INTERNATIONAL STANDARD

ISO/IEC 14496-3

Third edition
2005-12-01

AMENDMENT 9
2008-07-01

Information technology — Coding of audio-visual objects

Part 3:
Audio

AMENDMENT 9: Enhanced low delay AAC

Technologies de l'information — Codage des objets audiovisuels

Partie 3: Codage audio

AMENDEMENT 9: Retard faible amélioré AAC

Reference number
ISO/IEC 14496-3:2005/Amd.9:2008(E)

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.

COPYRIGHT PROTECTED DOCUMENT

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

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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

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

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

Amendment 9 to ISO/IEC 14496-3:2005 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Information technology — Coding of audio-visual objects

Part 3: **Audio**

AMENDMENT 9: Enhanced low delay AAC

In the following, changes in existing text and tables are highlighted by grey background.

In 1.5.1.1, extend Table 1.1 with the following entries:

Table 1.1 — Audio Object Type definition based on Tools/Modules

Object Type ID	Audio Object Type	gain control	block switching	window shapes - standard	window shapes – AAC LD	Low Delay Window	filterbank - standard	filterbank - SSR	TNS	LTP	intensity	coupling	frequency deomain prediction	PNS	MS	SIAQ	FSS	upsampling filter tool	quantisation&coding - AAC	quantisation&coding - TwinVQ	quantisation&coding - BSAC	AAC ER Tools	ER payload syntax	EP Tool 1)	CELP	Silence Compression	HVXC	HVXC 4kbit/s VR	SA tools	SASBF	MIDI	HILN	TTSI	SBR	Layer-1	Layer-2	Layer-3	SSC (Transient, Sinusoid, Noise)	Parametric stereo	Low Delay SBR	Remark
	…																																								
39	ER AAC ELD					X	X		X		X			X	X				X			X	X	X																X	
40-95	(reserv-ed)																																								

Before 1.5.2, add 1.5.1.2.37:

1.5.1.2.37 Error Resilient (ER) AAC ELD object type

The enhanced low delay AAC object type (ER AAC ELD) is identical to the ER AAC LD object type, plus the utilization of a Low Delay Filterbank (LDFB) and an enhanced low delay window. It also permits combinations with the PNS tool as well as the Low Delay SBR tool. The ER AAC ELD object type provides the ability to extend the usage of generic low bitrate audio coding to applications requiring a very low delay of the encoding / decoding chain (e.g. full-duplex real-time communications)."

In 1.5.2.4, insert the following new entries for the Baseline MPEG Surround Profile into Table 1.12 audioProfileLevelIndication and adapt the "reserved for ISO use" range accordingly (changes are highlighted in gray):

Value	Profile	Level
…	…	…
0x33	High Efficiency AAC v2 Profile	L5
0x34	Low Delay AAC Profile	L1
0x35	Baseline MPEG Surround Profile (see ISO/IEC 23003-1)	L1
0x36	Baseline MPEG Surround Profile (see ISO/IEC 23003-1)	L2
0x37	Baseline MPEG Surround Profile (see ISO/IEC 23003-1)	L3
0x38	Baseline MPEG Surround Profile (see ISO/IEC 23003-1)	L4
0x39	Baseline MPEG Surround Profile (see ISO/IEC 23003-1)	L5
0x3A	Baseline MPEG Surround Profile (see ISO/IEC 23003-1)	L6
0x3B - 0x7F	reserved for ISO use	-
0x80 - 0xFD	user private	-
0xFE	no audio profile specified	-
0xFF	no audio capability required	-
NOTE — Usage of the value 0xFE indicates that the content described by this InitialObjectDescriptor does not comply to any audio profile specified in ISO/IEC 14496-3. Usage of the value 0xFF indicates that none of the audio profile capabilities are required for this content.		

In 1.6.2.1, extend Table 1.13 "AudioSpecificConfig()" as follows:

Table 1.13 — Syntax of AudioSpecificConfig()

Syntax	No. of bits	Mnemonic
AudioSpecificConfig ()		
{		
audioObjectType = GetAudioObjectType();		
samplingFrequencyIndex;	**4**	**bslbf**
if (samplingFrequencyIndex == 0xf) {		
samplingFrequency;	**24**	**uimsbf**
}		
channelConfiguration;	**4**	**bslbf**
sbrPresentFlag = -1;		
psPresentFlag = -1;		
if (audioObjectType == 5 ||		
audioObjectType == 29) {		
extensionAudioObjectType = audioObjectType;		
sbrPresentFlag = 1;		
if (audioObjectType == 29) {		
psPresentFlag = 1;		
}		
extensionSamplingFrequencyIndex;	**4**	**uimsbf**
if (extensionSamplingFrequencyIndex == 0xf)		
extensionSamplingFrequency;	**24**	**uimsbf**
audioObjectType = GetAudioObjectType();		
}		
else {		
extensionAudioObjectType = 0;		
}		
switch (audioObjectType) {		

```
case 1:
case 2:
case 3:
case 4:
case 6:
case 7:
case 17:
case 19:
case 20:
case 21:
case 22:
case 23:
    GASpecificConfig();
    break:
case 8:
    CelpSpecificConfig();
    break;
case 9:
    HvxcSpecificConfig();
    break:
case 12:
    TTSSpecificConfig();
    break;
case 13:
case 14:
case 15:
case 16:
    StructuredAudioSpecificConfig();
    break;
case 24:
    ErrorResilientCelpSpecificConfig();
    break;
case 25:
    ErrorResilientHvxcSpecificConfig();
    break;
case 26:
case 27:
    ParametricSpecificConfig();
    break;
case 28:
    SSCSpecificConfig();
    break;
case 32:
case 33:
case 34:
    MPEG_1_2_SpecificConfig();
    break;
case 35:
    DSTSpecificConfig();
    break;
case 36:
    fillBits;                                      5          bslbf
    ALSSpecificConfig();
    break;
case 37:
case 38:
    SLSSpecificConfig();
    break;
case 39:
        ELDSpecificConfig(channelConfiguration);
        break;
    default:
        /* reserved */
    }
    switch (audioObjectType) {
    case 17:
    case 19:
    case 20:
    case 21:
    case 22:
    case 23:
    case 24:
    case 25:
    case 26:
    case 27:
    case 39:
        epConfig;                                                   2       bslbf
        if ( epConfig == 2 || epConfig == 3 ) {
            ErrorProtectionSpecificConfig();
        }
        if ( epConfig == 3 )
            directMapping;                                          1       bslbf
            if ( ! directMapping ) {
                /* tbd */
            }
        }
    }
    if ( extensionAudioObjectType != 5 && bits_to_decode() >= 16 ) {
        syncExtensionType;                                          11      bslbf
        if (syncExtensionType == 0x2b7) {
            extensionAudioObjectType = GetAudioObjectType();
            if ( extensionAudioObjectType == 5 ) {
                sbrPresentFlag;                                     1       uimsbf
                if (sbrPresentFlag == 1) {
                    extensionSamplingFrequencyIndex;                4       uimsbf
                    if ( extensionSamplingFrequencyIndex == 0xf ) {
                        extensionSamplingFrequency;                 24      uimsbf
                    }
                if ( bits_to_decode() >= 12 ) {
                    syncExtensionType;                              11      bslbf
                    if (syncExtensionType == 0x548) {
                        psPresentFlag;                              1       uimsbf
                    }
                }
            }
        }
    }
}
```

In 1.6.2.2.1, extend Table 1.15 "Audio Object Types" as follows:

Table 1.15 — Audio Object Types

Object Type ID	Audio Object Type	definition of elementary stream payloads and detailed syntax	Mapping of audio payloads to access units and elementary streams
…	…	…	…
39	ER AAC ELD	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.4

Add 1.6.2.2.4 with the title "ER AAC ELD":

1.6.2.2.2.4 ER AAC ELD

The top level payload for ER AAC ELD is defined in er_raw_data_block_eld(). All definitions mentioned in subclause 1.6.2.2.2.3 are also valid for this AOT.

At the end of 1.6.5.1, add a sentence:

"NOTE: None of these signaling methods described in this subclause is allowed for AAC ELD in order to signal the low delay sbr tool. For this case the ldSbrPresentFlag in the ELDSpecificConfig is to be used"

Before 4.5, insert Tables AMD9.2 to AMD9.8:

Table AMD9.2 — Syntax of top level payload for audio object types ER AAC ELD (er_raw_data_block_eld())

Syntax	No. of bits	Mnemonic
er_raw_data_block_eld(channelConfiguration)		
{		
switch(channelConfiguration) {		
case 1:		
single_channel_element_eld();		
break;		
case 2:		
channel_pair_element_eld ();		
break;		
case 3:		
single_channel_element_eld ();		
channel_pair_element_eld ();		
break;		
case 4:		
single_channel_element_eld ();		
channel_pair_element_eld ();		
single_channel_element_eld ();		
break;		

```
        case 5:
            single_channel_element_eld ();
            channel_pair_element_eld ();
            channel_pair_element_eld ();
            break;
        case 6:
            single_channel_element_eld ();
            channel_pair_element_eld ();
            channel_pair_element_eld ();
            lfe_channel_element_eld ();
            break;
        case 7:
            single_channel_element_eld ();
            channel_pair_element_eld ();
            channel_pair_element_eld ();
            channel_pair_element_eld ();
            lfe_channel_element_eld ();
            break;
        default:
            /* reserved */
            break;
    }
    if (ldSbrPresentFlag) {
        er_low_delay_sbr_block(channelConfiguration);
    }
    cnt = bits_to_decode() / 8;
    while ( cnt >= 1 ) {
        cnt -= extension_payload(cnt);
    }
    byte_alignment();
}
```

Table AMD9.3 — Syntax of single_channel_element_eld()

Syntax	No. of bits	Mnemonic
single_channel_element_eld()		
{		
individual_channel_stream_eld (0);		
}		

Table AMD9.4 — Syntax of lfe_channel_element_eld()

Syntax	No. of bits	Mnemonic
lfe_channel_element_eld()		
{		
individual_channel_stream_eld (0);		
}		

Table AMD9.5 — Syntax of channel_pair_element_eld()

Syntax	No. of bits	Mnemonic
channel_pair_element_eld()		
{		
common_window = 1;		
max_sfb;	**6**	**uimsbf**
ms_mask_present;	**2**	**uimsbf**
if (ms_mask_present == 1) {		
for (sfb = 0; sfb < max_sfb; sfb++) {		
ms_used[0][sfb**];**	**1**	**uimsbf**
}		
}		
individual_channel_stream_eld (common_window);		
individual_channel_stream_eld (common_window);		
}		

Table AMD9.6 — Syntax of individual_channel_stream_eld()

Syntax	No. of bits	Mnemonic
individual_channel_stream_eld (common_window)		
{		
global_gain;	**8**	**uimsbf**
if (! common_window) {		
max_sfb;	**6**	**uimsbf**
}		
section_data ();		
scale_factor_data ();		
tns_data_present;	**1**	**uimsbf**
if (tns_data_present) {		
tns_data ();		
}		
if (! aacSpectralDataResilienceFlag) {		
spectral_data ();		
}		
else {		
length_of_reordered_spectral_data;	**14**	**uimsbf**
length_of_longest_codeword;	**6**	**uimsbf**
reordered_spectral_data ();		
}		
}		

Table AMD9.7 — Syntax of er_low_delay_sbr_block

Syntax	No. of bits	Mnemonic
er_low_delay_sbr_block(channelConfiguration)		
{		
switch (channelConfiguration) {		
case 1:		
low_delay_sbr_data(ID_SCE, ldSbrCrcFlag, bs_amp_res);		
break;		
case 2:		
low_delay_sbr_data(ID_CPE, ldSbrCrcFlag, bs_amp_res);		
break;		
case 3:		
low_delay_sbr_data(ID_SCE, ldSbrCrcFlag, bs_amp_res);		
low_delay_sbr_data(ID_CPE, ldSbrCrcFlag, bs_amp_res);		
break;		
case 4:		
low_delay_sbr_data(ID_SCE, ldSbrCrcFlag, bs_amp_res);		
low_delay_sbr_data(ID_CPE, ldSbrCrcFlag, bs_amp_res);		
low_delay_sbr_data(ID_SCE, ldSbrCrcFlag, bs_amp_res);		
break;		
case 5:		
low_delay_sbr_data(ID_SCE, ldSbrCrcFlag, bs_amp_res);		
low_delay_sbr_data(ID_CPE, ldSbrCrcFlag, bs_amp_res);		
low_delay_sbr_data(ID_CPE, ldSbrCrcFlag, bs_amp_res);		
break;		
case 6:		
low_delay_sbr_data(ID_SCE, ldSbrCrcFlag, bs_amp_res);		
low_delay_sbr_data(ID_CPE, ldSbrCrcFlag, bs_amp_res);		
low_delay_sbr_data(ID_CPE, ldSbrCrcFlag, bs_amp_res);		
break;		
case 7:		
low_delay_sbr_data(ID_SCE, ldSbrCrcFlag, bs_amp_res);		
low_delay_sbr_data(ID_CPE, ldSbrCrcFlag, bs_amp_res);		
low_delay_sbr_data(ID_CPE, ldSbrCrcFlag, bs_amp_res);		
low_delay_sbr_data(ID_CPE, ldSbrCrcFlag, bs_amp_res);		
break;		
default:		
/* reserved */		
break;		
}		
}		

Table AMD9.8 — Syntax of low_delay_sbr_data()

Syntax	No. of bits	Mnemonic
low_delay_sbr_data(id_aac, ldSbrCrcFlag, bs_amp_res)		
{		
if (ldSbrCrcFlag) {		
bs_sbr_crc_bits;	**10**	**uimsbf**
}		
if (**bs_header_flag**) {	**1**	
sbr_header();		
}		
If (id_aac==ID_SCE) {		
sbr_single_channel_element(bs_amp_res);		
} else if (id_aac==ID_CPE) {		
sbr_channel_pair_element(bs_amp_res);		
}		
}		
Note 1: bs_amp_res is in general transmitted inside sbr_header() function enclosed in the ELDSpecificConfig(). But the parameter can be updated by the sbr_header() function here.		

Extend Table 4.142 as followings:

Table 4.142 — AAC error sensitivity category assignment for extended payload and low delay sbr payload

extension_payload	low delay sbr payload	data_element	Function
6	-	drc_band_top	dynamic_range_info()
6	-	drc_bands_incr	dynamic_range_info()
6	-	drc_bands_present	dynamic_range_info()
6	-	drc_bands_reserved_bits	dynamic_range_info()
6	-	drc_tag_reserved_bits	dynamic_range_info()
6	-	dyn_rng_ct	dynamic_range_info()
6	-	dyn_rng_sgn	dynamic_range_info()
6	-	excluded_chns_present	dynamic_range_info()
6	-	pce_instance_tag	dynamic_range_info()
6	-	pce_tag_present	dynamic_range_info()
6	-	prog_ref_level	dynamic_range_info()
6	-	prog_ref_level_present	dynamic_range_info()
6	-	prog_ref_level_reserved_bits	dynamic_range_info()
6	-	additional_excluded_chns	excluded_channels()
6	-	exclude_mask	excluded_channels()
5	-	extension_type	Extension_payload()
5	-	data_element_version	Extension_payload()
7	-	fill_byte	Extension_payload()
7	-	fill_nibble	Extension_payload()
7	-	other_bits	Extension_payload()
8	-	dataElementLengthPart	Extension_payload()
8	-	data_element_byte	extension_payload()

9	-	bs_sbr_crc_bits	sbr_extension_data()
9	-	bs_header_flag	sbr_extension_data()
9	-	bs_fill_bits	sbr_extension_data()
9	9	bs_amp_res	sbr_header()
9	9	bs_start_freq	sbr_header()
9	9	bs_stop_freq	sbr_header()
9	9	bs_xover_band	sbr_header()
9	9	bs_reserved	sbr_header()
9	9	bs_header_extra_1	sbr_header()
9	9	bs_header_extra_2	sbr_header()
9	9	bs_freq_scale	sbr_header()
9	9	bs_alter_scale	sbr_header()
9	9	bs_noise_bands	sbr_header()
9	9	bs_limiter_bands	sbr_header()
9	9	bs_limiter_gains	sbr_header()
9	9	bs_interpol_freq	sbr_header()
9	9	bs_smoothing_mode	sbr_header()
9	9	bs_data_extra	sbr_single_channel_element()
9	9	bs_reserved	sbr_single_channel_element()
9	10	bs_add_harmonic_flag	sbr_single_channel_element()
9	10	bs_extended_data	sbr_single_channel_element()
9	10	bs_extension_size	sbr_single_channel_element()
9	10	bs_esc_count	sbr_single_channel_element()
9	10	bs_extension_id	sbr_single_channel_element()
9	9	bs_data_extra	sbr_channel_pair_element()
9	9	bs_reserved	sbr_channel_pair_element()
9	9	bs_coupling	sbr_channel_pair_element()
9	10	bs_add_harmonic_flag	sbr_channel_pair_element()
9	10	bs_extended_data	sbr_channel_pair_element()
9	10	bs_extension_size	sbr_channel_pair_element()
9	10	bs_esc_count	sbr_channel_pair_element()
9	10	bs_extension_id	sbr_channel_pair_element()
9	-	bs_frame_class	sbr_grid()
9	-	tmp	sbr_grid()
9	-	bs_freq_res	sbr_grid()
9	-	bs_pointer	sbr_grid()
9	-	bs_var_bord_0	sbr_grid()
9	-	bs_var_bord_1	sbr_grid()
9	-	bs_num_rel_0	sbr_grid()
9	-	bs_num_rel_1	sbr_grid()
9	9	bs_df_env	sbr_dtdf()
9	9	bs_df_noise	sbr_dtdf()
9	10	bs_invf_mode	sbr_invf()
9	10	bs_env_start_value_balance	sbr_envelope()
9	10	bs_env_start_value_level	sbr_envelope()
9	10	bs_codeword	sbr_envelope()
9	10	bs_noise_start_value_balanc e	sbr_noise()
9	10	bs_noise_start_value_level	sbr_noise()
9	10	bs_codeword	sbr_noise()
9	10	bs_add_harmonic	sbr_sinusoidal_coding()
-	9	bs_frame_class	sbr_ld_grid()
-	9	tmp	sbr_ld_grid()
-	9	bs_freq_res	sbr_ld_grid()
-	9	bs_transient_position	sbr_ld_grid()
-	9	bs_sbr_crc_bits	low_delay_sbr_data()
-	9	bs_header_flag	low_delay_sbr_data()

At the end of 4.5.5, add the following new Figure:

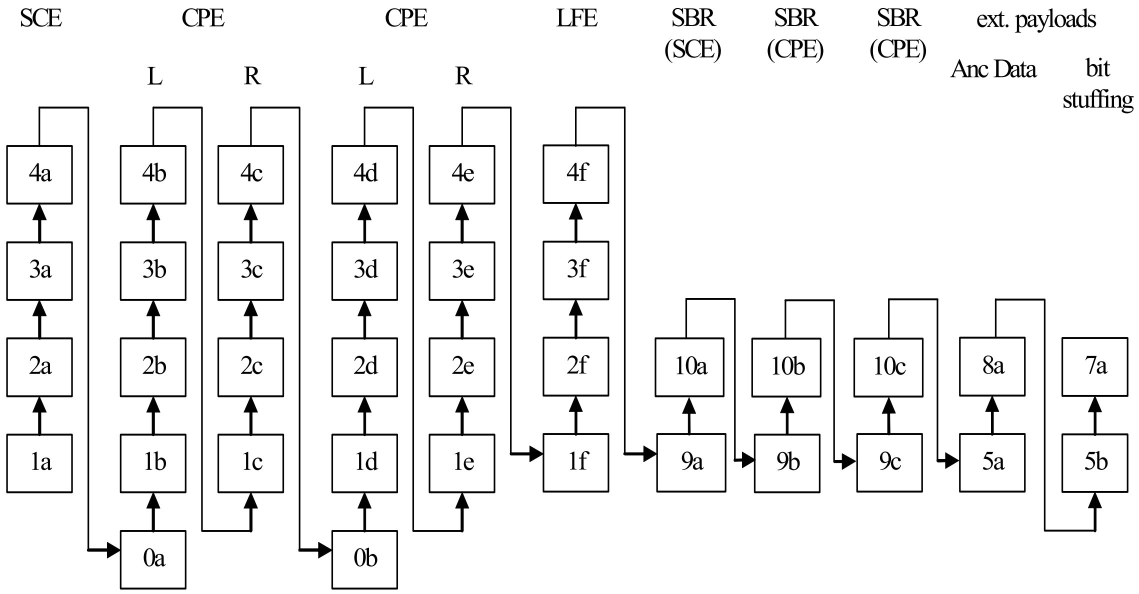

Figure AMD9.1 — Dependency structure in case of ER multichannel AAC ELD syntax (channelConfigurati == 6)

After 4.6.18, add a new subclause with the title "Low Delay SBR":

4.6.19 Low Delay SBR

4.6.19.1 Introduction

Low Delay SBR is derived from the standard SBR tool in order to be utilized as a bandwidth extension coder in communication scenarios. Thus, the algorithmic delay of this tool is minimized to achieve an overall delay low enough for bi-directional communication applications.

Summary of modifications:

- Frame length adopted to a core codec with 512 or 480 samples per frame
- Frame-locked time/frequency grid
- Minimization of delay in QMF buffer
- Utilizing a Complex Low Delay Filterbank

The low delay SBR tool is defined by the following modifications with respect to the standard algorithm (i.e. SBR audio object type). All clauses/subclauses can be found in the brackets after each headline.

4.6.19.2 Definitions, Constants and Variables

4.6.19.2.1 Definitions (changes to 4.6.18.2.1.20)

- **time slot**: finest resolution in time for SBR envelopes and noise floors. One time slot equals one subsample in the QMF domain.