled by one scalefactor

3.76**scalefactor index**

numerical code for a scalefactor

3.77**side information**

information in the bitstream necessary for controlling the decoder

3.78**spectral coefficients**

discrete frequency domain data output from the analysis filterbank

3.79**spreading function**

function that describes the frequency spread of masking effects

3.80**stereo-irrelevant**

portion of a stereophonic audio signal which does not contribute to spatial perception

3.81**stuffing (bits)****stuffing (bytes)**

code words that may be inserted at particular locations in the coded bitstream that are discarded in the decoding process whose purpose is to increase the bitrate of the stream which would otherwise be lower than the desired bitrate

3.82**surround channel**

audio presentation channel added to the front channels (L and R or L, R, and C) to enhance the spatial perception

3.83**Syncword**

a 12-bit code embedded in the audio bitstream that identifies the start of a `adts_frame()` (see 6.2, Table 5)

3.84**synthesis filterbank**

filterbank in the decoder that reconstructs a PCM audio signal from subband samples

3.85

tonal component

sinusoid-like component of an audio signal

3.86

variable bitrate

operation in which the bitrate varies with time during the decoding of a coded bitstream

3.87

variable length coding

reversible procedure for coding that assigns shorter code words to frequent symbols and longer code words to less frequent symbols

3.88

variable length code

VLC

code word assigned by variable length encoder (see variable length coding)

3.89

variable length decoder

procedure to obtain the symbols encoded with a variable length coding technique

3.90

variable length encoder

procedure to assign variable length codewords to symbols

4 Symbols and Abbreviations

The mathematical operators used to describe this International Standard are similar to those used in the C programming language. However, integer division with truncation and rounding are specifically defined. The bitwise operators are defined assuming twos-complement representation of integers. Numbering and counting loops generally begin from zero.

4.1 Arithmetic Operators

+	Addition.
–	Subtraction (as a binary operator) or negation (as a unary operator).
++	Increment.
--	Decrement.
*	Multiplication.
^	Power.
/	Integer division with truncation of the result toward zero. For example, $7/4$ and $-7/4$ are truncated to 1 and $-7/4$ and $7/-4$ are truncated to -1 .
//	Integer division with rounding to the nearest integer. Half-integer values are rounded away from zero unless otherwise specified. For example $3//2$ is rounded to 2, and $-3//2$ is rounded to -2 .
DIV	Integer division with truncation of the result towards $-\infty$.

$|x|$ Absolute value. $|x| = x$ when $x > 0$
 $|x| = 0$ when $x == 0$
 $|x| = -x$ when $x < 0$

% Modulus operator. Defined only for positive numbers.

Sign() Sign.
 $\text{Sign}(x) = 1$ when $x > 0$
 $\text{Sign}(x) = 0$ when $x == 0$
 $\text{Sign}(x) = -1$ when $x < 0$

INT () Truncation to integer operator. Returns the integer part of the real-valued argument.

NINT () Nearest integer operator. Returns the nearest integer value to the real-valued argument. Half-integer values are rounded away from zero.

sin Sine.

cos Cosine.

exp Exponential.

$\sqrt{\quad}$ Square root.

$\log_{10}$ Logarithm to base ten.

$\log_e$ Logarithm to base e.

$\log_2$ Logarithm to base 2.

4.2 Logical Operators

|| Logical OR.

&& Logical AND.

! Logical NOT

4.3 Relational Operators

> Greater than.

>= Greater than or equal to.

< Less than.

<= Less than or equal to.

== Equal to.

!= Not equal to.

max [...], the maximum value in the argument list.

min [...], the minimum value in the argument list.

4.4 Bitwise Operators

A twos complement number representation is assumed where the bitwise operators are used.

&	AND
	OR
>>	Shift right with sign extension.
<<	Shift left with zero fill.

4.5 Assignment

=	Assignment operator.
---	----------------------

4.6 Mnemonics

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

bslbf	Bit string, left bit first, where "left" is the order in which bit strings are written in ISO/IEC 13818. Bit strings are written as a string of 1s and 0s within single quote marks, e.g. '1000 0001'. Blanks within a bit string are for ease of reading and have no significance.
L, C, R, LS, RS	Left, center, right, left surround and right surround audio signals
rpchof	Remainder polynomial coefficients, highest order first. (Audio)
uimsbf	Unsigned integer, most significant bit first.
vlcblf	Variable length code, left bit first, where "left" refers to the order in which the VLC codes are written.
window	Number of the actual time slot in case of block_type == 2, 0 <= window <= 2. (Audio)

The byte order of multi-byte words is most significant byte first.

4.7 Constants

π	3.14159265358...
e	2.71828182845...

5 Method of Describing Bitstream Syntax

The bitstream retrieved by the decoder is described in clause 6. Each data item in the bitstream is in bold type. It is described by

- its name;
- its length in bits, where "X..Y" indicates that the number of bits is one of the values between X and Y including X and Y. "{X;Y}" means the number of bits is X or Y, depending on the value of other data elements in the bitstream;
- a mnemonic for its type and order of transmission.

The action caused by a decoded data element in a bitstream depends on the value of that data element and on data elements previously decoded. The decoding of the data elements and the definition of the state variables used in their decoding are described in the clauses following the syntax clause. The following constructs are used to express the conditions when data elements are present, and are in normal type:

Note this syntax uses the 'C'-code convention that a variable or expression evaluating to a non-zero value is equivalent to a condition that is true.

<pre>while (condition) { data_element; ... }</pre>	If the condition is true, then the group of data elements occurs next in the data stream. This repeats until the condition is not true.
--	--

<pre>do { data_element; ... } while (condition)</pre>	The data element always occurs at least once. The data element is repeated until the condition is not true.
---	--

<pre>if (condition) { data_element; ... } else { data_element; ... }</pre>	If the condition is true, then the first group of data elements occurs next in the data stream If the condition is not true, then the second group of data elements occurs next in the data stream.
--	---

<pre>switch (expression) { case const-expr: data_element; break; case const-expr: data_element; }</pre>	If the condition formed by the comparison of expression and const-expr. is true, then the data stream continues with the subsequent data elements. An optionally break statement can be used to immediately leave the switch, data elements beyond a break do not occur in the data stream.
---	--

<pre>for (expr1; expr2; expr3) { data_element; ... }</pre>	Expr1 is an expression specifying the initialisation of the loop. Normally it specifies the initial state of the counter. Expr2 is a condition specifying a test made before each iteration of the loop. The loop terminates when the condition is not true. Expr3 is an expression that is performed at the end of each iteration of the loop, normally it increments a counter.
--	--

Note that the most common usage of this construct is as follows:

<pre>for (i = 0; i < n; i++) { data_element ... }</pre>	The group of data elements occurs n times. Conditional constructs within the group of data elements may depend on the value of the loop control variable i, which is set to zero for the first occurrence, incremented to one for the second occurrence, and so forth.
--	---

As noted, the group of data elements may contain nested conditional constructs. For compactness, the {} may be omitted when only one data element follows.

data_element []	data_element [] is an array of data. The number of data elements is indicated by the context.
data_element [n]	data_element [n] is the n+1th element of an array of data.
data_element [m][n]	data_element [m][n] is the m+1,n+1 th element of a two-dimensional array of data.
data_element [l][m][n]	data_element [l][m][n] is the l+1,m+1,n+1 th element of a three-dimensional array of data.
data_element [m..n]	data_element [m..n] is the inclusive range of bits between bit m and bit n in the data_element.

While the syntax is expressed in procedural terms, it should not be assumed that clause 6 implements a satisfactory decoding procedure. In particular, it defines a correct and error-free input bitstream. Actual decoders must include a means to look for start codes in order to begin decoding correctly.

Definition of nextbits function

The function nextbits() permits comparison of a bit string with the next bits to be decoded in the bitstream.

6 Syntax

6.1 Audio Data Interchange Format, ADIF

Table 2 — Syntax of adif_sequence()

Syntax	No. of bits	Mnemonic
adif_sequence() { adif_header(); byte_alignment(); raw_data_stream(); }		

Table 3 — Syntax of `adif_header()`

Syntax	No. of bits	Mnemonic
<code>adif_header()</code>		
{		
adif_id;	32	bslbf
copyright_id_present;	1	bslbf
if (copyright_id_present) {		
copyright_id;	72	bslbf
}		
original_copy;	1	bslbf
home;	1	bslbf
bitstream_type;	1	bslbf
bitrate;	23	uimsbf
num_program_config_elements;	4	bslbf
if (bitstream_type == '0') {		
adif_buffer_fullness;	20	uimsbf
}		
for (i = 0; i < num_program_config_elements + 1; i++) {		
program_config_element();		
}		
}		

6.2 Audio Data Transport Stream, ADTS

Table 4 — Syntax of `adts_sequence()`

Syntax	No. of bits	Mnemonic
<code>adts_sequence()</code>		
{		
while (nextbits() == syncword) {		
adts_frame();		
}		
}		

Table 5 — Syntax of `adts_frame()`

Syntax	No. of bits	Mnemonic
<code>adts_frame()</code>		
{		
adts_fixed_header();		
adts_variable_header();		
if (number_of_raw_data_blocks_in_frame == 0) {		
adts_error_check();		
raw_data_block();		
}		
else {		
adts_header_error_check();		
for (i = 0; i <= number_of_raw_data_blocks_in_frame;		
i++) {		
raw_data_block();		
adts_raw_data_block_error_check();		
}		
}		
}		

Table 6 — Syntax of adts_header_error_check()

Syntax	No. of bits	Mnemonic
adts_header_error_check () { if (protection_absent == '0') { for (i = 1; i <= number_of_raw_data_blocks_in_frame; i++) { raw_data_block_position[i]; } crc_check; } }	16 16	uimsfb rpchof

Table 7 — Syntax of adts_raw_data_block_error_check()

Syntax	No. of bits	Mnemonic
adts_raw_data_block_error_check() { if (protection_absent == '0') crc_check; }	16	rpchof

6.2.1 Fixed Header of ADTS

Table 8 — Syntax of adts_fixed_header()

Syntax	No. of bits	Mnemonic
adts_fixed_header() { syncword; ID; layer; protection_absent; profile; sampling_frequency_index; private_bit; channel_configuration; original_copy; home; }	12 1 2 1 2 4 1 3 1 1	bslbf bslbf uimsbf bslbf uimsbf uimsbf bslbf uimsbf bslbf bslbf

6.2.2 Variable Header of ADTS

Table 9 — Syntax of adts_variable_header()

Syntax	No. of bits	Mnemonic
adts_variable_header() { copyright_identification_bit; copyright_identification_start; aac_frame_length; adts_buffer_fullness; number_of_raw_data_blocks_in_frame; }	1 1 13 11 2	bslbf bslbf bslbf bslbf uimsfb

6.2.3 Error Detection

Table 10 — Syntax of adts_error_check()

Syntax	No. of bits	Mnemonic
<pre>adts_error_check() { if (protection_absent == '0') crc_check; }</pre>	16	rpchof

6.3 Raw Data

Table 11 — Syntax of raw_data_stream()

Syntax	No. of bits	Mnemonic
<pre>raw_data_stream() { while (data_available()) { raw_data_block(); } }</pre>		

Table 12 — Syntax of raw_data_block()

Syntax	No. of bits	Mnemonic
<pre>raw_data_block() { while ((id = id_syn_ele) != ID_END) { switch (id) { case ID_SCE: single_channel_element(); break; case ID_CPE: channel_pair_element(); break; case ID_CCE: coupling_channel_element(); break; case ID_LFE: lfe_channel_element(); break; case ID_DSE: data_stream_element(); break; case ID_PCE: program_config_element(); break; case ID_FIL: fill_element(); } } byte_alignment(); }</pre>	3	uimbsf

Table 13 — Syntax of single_channel_element()

Syntax	No. of bits	Mnemonic
single_channel_element() { element_instance_tag ; individual_channel_stream(0); }	4	uimsbf

Table 14 — Syntax of channel_pair_element()

Syntax	No. of bits	Mnemonic
channel_pair_element() { element_instance_tag ; common_window ; if (common_window) { ics_info(); ms_mask_present ; if (ms_mask_present == 1) { for (g = 0; g < num_window_groups; g++) { for (sfb = 0; sfb < max_sfb; sfb++) { ms_used[g][sfb] ; } } } } individual_channel_stream(common_window); individual_channel_stream(common_window); }	4 1 2 1	uimsbf uimsbf uimsbf uimsbf

Table 15 — Syntax of ics_info()

Syntax	No. of bits	Mnemonic
ics_info() {		
ics_reserved_bit;	1	bslbf
window_sequence;	2	uimsbf
window_shape;	1	uimsbf
if (window_sequence == EIGHT_SHORT_SEQUENCE) {		
max_sfb;	4	uimsbf
scale_factor_grouping;	7	uimsbf
}		
else {		
max_sfb;	6	uimsbf
predictor_data_present;	1	uimsbf
if (predictor_data_present) {		
predictor_reset;	1	uimsbf
if (predictor_reset) {		
predictor_reset_group_number;	5	uimsbf
}		
for (sfb = 0; sfb < min(max_sfb, PRED_SFB_MAX); sfb++) {		
prediction_used[sfb];	1	uimsbf
}		
}		
}		
}		

Table 16 — Syntax of individual_channel_stream()

Syntax	No. of bits	Mnemonic
individual_channel_stream(common_window) {		
global_gain;	8	uimsbf
if (!common_window)		
ics_info();		
section_data();		
scale_factor_data();		
pulse_data_present;	1	uimsbf
if (pulse_data_present) {		
pulse_data();		
}		
tns_data_present;	1	uimsbf
if (tns_data_present) {		
tns_data();		
}		
gain_control_data_present;	1	uimsbf
if (gain_control_data_present) {		
gain_control_data();		
}		
spectral_data();		
}		

Table 17 — Syntax of section_data()

Syntax	No. of bits	Mnemonic
<pre> section_data() { if (window_sequence == EIGHT_SHORT_SEQUENCE) sect_esc_val = (1<<3) - 1; else sect_esc_val = (1<<5) - 1; for (g = 0; g < num_window_groups; g++) { k = 0; i = 0; while (k < max_sfb) { sect_cb[g][i]; sect_len = 0; while (sect_len_incr == sect_esc_val) { sect_len += sect_esc_val; } sect_len += sect_len_incr; sect_start[g][i] = k; sect_end[g][i] = k+sect_len; for (sfb = k; sfb < k+sect_len; sfb++) sfb_cb[g][sfb] = sect_cb[g][i]; k += sect_len; i++; } num_sec[g] = i; } } </pre>	4 {3;5}	uimsbf uimsbf

Table 18 — Syntax of scale_factor_data()

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

Table 19 — Syntax of tns_data()

Syntax	No. of bits	Mnemonic
tns_data() { for (w = 0; w < num_windows; w++) { n_filt[w]; if (n_filt[w]) coef_res[w]; for (filt = 0; filt < n_filt[w]; filt++) { length[w][filt]; order[w][filt]; if (order[w][filt]) { direction[w][filt]; coef_compress[w][filt]; for (i = 0; i < order[w][filt]; i++) coef[w][filt][i]; } } } }	1..2 1 {4;6} {3;5} 1 1 2..4	uimbsf uimbsf uimbsf uimbsf uimbsf uimbsf

Table 20 — Syntax of spectral_data()

Syntax	No. of bits	Mnemonic
spectral_data() { for (g = 0; g < num_window_groups; g++) { for (i = 0; i < num_sec[g]; i++) { if (sect_cb[g][i] != ZERO_HCB && sect_cb[g][i] <= ESC_HCB) { for (k = sect_sfb_offset[g][sect_start[g][i]; k < sect_sfb_offset[g][sect_end[g][i]];) { if (sect_cb[g][i] < FIRST_PAIR_HCB) { hcod[sect_cb[g][i]][w][x][y][z]; if (unsigned_cb[sect_cb[g][i]]) quad_sign_bits; k += QUAD_LEN; } else { hcod[sect_cb[g][i]][y][z]; if (unsigned_cb[sect_cb[g][i]]) pair_sign_bits; k += PAIR_LEN; if (sect_cb[g][i] == ESC_HCB) { if (y == ESC_FLAG) hcod_esc_y; if (z == ESC_FLAG) hcod_esc_z; } } } } } } }	1..16 0..4 1..15 0..2 5..21 5..21	vlclbf bslbf vlclbf bslbf vlclbf vlclbf

Table 21 — Syntax of pulse_data()

Syntax	No. of bits	Mnemonic
pulse_data() {		
number_pulse;	2	uimsbf
pulse_start_sfb;	6	uimsbf
for (i = 0; i < number_pulse+1; i++) {		
pulse_offset[i];	5	uimsbf
pulse_amp[i];	4	uimsbf
}		
}		

Table 22 — Syntax of coupling_channel_element()

Syntax	No. of bits	Mnemonic
coupling_channel_element()		
{		
element_instance_tag;	4	uimsbf
ind_sw_cce_flag;	1	uimsbf
num_coupled_elements;	3	uimsbf
num_gain_element_lists = 0;		
for (c = 0; c < num_coupled_elements+1; c++) {		
num_gain_element_lists++;		
cc_target_is_cpe[c];	1	uimsbf
cc_target_tag_select[c];	4	uimsbf
if (cc_target_is_cpe[c]) {		
cc_l[c];	1	uimsbf
cc_r[c];	1	uimsbf
if (cc_l[c] && cc_r[c])		
num_gain_element_lists++;		
}		
}		
cc_domain;	1	uimsbf
gain_element_sign;	1	uimsbf
gain_element_scale;	2	uimsbf
individual_channel_stream(0);		
for (c = 1; c < num_gain_element_lists; c++) {		
if (ind_sw_cce_flag) {		
cge = 1;		
} else {		
common_gain_element_present[c];	1	uimsbf
cge = common_gain_element_present[c];		
}		
if (cge)		
hcod_sf[common_gain_element[c]];	1..19	vlc1bf
else {		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = 0; sfb < max_sfb; sfb++) {		
if (sfb_cb[g][sfb] != ZERO_HCB);		
hcod_sf[dpcm_gain_element[c][g][sfb]];	1..19	vlc1bf
}		
}		
}		
}		
}		

Table 23 — Syntax of lfe_channel_element()

Syntax	No. of bits	Mnemonic
lfe_channel_element() { element_instance_tag ; individual_channel_stream(0); }	4	uimsbf

Table 24 — Syntax of data_stream_element()

Syntax	No. of bits	Mnemonic
data_stream_element() { element_instance_tag ; data_byte_align_flag ; cnt = count ; if (cnt == 255) { cnt += esc_count ; } if (data_byte_align_flag) { byte_alignment(); } for (i = 0; i < cnt; i++) { data_stream_byte [element_instance_tag][i]; } }	4 1 8 8 8	uimsbf uimsbf uimsbf uimsbf uimsbf

Table 25 — Syntax of program_config_element()

Syntax	No. of bits	Mnemonic
program_config_element() {		
element_instance_tag;	4	uimsbf
profile;	2	uimsbf
sampling_frequency_index;	4	uimsbf
num_front_channel_elements;	4	uimsbf
num_side_channel_elements;	4	uimsbf
num_back_channel_elements;	4	uimsbf
num_lfe_channel_elements;	2	uimsbf
num_assoc_data_elements;	3	uimsbf
num_valid_cc_elements;	4	uimsbf
mono_mixdown_present;	1	uimsbf
if (mono_mixdown_present == 1)		
mono_mixdown_element_number;	4	uimsbf
stereo_mixdown_present;	1	uimsbf
if (stereo_mixdown_present == 1)		
stereo_mixdown_element_number;	4	uimsbf
matrix_mixdown_idx_present;	1	uimsbf
if (matrix_mixdown_idx_present == 1) {		
matrix_mixdown_idx ;	2	uimsbf
pseudo_surround_enable;	1	uimsbf
}		
for (i = 0; i < num_front_channel_elements; i++) {		
front_element_is_cpe[i];	1	bslbf
front_element_tag_select[i];	4	uimsbf
}		
for (i = 0; i < num_side_channel_elements; i++) {		
side_element_is_cpe[i];	1	bslbf
side_element_tag_select[i];	4	uimsbf
}		
for (i = 0; i < num_back_channel_elements; i++) {		
back_element_is_cpe[i];	1	bslbf
back_element_tag_select[i];	4	uimsbf
}		
for (i = 0; i < num_lfe_channel_elements; i++)		
lfe_element_tag_select[i];	4	uimsbf
for (i = 0; i < num_assoc_data_elements; i++)		
assoc_data_element_tag_select[i];	4	uimsbf
for (i = 0; i < num_valid_cc_elements; i++) {		
cc_element_is_ind_sw[i];	1	uimsbf
valid_cc_element_tag_select[i];	4	uimsbf
}		
byte_alignment();		
comment_field_bytes;	8	uimsbf
for (i = 0; i < comment_field_bytes; i++)		
comment_field_data[i];	8	uimsbf
}		

Table 26 — Syntax of fill_element()

Syntax	No. of bits	Mnemonic
fill_element() { cnt = count ; if (cnt == 15) cnt += esc_count - 1; while (cnt > 0) { cnt -= extension_payload(cnt); } }	4 8	uimsbf uimsbf

Table 27 — Syntax of gain_control_data()

Syntax	No. of bits	Mnemonic
gain_control_data() {		
max_band;	2	uimsbf
if (window_sequence == ONLY_LONG_SEQUENCE) {		
for (bd = 1; bd <= max_band; bd++) {		
for (wd = 0; wd < 1; wd++) {		
adjust_num[bd][wd];	3	uimsbf
for (ad = 0; ad < adjust_num[bd][wd]; ad++) {		
alevcode[bd][wd][ad];	4	uimsbf
alocode[bd][wd][ad];	5	uimsbf
}		
}		
}		
} else if (window_sequence == LONG_START_SEQUENCE)		
{		
for (bd = 1; bd <= max_band; bd++) {		
for (wd = 0; wd < 2; wd++) {		
adjust_num[bd][wd];	3	uimsbf
for (ad = 0; ad < adjust_num[bd][wd]; ad++) {		
alevcode[bd][wd][ad];	4	uimsbf
if (wd == 0)		
alocode[bd][wd][ad];	4	uimsbf
else		
alocode[bd][wd][ad];	2	uimsbf
}		
}		
}		
} else if (window_sequence ==		
EIGHT_SHORT_SEQUENCE) {		
for (bd = 1; bd <= max_band; bd++) {		
for (wd = 0; wd < 8; wd++) {		
adjust_num[bd][wd];	3	uimsbf
for (ad = 0; ad < adjust_num[bd][wd]; ad++) {		
alevcode[bd][wd][ad];	4	uimsbf
alocode[bd][wd][ad];	2	uimsbf
}		
}		
}		
} else if (window_sequence == LONG_STOP_SEQUENCE) {		
for (bd = 1; bd <= max_band; bd++) {		
for (wd = 0; wd < 2; wd++) {		
adjust_num[bd][wd];	3	uimsbf
for (ad = 0; ad < adjust_num[bd][wd]; ad++) {		
alevcode[bd][wd][ad];	4	uimsbf
if (wd == 0)		
alocode[bd][wd][ad];	4	uimsbf
else		
alocode[bd][wd][ad];	5	uimsbf
}		
}		
}		
}		
}		

Table 28 — Syntax of extension_payload()

extension_payload(cnt)		
{		
extension_type;	4	uimbsf
switch (extension_type) {		
case EXT_DYNAMIC_RANGE:		
n = dynamic_range_info();		
return n;		
case EXT_SBR_DATA:		
return sbr_extension_data(id_aac, 0);		Note 1
case EXT_SBR_DATA_CRC:		
return sbr_extension_data(id_aac, 1);		Note 1
case EXT_FILL_DATA:		
fill_nibble; /* must be '0000' */	4	uimbsf
for (i = 0; i < cnt-1; i++)		
fill_byte[i]; /* must be '10100101' */	8	uimbsf
return cnt;		
case default:		
for (i = 0; i < 8*(cnt-1)+4; i++)		
other_bits[i];	1	uimbsf
return cnt;		
}		
}		

Note 1: id_aac is the id_syn_ele of the corresponding AAC element (ID_SCE or ID_CPE) or ID_SCE in case of CCE.

© ISO/IEC 2006

Table 29 — Syntax of dynamic_range_info()

Syntax	No. of bits	Mnemonic
dynamic_range_info() { n = 1; drc_num_bands = 1; pce_tag_present; if (pce_tag_present == 1) { pce_instance_tag; drc_tag_reserved_bits; n++; } excluded_chns_present; if (excluded_chns_present == 1) { n += excluded_channels(); } drc_bands_present ; if (drc_bands_present == 1) { drc_band_incr; drc_bands_reserved_bits; n++; drc_num_bands = drc_num_bands + drc_band_incr; for (i = 0; i < drc_num_bands; i++) { drc_band_top[i]; n++; } } prog_ref_level_present; if (prog_ref_level_present == 1) { prog_ref_level; prog_ref_level_reserved_bits; n++; } for (i = 0; i < drc_num_bands; i++) { dyn_rng_sgn[i]; dyn_rng_ctl[i]; n++; } return n; }	1 4 4 1 1 4 4 8 1 7 1 1 7	uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf

Table 30 — Syntax of `excluded_channels()`

Syntax	No. Of bits	Mnemonic
<pre> excluded_channels() { n = 0; num_excl_chan = 70; for (i = 0; i < 7; i++) exclude_mask[i]; n++; while (additional_excluded_chns[n-1] == 1) { for (i = num_excl_chan; i < num_excl_chan+7; i++) exclude_mask[i]; n++; num_excl_chan += 7; } return n; } </pre>	1	uimsbf
	1	uimsbf
	1	uimsbf

7 Profiles and Profile Interoperability

7.1 Profiles

There are three profiles identified in the MPEG-2 AAC standard:

Main Profile

Low Complexity Profile

Scalable Sampling Rate Profile

In the `program_config_element()` and `adts_fixed_header()`, a two bit field indicates the profile in use:

Table 31 — Profiles

index	profile
0	Main profile
1	Low Complexity profile (LC)
2	Scalable Sampling Rate profile (SSR)
3	(reserved)

7.1.1 Main

The Main profile is used when memory cost is not significant, and when there is substantial processing power available. With the exception of the gain control tool, all parts of the tools may be used in order to provide the best data compression possible. There shall be only one program (in the sense of what is specified in a `program_config_element()`) in a Main profile bitstream. The program in a Main profile bitstream shall not contain any mono or stereo mixdown elements.

7.1.2 Low Complexity

The Low Complexity profile is used when RAM usage, processing power, and compression requirements are all present. In the low complexity profile, prediction, and gain control tool are not permitted and TNS order is limited. There shall be only one program (in the sense of what is specified in a `program_config_element()`) in a Low Complexity profile bitstream. The program in a Low Complexity profile bitstream shall not contain any mono or stereo mixdown elements.

7.1.3 Scalable Sampling Rate

In the Scalable Sampling Rate profile, the gain-control tool is required. Prediction and coupling channels are not permitted, and TNS order and bandwidth are limited. Gain control is not used in the lowest of the 4 PQF subbands. In the case of a reduced audio bandwidth, the SSR profile will scale accordingly in complexity. There shall be only one program (in the sense of what is specified in a `program_config_element()`) in a Scalable Sampling Rate profile bitstream. The program in a Scalable Sampling Rate profile bitstream shall not contain any mono or stereo mixdown elements.

7.1.4 Naming Convention for MPEG-2 AAC Decoders and Bitstreams

A decoder or bitstream may be specified as an A.L.I.D Channel <Profile Name> Profile MPEG-2 AAC decoder or bitstream, where A is replaced by the number of main audio channels, L by the number of LFE channels, I by the number of independently switched coupling channels, D by the number of dependently switched coupling channels, and Profile Name by the actual profile name. An example would be a 5.1.1.1 Channel Main Profile MPEG-2 AAC Decoder, indicating a decoder capable of decoding 5 main audio channels, one LFE channel, and one each independently and dependently switched CCE, with each of the channels using the profile specified. This can be abbreviated as M.5.1.1.1 where the "M" indicates a main profile decoder. Similarly, a Low Complexity decoder can be specified by a leading "L", and an SSR profile by an "S".

7.1.4.1 Naming Convention for MPEG-2 AAC + MPEG-4 SBR Decoders and Bitstreams

A decoder or bitstream conforming additionally to the MPEG-4 AOT SBR at a certain level may be referenced in a similar manner by appending "+ SBR / X [HQ/LP]" to the name, where X is replaced with the level of the HE-AAC profile decoder/bitstream with the same characteristics as specified by ISO/IEC 14496-3. An example would be a 5.1.1.1 Channel Main Profile MPEG-2 AAC + SBR / 5 HQ Decoder.

7.1.5 Minimum Decoder Capability for Specified Number of Main Audio Channels and Profile

To insure a certain level of interoperability the following minimum decoder capabilities for decoders of a given profile and number of main audio channels are specified.

Table 32 — Profile dependent minimum decoder capabilities in terms of channel configuration

Number of Main Audio Channels	Main Profile Capability	Low Complexity Profile Capability	SSR Profile Capability
1	1.0.0.0	1.0.0.0	1.0.0.0
2	2.0.0.0	2.0.0.0	2.0.0.0
3	3.0.1.0	3.0.0.1	3.0.0.0
4	4.0.1.0	4.0.0.1	4.0.0.0
5	5.1.1.1	5.1.0.1	5.1.0.0
7	7.1.1.2	7.1.0.2	7.1.0.0

7.1.6 Profile Dependent Tool Parameters

Maximum TNS order and bandwidth:

According to the profile in use, the value for the constant `TNS_MAX_ORDER` is set as follows for long windows: For the main profile the constant `TNS_MAX_ORDER` is 20, for the low complexity profile and the scalable sampling rate profile the constant `TNS_MAX_ORDER` is 12. For short windows, the constant `TNS_MAX_ORDER` is 7 for all profiles.

According to the sampling rate and profile in use, the value for the constant TNS_MAX_BANDS is set as follows:

Table 33 — Profile and sampling rate dependent definition of TNS_MAX_BANDS

Sampling Rate [Hz]	Low Complexity / Main Profile (long windows)	Low Complexity / Main Profile (short windows)	Scalable Sampling Rate Profile (long windows)	Scalable Sampling Rate Profile (short windows)
96000	31	9	28	7
88200	31	9	28	7
64000	34	10	27	7
48000	40	14	26	6
44100	42	14	26	6
32000	51	14	26	6
24000	46	14	29	7
22050	46	14	29	7
16000	42	14	23	8
12000	42	14	23	8
11025	42	14	23	8
8000	39	14	19	7

7.2 Profile Interoperability

7.2.1 Interoperability of Bitstreams and Decoders

Any bitstream of a given profile (see Table 34) whose number of main audio channels, LFE channels, independent coupling channels, and dependent coupling channels is less than or equal to the corresponding number of channels supported by a decoder of the same profile can be decoded by that decoder.

Table 34 describes the interoperability of the three profiles.

Table 34 — Profile Interoperability

Decoder Profile	Encoder Profile		
	Main Profile	LC Profile	SSR Profile
Main Profile	yes	yes	no *
LC Profile	no	yes	no *
SSR Profile	no	no **	yes

*In Table 34, these entries can be decoded if the main or LC profile decoder is able to parse, but not decode, the gain control information, but the reconstructed audio will have a limited bandwidth.

**In Table 33, this entry can be decoded, but the bandwidth of the decoded signal will be limited to approximately 5 kHz, corresponding to the nonaliased portion of the first PQMF filter band.

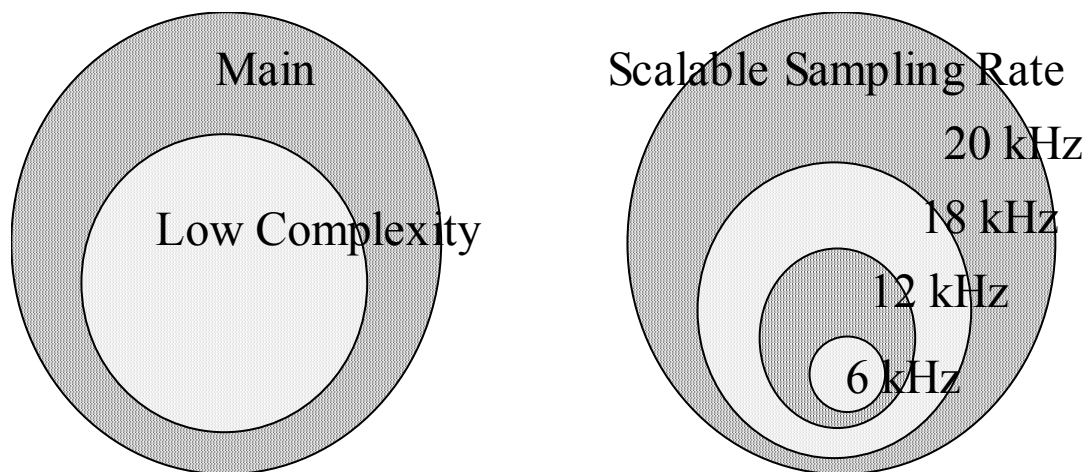


Figure 3 — Profile Interoperability

8 Overall Data Structure

8.1 AAC Interchange Formats

8.1.1 Overview

The `raw_data_block()` contains all data which belongs to the audio (including ancillary data). Beyond that, additional information like `sampling_frequency` is needed to fully describe an audio sequence. The Audio Data Interchange Format (ADIF) contains all elements that are necessary to describe a bitstream according to this standard.

For specific applications some or all of the syntax elements like those specified in the header of the ADIF, e.g. `sampling_rate`, may be known to the decoder by other means and hence do not appear in the bitstream.

Furthermore, additional information that varies from block to block (e.g. to enhance the parsability or error resilience) may be required. Therefore transport streams may be designed for a specific application and are not specified in this standard. However, one non-normative transport stream, called Audio Data Transport Stream (ADTS), is described. It may be used for applications in which the decoder can parse this stream.

8.1.2 Audio Data Interchange Format (ADIF)

8.1.2.1 Overview

The Audio Data Interchange Format (ADIF) contains one header at the start of the sequence followed by a `raw_data_stream()`. The `raw_data_stream()` may not contain any further `program_config_element()`'s.

As such, the ADIF is useful only for systems with a defined start and no need to start decoding from within the audio data stream, such as decoding from disk file. It can be used as an interchange format in that it contains all information necessary to decode and play the audio data.

8.1.2.2 Definitions

8.1.2.2.1 Data Functions

<code>adif_sequence()</code>	a sequence according to the Audio Data Interchange Format (Table 2).
<code>adif_header()</code>	header of the Audio Data Interchange Format located at the beginning of an <code>adif_sequence</code> (Table 3).
<code>byte_alignment()</code>	Align with respect to the first bit of the header.
<code>raw_data_stream()</code>	see subclause 8.2.1 and Table 11.
<code>program_config_element()</code>	contains information about the configuration for one program (Table 3). See subclause 8.5.

8.1.2.2.2 Data Elements

<code>adif_id</code>	ID that indicates the Audio Data Interchange Format. Its value is 0x41444946 (most significant bit first), the ASCII representation of the string „ADIF“ (Table 3).
<code>copyright_id_present</code>	indicates whether <code>copyright_id</code> is present or not (Table 3).
<code>copyright_id</code>	The field consists of an 8-bit <code>copyright_identifier</code> , followed by a 64-bit <code>copyright_number</code> (Table 3). The copyright identifier is given by a Registration Authority as designated by SC 29. The <code>copyright_number</code> is a value which identifies uniquely the copyrighted material. See ISO/IEC 13818-3, definition of data element <code>copyright_identification_bit</code> .
<code>original_copy</code>	see ISO/IEC 11172-3, definition of data element <code>copyright</code> .
<code>home</code>	see ISO/IEC 11172-3, definition of data element <code>original/copy</code> .
<code>bitstream_type</code>	a flag indicating the type of a bitstream (Table 3): ‘0’ constant rate bitstream. This bitstream may be transmitted via a channel with constant rate ‘1’ variable rate bitstream. This bitstream is not designed for transmission via constant rate channels
<code>bitrate</code>	a 23 bit unsigned integer indicating either the bitrate of the bitstream in bits/sec in case of constant rate bitstream or the maximum peak bitrate (measured per frame) in case of variable rate bitstreams. A value of 0 indicates that the bitrate is not known (Table 3).
<code>num_program_config_element</code>	number of <code>program_config_element()</code> ’s specified for this <code>adif_sequence()</code> is equal to <code>num_program_config_element+1</code> (Table 3). The minimum value is 0 indicating 1 <code>program_config_element()</code> .
<code>adif_buffer_fullness</code>	state of the bit reservoir after encoding the first <code>raw_data_block()</code> in the <code>adif_sequence()</code> . It is transmitted as the number of available bits in the bit reservoir (Table 3).

8.1.2.2.3 Help Elements

data_available() Function that returns '1' as long as data is available, otherwise '0'.

8.1.3 Audio Data Transport Stream (ADTS)

8.1.3.1 Overview

The Audio Data Transport Stream (ADTS) is similar to syntax used in ISO/IEC 11172-3 and ISO/IEC 13818-3. This will be recognized by ISO/IEC 11172-3 and ISO/IEC 13818-3 decoders as a "Layer 4" bitstream.

The fixed header of the ADTS contains the syncword plus all parts of the header which are necessary for decoding and which do not change from frame to frame. The variable header of the ADTS contains header data which changes from frame to frame.

8.1.3.2 Definitions

8.1.3.2.1 Data Functions

adts_sequence() a sequence according to Audio Data Transport Stream ADTS (Table 4).

adts_frame() an ADTS frame, consisting of a fixed header, a variable header, an optional error check and a specified number of *raw_data_block()*'s (Table 5).

adts_fixed_header() fixed header of ADTS. The information in this header does not change from frame to frame. It is repeated every frame to allow random access into a bitstream bitstream (Table 8).

adts_variable_header() variable header of ADTS. This header is transmitted every frame as well as the fixed header, but contains data that changes from frame to frame (Table 9).

adts_error_check() The following bits are protected and fed into the CRC algorithm in order of their appearance:

- all bits of *adts_fixed_header()*
- all bits of *adts_variable_header()*
- first 192 bits of any
 - *single_channel_element()*
 - *channel_pair_element()*
 - *coupling_channel_element()*
 - *lfe_channel_element()*
- First 128 bits of the second individual *channel_stream()* in the *channel_pair_element()* must be protected.
- All information in any *program_config_element()* or *data_stream_element()* must be protected.

For any element where the specified protection length of 128 or 192 bits exceeds its actual length, the element is zero padded to the specified protection length for CRC calculation.

The `id_syn_ele` bits shall be excluded from CRC protection. If the length of a CPE is shorter than 192 bits, zero data are appended to achieve the length of 192 bits. Furthermore, if the first ICS of the CPE ends at the N th bit ($N < 192$), the first $(192 - N)$ bits of the second ICS are protected twice, each time in order of their appearance. For example, if the second ICS starts at the 190th bit of CPE, the first 3 bits of the second ICS are protected twice. Finally, if the length of the second ICS is shorter than 128 bits, zero data are appended to achieve the length of 128 bits.

`adts_header_error_check()`

The following bits are protected and fed into the CRC algorithm in order of their appearance:

- all bits of `adts_fixed_header()`
- all bits of `adts_variable_header()`
- all bits of every `raw_data_block_position[i]`.

`adts_raw_data_block_error_check()`

With regard to the i -th `adts_raw_data_block_error_check()`, the bits of the i -th `raw_data_block()` are protected and fed into the CRC algorithm in order of their appearance according to what is specified with regard to the `adts_error_check()` with the exception that no header bits are considered.

`raw_data_block()`

see subclause 8.2.1 and Table 12.

8.1.3.2.2 Data Elements

`raw_data_block_position[i]`

Start position of the i -th `raw_data_block()` in the `adts_frame()`, measured as an offset in bytes from the start position of the first `raw_data_block()` in the `adts_frame()`.

`crc_check`

CRC error detection data generated as described in ISO/IEC 11172-3, subclause 2.4.3.1 (Table 6, Table 7 and Table 10).

`syncword`

The bit string '1111 1111 1111'. See ISO/IEC 11172-3, subclause 2.4.2.3 (Table 8).

`ID`

MPEG identifier, set to '1'. See ISO/IEC 11172-3, subclause 2.4.2.3 (Table 8).

`layer`

Indicates which layer is used. Set to '00'. See ISO/IEC 11172-3, subclause 2.4.2.3 (Table 8).

`protection_absent`

Indicates whether `error_check()` data is present or not. Same as syntax element 'protection_bit' in ISO/IEC 11172-3, subclause 2.4.1 and 2.4.2 (Table 8).

`profile`

profile used. See clause 2 (Table 8).

`sampling_frequency_index`

indicates the sampling frequency used according to the following table (Table 8):

Table 35 — Sampling frequency dependent on `sampling_frequency_index`

<code>sampling_frequency_index</code>	sampling frequency [Hz]
0x0	96000
0x1	88200
0x2	64000
0x3	48000
0x4	44100
0x5	32000
0x6	24000
0x7	22050
0x8	16000
0x9	12000
0xa	11025
0xb	8000
0xc	reserved
0xd	reserved
0xe	reserved
0xf	reserved

private_bit

see ISO/IEC 11172-3, subclause 2.4.2.3 (Table 8).

channel_configuration

indicates the channel configuration used. If `channel_configuration` is greater than 0, the channel configuration is given in Table 42, see subclause 8.5.3.1. If `channel_configuration` equals 0, the channel configuration is not specified in the header and must be given by a `program_config_element()` following as first syntactic element in the first `raw_data_block()` after the header (see subclause 8.5.3.2), or by the implicit configuration (see subclause 8.5.3.3) or must be known in the application (Table 8).

original_copy

see definition in 8.1.2.2.2.

home

see definition in 8.1.2.2.2.

copyright_identification_bit

One bit of the 72-bit copyright identification field (see `copyright_id` above). The bits of this field are transmitted frame by frame; the first bit is indicated by the `copyright_identification_start` bit set to '1'. The field consists of an 8-bit `copyright_identifier`, followed by a 64-bit `copyright_number`. The `copyright_identifier` is given by a Registration Authority as designated by SC29. The `copyright_number` is a value which identifies uniquely the copyrighted material. See ISO/IEC 13818-3, subclause 2.5.2.13 (Table 9).

copyright_identification_start

One bit to indicate that the `copyright_identification_bit` in this audio frame is the first bit of the 72-bit copyright identification. If no copyright identification is transmitted, this bit should be kept '0'. '0' no start of copyright identification in this audio frame '1' start of copyright identification in this audio frame See ISO/IEC 13818-3, subclause 2.5.2.13 (Table 9).

aac_frame_length

Length of the frame including headers and `error_check` in bytes (Table 9).

adts_buffer_fullness

state of the bit reservoir in the course of encoding the ADTS frame, up to and including the first `raw_data_block()` and the

optionally following `adts_raw_data_block_error_check()`. It is transmitted as the number of available bits in the bit reservoir divided by NCC divided by 32 and truncated to an integer value (Table 9). A value of hexadecimal 7FF signals that the bitstream is a variable rate bitstream. In this case, buffer fullness is not applicable.

number_of_raw_data_blocks_in_frame

Number of `raw_data_block()`'s that are multiplexed in the `adts_frame()` is equal to `number_of_raw_data_blocks_in_frame` + 1. The minimum value is 0 indicating 1 `raw_data_block()` (Table 9).

8.2 Raw Data

8.2.1 Definitions

8.2.1.1 Data Functions

`raw_data_stream()`

sequence of `raw_data_block()`'s.

`raw_data_block()`

block of raw data that contains audio data for a time period of 1024 samples, related information and other data. There are seven syntactic elements, identified by the data element `id_syn_ele`. The `audio_channel_element()`'s in one `raw_data_stream()` and one `raw_data_block()` must have one and only one sampling rate. In the `raw_data_block()`, several instances of the same syntactic element may occur, but must have a different 4 bit `element_instance_tag`, except for `data_stream_element()`'s and `fill_element()`'s. Therefore, in one `raw_data_block()`, there can be from 0 to at most 16 instances of any syntactic element, except for `data_stream_element()`'s and `fill_element()`'s, where this limitation does not apply. If multiple `data_stream_element()`'s occur which have the same `element_instance_tag` then they are part of the same data stream. The `fill_element()` has no `element_instance_tag` (since the content does not require subsequent reference) and can occur any number of times. The end of a `raw_data_block()` is indicated with a special `id_syn_ele` (TERM), which may occur only once in a `raw_data_block()`. (Table 12).

`single_channel_element()`

abbreviation SCE. Syntactic element of the bitstream containing coded data for a single audio channel. A `single_channel_element()` basically consists of an `individual_channel_stream()`. There may be up to 16 such elements per raw data block, each one must have a unique `element_instance_tag` (Table 13).

`channel_pair_element()`

abbreviation CPE. Syntactic element of the bitstream containing data for a pair of channels. A `channel_pair_element()` consists of two `individual_channel_stream()`'s and additional joint channel coding information. The two channels may share common side information. The `channel_pair_element()` has the same restrictions as the single channel element as far as `element_instance_tag`, and number of occurrences (Table 14).

`coupling_channel_element()`

Abbreviation CCE. Syntactic element that contains audio data for a coupling channel. A coupling channel represents the information for multi-channel intensity for one block, or alternately for dialogue for multilingual programming. The rules for number

lfe_channel_element()

of coupling_channel_element()'s and instance tags are as for single_channel_element()'s (Table 22). See subclause 12.3.

Abbreviation LFE. Syntactic element that contains a low sampling frequency enhancement channel. The rules for the number of lfe_channel_element()'s and instance tags are as for single_channel_element()'s (Table 23). See subclause 8.4.

audio_channel_element()

generic term for single_channel_element(), channel_pair_element(), coupling_channel_element() and lfe_channel_element().

program_config_element()

Abbreviation PCE. Syntactic element that contains program configuration data. The rules for the number of program_config_element()'s and element_instance_tag's are the same as for single_channel_element()'s (Table 25). PCE's must come before all other syntactic elements in a raw_data_block(). See subclause 8.5.

fill_element()

Abbreviation FIL. Syntactic element that contains fill data. There may be any number of fill elements, that can come in any order in the raw data block (Table 26). See subclause 8.7.

data_stream_element()

Abbreviation DSE. Syntactic element that contains data. Again, there are 16 element_instance_tags. There is, however, no restriction on the number of data_stream_element()'s with any one instance tag, as a single data stream may continue across multiple data_stream_element()'s with the same instance tag (Table 24). See subclause 8.5.3.

byte_alignment()

Align with respect to the first bit of the raw_data_block().

8.2.2 Data Elements

id_syn_ele

a data element that identifies either a syntactic element or the end of a raw_data_block() (Table 12):

Table 36 — Syntactic element identification

ID name	encoding	Abbreviation	Syntactic Element
ID_SCE	0x0	SCE	single_channel_element()
ID_CPE	0x1	CPE	channel_pair_element()
ID_CCE	0x2	CCE	coupling_channel_element()
ID_LFE	0x3	LFE	lfe_channel_element()
ID_DSE	0x4	DSE	data_stream_element()
ID_PCE	0x5	PCE	program_config_element()
ID_FIL	0x6	FIL	fill_element()
ID_END	0x7	TERM	

element_instance_tag Unique instance tag for syntactic elements other than fill_element(). All syntactic elements containing instance tags may occur more than once, but, except for data_stream_element()'s, must have a unique element_instance_tag in each raw_data_block(). This tag is also used to reference audio syntactic elements in single_channel_element()'s, channel_pair_element()'s, lfe_channel_element()'s, data_channel_element()'s, and coupling_channel_element()'s inside a program_config_element(), and provides the possibility of up to 16 independent program_config_element()'s (Table 13, Table 14, Table 22, Table 23, Table 24, Table 25, Table 26).

8.2.3 Buffer Requirements

8.2.3.1 Minimum Decoder Input Buffer

The following rules are used to calculate the maximum number of bits in the input buffer both for the bitstream as a whole, for any given program, or for any given SCE/CPE/CCE:

The input buffer size is 6144 bits per SCE or independently switched CCE, plus 12288 bits per CPE (6144*NCC). Both the total buffer and the individual buffer sizes are limited, so that the buffering limit can be calculated for either the entire bitstream, any entire program, or the individual audio_channel_element()'s permitting the decoder to break a multichannel bitstream into separate mono and stereo bitstreams which are decoded by separate mono and stereo decoders, respectively. All bits for LFE's or dependent CCE's must be supplied from the total buffer requirements based on the independent CCE's, SCE's, and CPE's. Furthermore, all bits required for any DSE's, PCE's, FIL's, or fixed headers, variable headers, byte_alignment, and CRC must also be supplied from the same total buffer requirements.

8.2.3.2 Bit Reservoir

The bit reservoir is controlled at the encoder. The maximum bit reservoir in the encoder depends on the NCC and the mean bitrate. The maximum bit reservoir size for constant rate channels can be calculated by subtracting the mean number of bits per block from the minimum decoder input buffer size. For example, at 96 kbit/s for a stereo signal at 44.1 kHz sampling frequency the mean number of bits per block (mean_framelength) is (96000 bit/s / 44100 1/s * 1024) = 2229.1156... . This leads to a maximum bit reservoir size (max_bit_reservoir) of INT (12288 bit - 2229.1156...) = 10058. For variable bitrate channels the encoder must operate in a way that the input buffer requirements do not exceed the minimum decoder input buffer.

The state of the bit reservoir (bit_reservoir_state) is transmitted in the buffer_fullness field, either as the state of the bit reservoir truncated to an integer value (adif_buffer_fullness) or as the state of the bit reservoir divided by NCC divided by 32 and truncated to an integer value (adts_buffer_fullness).

The bit_reservoir_state of subsequent frames can be derived as follows:

$$bit_reservoir_state[frame] = bit_reservoir_state[frame - 1] + mean_framelength - framelength[frame]$$

Framelengths have to be adjusted such that the following restriction is met

$$0 \leq bit_reservoir_state[frame] \leq max_bit_reservoir$$

8.2.3.3 Maximum Bitrate

Maximum bitrate:

The maximum bitrate depends on the audio sampling rate. It can be calculated based on the minimum input buffer size according to the formula:

$$\frac{6144 \frac{bit}{block}}{1024 \frac{samples}{block}} \cdot sampling_frequency \cdot NCC$$

Table 37 gives some examples of the maximum bitrates per channel depending on the used sampling frequency.

Table 37 — Maximum bitrate depending on the sampling frequency

sampling_frequency	maximum bitrate / NCC
48 kHz	288 kbit/s
44.1 kHz	264.6 kbit/s
32 kHz	192 kbit/s

8.2.4 Decoding Process

Assuming that the start of a raw_data_block() is known, it can be decoded without any additional „transport-level“ information and produces 1024 audio samples per output channel. The sampling rate of the audio signal, as specified by the **sampling_frequency_index**, may be specified in a program_config_element() or it may be implied in the specific application domain. In the latter case, the **sampling_frequency_index** must be deduced in order for the bitstream to be parsed.

Since a given sampling frequency is associated with only one sampling frequency table, and since maximum flexibility is desired in the range of possible sampling frequencies, the following Table shall be used to associate an implied sampling frequency with the desired sampling frequency dependent tables.

Table 38 — Sampling frequency mapping

Frequency range (in Hz)	Use tables for sampling frequency (in Hz)
f >= 92017	96000
92017 > f >= 75132	88200
75132 > f >= 55426	64000
55426 > f >= 46009	48000
46009 > f >= 37566	44100
37566 > f >= 27713	32000
27713 > f >= 23004	24000
23004 > f >= 18783	22050
18783 > f >= 13856	16000
13856 > f >= 11502	12000
11502 > f >= 9391	11025
9391 > f	8000

The raw_data_stream supports encoding for both constant rate and variable rate channels. In each case the structure of the bitstream and the operation of the decoder are identical except for some minor qualifications. For constant rate channels, the encoder may have to insert a FIL element to adjust the rate upwards to exactly the desired rate. A decoder reading from a constant rate channel must accumulate a minimum number of bits in its input buffer prior to the start of decoding so that output buffer underrun does not occur. In the case of variable rate, demand read channels, each raw_data_block() can have the minimum length (rate) such that the desired audio quality is achieved, and in the decoder there is no minimum input data requirement prior to the start of decoding.

Examples of the simplest possible bitstreams are:

bitstream segment

<SCE><TERM><SCE><TERM>...
<CPE><TERM><CPE><TERM>...
<SCE><CPE><CPE><LFE><TERM><SCE><CPE><CPE><LFE><TERM>...

output signal

mono signal
stereo signal
5.1 channel signal

where angle brackets (< >) are used to delimit syntactic elements. For the mono signal each SCE must have the same value in its **element_instance_tag**, and similarly, for the stereo signal each CPE must have the same value in its **element_instance_tag**. For the 5.1 channel signal each SCE must have the same value in