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

**Part 3:
Audio**

AMENDMENT 1: Audio extensions

Technologies de l'information — Codage des objets audiovisuels —

Partie 3: Codage audio

AMENDEMENT 1: Extensions audio



Reference number
ISO/IEC 14496-3:1999/Amd.1:2000(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.

© ISO/IEC 2000

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.ch
Web www.iso.ch

Printed in Switzerland

Contents

Page

1	Scope	1
2	Normative references	1
3	Terms and definitions	2
4	Symbols and abbreviations	2
5	Technical overview.....	2
5.1	Extended MPEG-4 audio object types	2
5.1.1	Audio object type definition.....	2
5.1.2	Description	3
5.2	Audio profiles and levels	5
5.2.1	Profiles.....	5
5.2.2	Complexity units	6
5.2.3	Levels within the profiles.....	7
6	Extension to interface to ISO/IEC 14496-1 (MPEG-4 Systems).....	9
6.1	Introduction	9
6.2	Extension to syntax.....	10
6.2.1	AudioSpecificInfo.....	10
6.2.2	Payloads	11
6.3	Semantics	12
6.3.1	audioObjectType	12
6.3.2	samplingFrequency.....	12
6.3.3	samplingFrequencyIndex.....	12
6.3.4	channelConfiguration.....	12
6.3.5	epConfig	12
6.4	Upstream	12
6.4.1	Introduction	12
6.4.2	Syntax	13
6.4.3	Definitions	13
6.4.4	Decoding process	14
6.5	MPEG-4 Audio transport stream.....	15
6.5.1	Overview	15
6.5.2	Synchronization Layer	15
6.5.3	Multiplex Layer	18
7	Parametric audio coding (HILN)	28
7.1	Overview of the tools	28
7.2	Terms and definitions	29
7.3	Bitstream syntax	29
7.3.1	Decoder configuration (ParametricSpecificConfig).....	29
7.3.2	Bitstream frame (alPduPayload).....	32
7.4	Bitstream semantics	52
7.4.1	Decoder configuration (ParametricSpecificConfig).....	52
7.4.2	Bitstream frame (alPduPayload).....	53
7.5	Parametric decoder tools.....	56
7.5.1	HILN decoder tools.....	56
7.5.2	Integrated parametric coder	75
8	Extension to General Audio Coding.....	76
8.1	Encoder and decoder block diagrams	76
8.2	Decoder configuration (GASpecificConfig).....	78
8.2.1	Syntax	78
8.2.2	Semantics	78

8.3	Fine granule audio (BSAC).....	79
8.3.1	Overview of tools.....	79
8.3.2	Bitstream syntax.....	79
8.3.3	General information	85
8.3.4	Tool description	104
8.4	Low delay coding mode	126
8.4.1	Introduction	126
8.4.2	Syntax	128
8.4.3	General information	130
8.4.4	Coder description.....	130
8.5	AAC Error resilience.....	135
8.5.1	Overview of tools.....	135
8.5.2	Bitstream payload	136
8.5.3	Tool descriptions.....	139
9	Error protection	152
9.1	Overview of the tools	152
9.2	Syntax	154
9.2.1	Error protection specific configuration	154
9.2.2	Error protection bitstream payloads	155
9.3	General information	157
9.3.1	Definitions.....	157
9.4	Tool description	160
9.4.1	Out-of-band information.....	160
9.4.2	Derivation of pre-defined sets	161
9.4.3	In-band information	162
9.4.4	Concatenation functionality.....	163
9.4.5	CRC	164
9.4.6	Systematic rate-compatible punctured convolutional (SRCPC) codes	165
9.4.7	Shortened Reed-Solomon codes.....	168
9.4.8	Recursive interleaving	170
9.4.9	Class reordered output	176
10	Error resilient bitstream payloads	176
10.1	Overview of the tools	176
10.2	CELP	177
10.2.1	Syntax	178
10.2.2	General information	188
10.2.3	Tool description	188
10.3	HVXC.....	188
10.3.1	Syntax	188
10.3.2	General information	205
10.3.3	Tool description	205
10.4	TwinVQ.....	205
10.4.1	Syntax	205
10.4.2	General information	209
10.4.3	Tool description	209
10.5	AAC	209
10.5.1	Syntax	209
10.5.2	General information	210
10.5.3	Tables	212
10.5.4	Figures.....	214
10.6	ER HILN.....	215
10.7	ER BSAC.....	215
11	Silence Compression Tool.....	216
11.1	Overview of the silence compression tool.....	216
11.2	Definitions.....	216
11.3	Specifications of the silence compression tool.....	217
11.3.1	Transmission payload.....	217
11.3.2	Bitrates of the silence compression tool	217

11.3.3	Algorithmic delay of the silence compression tool	218
11.4	Syntax	218
11.4.1	Bitstream syntax	219
11.4.2	Bitstream semantics	222
11.4.3	LSP transmission	223
11.5	CNG module	223
11.5.1	Definitions	223
11.5.2	LSP decoder	224
11.5.3	LSP smoother	224
11.5.4	LSP interpolation and LSP-LPC conversion	224
11.5.5	RMS decoder	225
11.5.6	RMS smoother	225
11.5.7	CNG excitation generation	225
11.5.8	LP Synthesis filter	228
11.5.9	Memory update	228
12	Extension of HVXC variable rate mode	228
12.1	Overview	228
12.2	Definitions	229
12.3	Syntax	229
12.3.1	Decoder configuration (ER HvxcSpecificConfig)	229
12.3.2	Bitstream frame (aIPduPayload)	230
12.4	Specification of variable rate mode	231
12.4.1	Transmission payload	231
12.4.2	Bitrates of variable rate mode	232
12.5	Decoding process of variable rate mode	232
12.5.1	Decoding process	232
Annex A	(informative) Parametric audio encoder	234
A.1	Overview of the encoder tools	234
A.2	HILN encoder tools	234
A.2.1	HILN parameter extraction	234
A.2.2	HILN parameter encoder	238
A.2.3	HILN bitrate scalability	241
A.3	Music/Speech Mixed Encoder tool	241
A.3.1	Music/Speech classification tool	242
A.3.2	Integrated parametric coder	243
Annex B	(informative) Fine grain scalability tool: BSAC (Bit-Sliced Arithmetic Coding)	245
B.1	Introduction	245
B.2	Bit Slicing of the quantized spectral coefficients	245
B.3	Probability Table Determinations	246
B.4	Grouping and interleaving	246
B.5	Scalefactors	247
B.6	Arithmetic coding	247
B.6.1	Arithmetic Coding Procedure	248
B.7	Stereo-related data and PNS data	249
B.8	Payload transmitted over Elementary Stream	250
Annex C	(informative) Error protection tool	253
C.1	Text format of out-of-band information	253
C.2	Example of out-of-band information	254
C.2.1	Example for AAC	254
C.2.2	Example for Twin-VQ	254
C.2.3	Example for CELP	260
C.2.4	Example for HVXC	262
C.2.5	Example for ER BSAC	263
C.3	Example of error concealment	264
C.3.1	Example for CELP	264
C.3.2	Error concealment for the silence compression tool	269
C.4	Example of EP tool setting and error concealment for HVXC	269
C.4.1	Definitions	270

C.4.2	Channel coding	270
C.4.3	EP tool setting	281
C.4.4	Error concealment.....	282
Annex D	(informative) Silence compression tool.....	285
D.1	VAD module.....	285
D.1.1	Definitions.....	285
D.1.2	Down-sampling for the wideband	285
D.1.3	Parameter calculation	286
D.1.4	Temporal voice activity decision	286
D.1.5	Frame voice activity decision	287
D.1.6	Hangover	287
D.2	DTX module	288
D.2.1	Definitions.....	288
D.2.2	LP analysis	289
D.2.3	Averaging of the LSP	289
D.2.4	RMS calculation.....	289
D.2.5	Averaging of the RMS	289
D.2.6	Detection of the characteristics change.....	289
D.2.7	Parameter encoding	290
D.2.8	Local CNG decoder	290
Annex E	(informative) Extension of HVXC variable rate mode.....	291
E.1	Encoder Description.....	291
E.1.1	Definitions.....	291
E.1.2	RMS computation.....	291
E.1.3	Spectral comparison	291
E.1.4	Signal stability detection.....	292
E.1.5	Background noise detection.....	292
E.1.6	Transmitted parameters decision	292
Annex F	(informative) Patent statements	296

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. 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 Amendment may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Amendment 1 to International Standard ISO/IEC 14496-3:2000 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Annexes A, B, C, D, E and F of this Amendment are for information only.

Introduction

MPEG-4 version 2 is an amendment to MPEG-4 version 1. This document contains the description of bitstream and decoder extensions related to new tools defined within MPEG-4 version 2. As long as nothing else is mentioned, the description made in MPEG-4 version 1 is not changed but only extended.

Overview

ISO/IEC 14496-3 (MPEG-4 Audio) is a new kind of audio standard that integrates many different types of audio coding: natural sound with synthetic sound, low bitrate delivery with high-quality delivery, speech with music, complex soundtracks with simple ones, and traditional content with interactive and virtual-reality content. By standardizing individually sophisticated coding tools as well as a novel, flexible framework for audio synchronization, mixing, and downloaded post-production, the developers of the MPEG-4 Audio standard have created new technology for a new, interactive world of digital audio.

MPEG-4, unlike previous audio standards created by ISO/IEC and other groups, does not target a single application such as real-time telephony or high-quality audio compression. Rather, MPEG-4 Audio is a standard that applies to every application requiring the use of advanced sound compression, synthesis, manipulation, or playback. The subparts that follow specify the state-of-the-art coding tools in several domains; however, MPEG-4 Audio is more than just the sum of its parts. As the tools described here are integrated with the rest of the MPEG-4 standard, exciting new possibilities for object-based audio coding, interactive presentation, dynamic soundtracks, and other sorts of new media, are enabled.

Since a single set of tools is used to cover the needs of a broad range of applications, *interoperability* is a natural feature of systems that depend on the MPEG-4 Audio standard. A system that uses a particular coder—for example, a real-time voice communication system making use of the MPEG-4 speech coding toolset—can easily share data and development tools with other systems, even in different domains, that use the same tool—for example, a voicemail indexing and retrieval system making use of MPEG-4 speech coding.

The following subclauses give a more detailed overview of the capabilities and functionalities provided with MPEG-4 Audio version 2.

New concepts

With this extension, new tools are added to the MPEG-4 standard, while none of the existing tools of version 1 is replaced. Version 2 is therefore fully backward compatible to version 1.

In the area of Audio, new tools are added in MPEG-4 version 2 to provide the following new functionalities:

- Error Robustness

The error robustness tools provide improved performance on error-prone transmission channels. They can be distinguished into codec specific error resilience tools and an common error protection tool.

Improved error robustness for AAC is provided by a set of error resilience tools. These tools reduce the perceived deterioration of the decoded audio signal that is caused by corrupted bits in the bitstream. The following tools are provided to improve the error robustness for several parts of an AAC frame:

- Virtual CodeBook tool (VCB11)
- Reversible Variable Length Coding tool (RVLC)
- Huffman Codeword Reordering tool (HCR)

Improved error robustness capabilities for all coding tools are provided through the error resilient bitstream payload syntax. It allows advanced channel coding techniques, which can be adapted to the special needs of the different coding tools. This error resilient bitstream payload syntax is mandatory for all version 2 object types.

The error protection tool (EP tool) provides unequal error protection (UEP) for MPEG-4 Audio in conjunction with the error resilient bitstream payload. UEP is an efficient method to improve the error robustness of source coding schemes. It is used by various speech and audio coding systems operating over error-prone channels such as mobile telephone networks or Digital Audio Broadcasting (DAB). The bits of the coded signal representation are first grouped into different classes according to their error sensitivity. Then error protection is individually applied to the different classes, giving better protection to more sensitive bits.

- Low-Delay Audio Coding

The MPEG-4 General Audio Coder provides very efficient coding of general audio signals at low bitrates. However it has an algorithmic delay of up to several 100ms and is thus not well suited for applications requiring low coding delay, such as real-time bi-directional communication. As an example, for the General Audio Coder operating at 24 kHz sampling rate and 24 kbit/s this results in an algorithmic coding delay of about 110 ms plus up to additional 210 ms for the bit reservoir. To enable coding of general audio signals with an algorithmic delay not exceeding 20 ms, MPEG-4 version 2 specifies a Low-Delay Audio Coder which is derived from MPEG-2/4 Advanced Audio Coding (AAC). It operates at up to 48 kHz sampling rate and uses a frame length of 512 or 480 samples, compared to the 1024 or 960 samples used in standard MPEG-2/4 AAC. Also the size of the window used in the analysis and synthesis filterbank is reduced by a factor of 2. No block switching is used to avoid the "look-ahead" delay due to the block switching decision. To reduce pre-echo artefacts in case of transient signals, window shape switching is provided instead. For non-transient parts of the signal a sine window is used, while a so-called low overlap window is used in case of transient signals. Use of the bit reservoir is minimized in the encoder in order to reach the desired target delay. As one extreme case, no bit reservoir is used at all.

- Fine Grain Scalability

Bitrate scalability, also known as embedded coding, is a very desirable functionality. The General Audio Coder of version 1 supports large step scalability where a base layer bitstream can be combined with one or more enhancement layer bitstreams to utilize a higher bitrate and thus obtain a better audio quality. In a typical configuration, a 24 kbit/s base layer and two 16 kbit/s enhancement layers could be used, permitting decoding at a total bitrate of 24 kbit/s (mono), 40 kbit/s (stereo), and 56 kbit/s (stereo). Due to the side information carried in each layer, small bitrate enhancement layers are not efficiently supported in version 1. To address this problem and to provide efficient small step scalability for the General Audio Coder, the Bit-Sliced Arithmetic Coding (BSAC) tool is available in version 2. This tool is used in combination with the AAC coding tools and replaces the noiseless coding of the quantized spectral data and the scalefactors. BSAC provides scalability in steps of 1 kbit/s per audio channel, i.e. 2 kbit/s steps for a stereo signal. One base layer bitstream and many small enhancement layer bitstreams are used. The base layer contains the general side information, specific side information for the first layer and the audio data of the first layer. The enhancement streams contain only the specific side information and audio data for the corresponding layer. To obtain fine step scalability, a bit-slicing scheme is applied to the quantized spectral data. First the quantized spectral values are grouped into frequency bands. Each of these groups contains the quantized spectral values in their binary representation. Then the bits of a group are processed in slices according to their significance. Thus first all most significant bits (MSB) of the quantized values in a group are processed, etc. These bit-slices are then encoded using an arithmetic coding scheme to obtain entropy coding with minimal redundancy. Various arithmetic coding models are provided to cover the different statistics of the bit-slices. The scheme used to assign the bit-slices of the different frequency bands to the enhancement layer is constructed in a special way. This ensures that, with an increasing number of enhancement layers utilized by the decoder, quantized spectral data is refined by providing more of the less significant bits. But also the bandwidth is increased by providing bit-slices of the spectral data in higher frequency bands.

- Parametric Audio Coding

The Parametric Audio Coding tools combine very low bitrate coding of general audio signals with the possibility of modifying the playback speed or pitch during decoding without the need for an effects processing unit. In combination with the speech and audio coding tools of version 1, improved overall coding efficiency is expected for applications of object based coding allowing selection and/or switching between different coding techniques.

Parametric Audio Coding uses the Harmonic and Individual Lines plus Noise (HILN) technique to code general audio signals at bitrates of 4 kbit/s and above using a parametric representation of the audio signal. The basic idea of this technique is to decompose the input signal into audio objects which are described by appropriate source models and represented by model parameters. Object models for sinusoids, harmonic tones, and noise are utilized in the HILN coder.

This approach allows to introduce a more advanced source model than just assuming a stationary signal for the duration of a frame, which motivates the spectral decomposition used e.g. in the MPEG-4 General Audio Coder. As known from speech coding, where specialized source models based on the speech generation process in the human vocal tract are applied, advanced source models can be advantageous in particular for very low bitrate coding schemes.

Due to the very low target bitrates, only the parameters for a small number of objects can be transmitted. Therefore a perception model is employed to select those objects that are most important for the perceptual quality of the signal.

In HILN, the frequency and amplitude parameters are quantized according to the “just noticeable differences” known from psychoacoustics. The spectral envelope of the noise and the harmonic tone is described using LPC modeling as known from speech coding. Correlation between the parameters of one frame and between consecutive frames is exploited by parameter prediction. The quantized parameters are finally entropy coded and multiplexed to form a bitstream.

A very interesting property of this parametric coding scheme arises from the fact that the signal is described in terms of frequency and amplitude parameters. This signal representation permits speed and pitch change functionality by simple parameter modification in the decoder. The HILN Parametric Audio Coder can be combined with MPEG-4 Parametric Speech Coder (HVXC) to form an integrated parametric coder covering a wider range of signals and bitrates. This integrated coder supports speed and pitch change. Using a speech/music classification tool in the encoder, it is possible to automatically select the HVXC for speech signals and the HILN for music signals. Such automatic HVXC/HILN switching was successfully demonstrated and the classification tool is described in the informative Annex of the version 2 standard.

- CELP Silence Compression

The silence compression tool reduces the average bitrate thanks to compression at a lower-bitrate for silence. In the encoder, a voice activity detector is used to distinguish between regions with normal speech activity and those with silence or background noise. During normal speech activity, the CELP coding as in version 1 is used. Otherwise a Silence Insertion Descriptor (SID) is transmitted at a lower bitrate. This SID enables a Comfort Noise Generator (CNG) in the decoder. The amplitude and the spectral shape of this comfort noise are specified by energy and LPC parameters in similar methods to those in a normal CELP frame. These parameters are optionally re-transmitted in the SID and thus can be updated as required.

- Extended HVXC

In the Version 1 HVXC, variable bitrate mode of 2.0 kbit/s maximum is supported as well as 2.0 and 4.0 kbit/s fixed bitrate modes. In the Version 2 Error Resilient (ER) HVXC, the variable bitrate mode of 4.0 kbit/s maximum is additionally supported. The ER HVXC therefore provides fixed bitrate modes (2.0-4.0kbit/s) and variable bitrate mode(<2.0kbit/s, <4.0kbit/s) both in a scalable and non-scalable scheme. In the variable bitrate modes, non-speech parts are detected in unvoiced signals, and a smaller number of bits is used for these non-speech parts to reduce the average bitrate. ER HVXC provides communications-quality to near-toll-quality speech in the 100-3800 Hz band at 8kHz sampling rate. When the variable bitrate mode is allowed, operation at lower average bitrate is possible. Coded speech with variable bitrate mode at typical bitrate of 1.5kbit/s average, and at typical bitrate of 3.0kbit/s average has essentially the same quality as 2.0 kbit/s fixed rate and 4.0 kbit/s fixed rate respectively. The functionality of pitch and speed change during decoding is supported for all modes. ER HVXC has the syntax with the error sensitivity classes to be used with the EP-Tool, and the error concealment functionality is supported for the use for error-prone channel like mobile communication channels. The ER HVXC speech coder targets applications from mobile and satellite communications, to Internet telephony, to packaged media and speech databases.

Capabilities

Overview of capabilities

MPEG-4 Audio version 2 provides the following new capabilities:

- error robustness (including error resilience as well as error protection)
- low delay audio coding
- backchannel
- fine granule scalability
- parametric audio
- silence compression in CELP
- extended HVXC

Those new capabilities are discussed in more detail below.

Error robustness

Error resilience tools for AAC

Several tools are provided to increase the error resilience for AAC. These tools improve the perceptual audio quality of the decoded audio signal in case of corrupted bitstreams, which may occur e. g. in the presence of noisy transmission channels.

The Virtual CodeBooks tool (VCB11) extends the sectioning information of an AAC bitstream. This permits to detect serious errors within the spectral data of an MPEG-4 AAC bitstream. Virtual codebooks are used to limit the largest absolute value possible within a certain scalefactor band where escape values are. While referring to the same codes as codebook 11, the sixteen virtual codebooks introduced by this tool provide sixteen different limitations of the spectral values belonging to the corresponding section. Due to this, errors within spectral data resulting in spectral values exceeding the indicated limit can be located and appropriately concealed.

The Reversible Variable Length Coding tool (RVLC) replaces the Huffman and DPCM coding of the scalefactors in an AAC bitstream. The RVLC uses symmetric codewords to enable both forward and backward decoding of the scalefactor data. In order to have a starting point for backward decoding, the total number of bits of the RVLC part of the bitstream is transmitted. Because of the DPCM coding of the scalefactors, also the value of the last scalefactor is transmitted to enable backward DPCM decoding. Since not all nodes of the RVLC code tree are used as codewords, some error detection is also possible.

The Huffman codeword reordering (HCR) algorithm for AAC spectral data is based on the fact that some of the codewords can be placed at known positions so that these codewords can be decoded independent of any error within other codewords. Therefore, this algorithm avoids error propagation to those codewords, the so-called priority codewords (PCW). To achieve this, segments of known length are defined and those codewords are placed at the beginning of these segments. The remaining codewords (non-priority codewords, non-PCW) are filled into the gaps left by the PCWs using a special algorithm that minimizes error propagation to the non-PCWs codewords. This reordering algorithm does not increase the size of spectral data. Before applying the reordering algorithm itself, a pre-sorting process is applied to the codewords. It sorts all codewords depending on their importance, i. e. it determines the PCWs.

Error protection

The EP tool provides unequal error protection. It receives several classes of bits from the audio coding tools, and then applies forward error correction codes (FEC) and/or cyclic redundancy codes (CRC) for each class, according to its error sensitivity.

The error protection tool (EP tool) provides the unequal error protection (UEP) capability to the ISO/IEC 14496-3 codecs. Main features of this tool are:

- providing a set of error correcting/detecting codes with wide and small-step scalability, in performance and in redundancy
- providing a generic and bandwidth-efficient error protection framework, which covers both fixed-length frame bitstreams and variable-length frame bitstreams
- providing a UEP configuration control with low overhead

Error resilient bitstream reordering

Error resilient bitstream reordering allows the effective use of advanced channel coding techniques like unequal error protection (UEP), that can be perfectly adapted to the needs of the different coding tools. The basic idea is to rearrange the audio frame content depending on its error sensitivity in one or more instances belonging to different error sensitivity categories (ESC). This rearrangement works either data element-wise or even bit-wise. An error resilient bitstream frame is build by concatenating these instances.

Low delay

The low delay coding functionality 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 contrast to traditional low delay coders based on speech coding technology, the concept of this low delay coder is based on general perceptual audio coding and is thus suitable for a wide range of audio signals. Specifically, it is derived closely from the proven architecture of MPEG-2/4 Advanced Audio Coding (AAC). Furthermore, all capabilities for coding of 2 (stereo) or more sound channels (multi-channel) are available within the low delay coder as inherited from Advanced Audio Coding.

Upstream

To allow for user on a remote side to dynamically control the streaming of the server, backchannel streams carrying user interaction information are defined.

Fine granule scalability in audio

BSAC provides fine grain scalability in steps of 1 kbit/s per audio channel, i.e. 2 kbit/s steps for a stereo signal. One base layer bitstream and many small enhancement layer bitstreams are used. In order to implement the fine grain scalability efficiently in MPEG-4 system, the fine grain audio data can be divided into the large-step layers and the large-step layers are concatenated from the several sub-frames. And the configuration of the payload transmitted over Elementary Stream (ES) can be changed dynamically depending on the environment such as the network traffic or the user interaction. So, BSAC can allow for real-time adjustments to the quality of service.

In addition to fine grain scalability, it can improve the quality of the audio signal which is decoded from the bitstreams transmitted over error-prone channels such as mobile communication networks or Digital Audio Broadcasting (DAB)

HILN: harmonic and individual lines plus noise (parametric audio coding)

MPEG-4 parametric audio coding uses the HILN technique (Harmonic and Individual Lines plus Noise) to code non-speech signals like music at bitrates of 4 kbit/s and higher using a parametric representation of the audio signal. HILN allows independent change of speed and pitch during decoding. Furthermore HILN can be combined with MPEG-4 parametric speech coding (HVXC) to form an integrated parametric coder covering a wider range of signals and bitrates.

Silence compression for CELP

The silence compression tool comprises a Voice Activity Detector (VAD), a Discontinuous Transmission (DTX) unit and a Comfort Noise Generator (CNG) module. The tool encodes/decodes the input signal at a lower bitrates

during the non-active-voice (silent) frames. During the active-voice (speech) frames, MPEG-4 CELP encoding and decoding are used.

Extension of HVXC

The operation of maximum of 4.0 kbit/s variable bitrate mode of the MPEG-4 parametric speech coder HVXC is provided in addition to the Version1 HVXC functionalities including 2.0-4.0 kbit/s fixed bitrate mode and 2.0 kbit/s maximum variable bitrate mode. Here extended operation of the variable bitrate mode with 4.0 kbit/s maximum is provided which allows higher quality variable rate coding.

Information technology — Coding of audio-visual objects —

Part 3: Audio

Amendment 1: Audio extensions

1 Scope

ISO/IEC 14496-3 (MPEG-4 Audio) is a new kind of audio standard that integrates many different types of audio coding: natural sound with synthetic sound, low bitrate delivery with high-quality delivery, speech with music, complex soundtracks with simple ones, and traditional content with interactive and virtual-reality content. By standardizing individually sophisticated coding tools as well as a novel, flexible framework for audio synchronization, mixing, and downloaded post-production, the developers of the MPEG-4 Audio standard have created new technology for a new, interactive world of digital audio.

MPEG-4, unlike previous audio standards created by ISO/IEC and other groups, does not target a single application such as real-time telephony or high-quality audio compression. Rather, MPEG-4 Audio is a standard that applies to every application requiring the use of advanced sound compression, synthesis, manipulation, or playback. The subparts that follow specify the state-of-the-art coding tools in several domains; however, MPEG-4 Audio is more than just the sum of its parts. As the tools described here are integrated with the rest of the MPEG-4 standard, exciting new possibilities for object-based audio coding, interactive presentation, dynamic soundtracks, and other sorts of new media, are enabled.

Since a single set of tools is used to cover the needs of a broad range of applications, *interoperability* is a natural feature of systems that depend on the MPEG-4 Audio standard. A system that uses a particular coder—for example, a real-time voice communication system making use of the MPEG-4 speech coding toolset—can easily share data and development tools with other systems, even in different domains, that use the same tool—for example, a voicemail indexing and retrieval system making use of MPEG-4 speech coding.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this Amendment. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this Amendment are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO/IEC 11172-3:1993, *Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s - Part 3: Audio*.

ITU-T Rec.H.222.0(1995) | ISO/IEC 13818-1:1996, *Information technology – Generic coding of moving pictures and associated audio information: Systems*.

ISO/IEC 13818-3:1998, *Information technology – Generic coding of moving pictures and associated audio information - Part 3: Audio*.

ISO/IEC 13818-7:1997, *Information technology – Generic coding of moving pictures and associated audio information - Part 7: Advanced Audio Coding (AAC)*.

ITU-T Recommendation H.223 (1998), Annex C, Multiplexing protocol for low bit-rate multimedia communication over high error-prone channels.

3 Terms and definitions

3.1 virtual codebook – If several codebook values refer to one and the same physical codebook, these values are called virtual codebooks.

See also ISO/IEC 14496-3 Subpart 1 Main.

4 Symbols and abbreviations

RVLC – Reversible Variable Length Coding

VCB11 – Virtual Codebooks for codebook 11

EP – Error Protection

ER – Error Resilience or Error Resilient (as appropriate)

HCR – Huffman Codeword Reordering

HILN – Harmonic and Individual Lines plus Noise

BSAC – Bit Sliced Arithmetic Coding

See also ISO/IEC 14496-3 Subpart 1 Main.

5 Technical overview

5.1 Extended MPEG-4 audio object types

5.1.1 Audio object type definition

Table 1: Audio Object Type definition

Tools	13818-7 main	13818-7 LC	13818-7 SSR	PNS	LTP	TLSS	Twin VQ	CELP	HVXC	TTSI	SA tools	SASBF	MIDI	BSAC	HILN	Low Delay AAC	HVXC 4kbs VR	Silence Compression	Error Robust	Hierarchy	Object Type ID
Null																					0
AAC main	X			X																contains AAC LC	1
AAC LC		X		X																	2
AAC SSR			X	X																	3
AAC LTP		X		X	X															contains AAC LC	4

Note: The error-robust tool is composed of error resilience and error protection. With respect of the decoder it is mandatory to equip the bit parsing function for error protection. However, it is optional to have the error detection and correction function.

5.1.2.1 NULL object type

5.1.2.2 AAC Main object type

5.1.2.3 AAC Low Complexity (LC) Object type

See ISO/IEC 14496-3 Subpart 1 Main.

5.1.2.4 AAC Scalable Sampling Rate (SSR) object type

See ISO/IEC 14496-3 Subpart 1 Main.

5.1.2.5 AAC Long Term Predictor (LTP) object type

See ISO/IEC 14496-3 Subpart 1 Main.

5.1.2.6 AAC Scalable object type

See ISO/IEC 14496-3 Subpart 1 Main.

5.1.2.7 TwinVQ object type

See ISO/IEC 14496-3 Subpart 1 Main.

5.1.2.8 CELP object type

See ISO/IEC 14496-3 Subpart 1 Main.

5.1.2.9 HVXC object type

See ISO/IEC 14496-3 Subpart 1 Main.

5.1.2.10 TTSI object type

See ISO/IEC 14496-3 Subpart 1 Main.

5.1.2.11 Main Synthetic object type

See ISO/IEC 14496-3 Subpart 1 Main.

5.1.2.12 Wavetable Synthesis object type

See ISO/IEC 14496-3 Subpart 1 Main.

5.1.2.13 General MIDI object type

See ISO/IEC 14496-3 Subpart 1 Main.

5.1.2.14 Algorithmic Synthesis and Audio FX object type

See ISO/IEC 14496-3 Subpart 1 Main.

5.1.2.15 Error Resilient (ER) AAC Low Complexity (LC) object type

The Error Resilient (ER) MPEG-4 AAC Low Complexity object type is the counterpart to the MPEG-4 AAC Low Complexity object, with additional error resilient functionality.

5.1.2.16 Error Resilient (ER) AAC Long Term Predictor (LTP) object type

The Error Resilient (ER) MPEG-4 AAC LTP object type is the counterpart to the MPEG-4 AAC LTP object, with additional error resilient functionality.

5.1.2.17 Error Resilient (ER) AAC scalable object type

The Error Resilient (ER) MPEG-4 AAC scalable object type is the counterpart to the MPEG-4 AAC scalable object, with additional error resilient functionality.

5.1.2.18 Error Resilient (ER) TwinVQ object type

The Error Resilient (ER) TwinVQ object type is the counterpart to the MPEG-4 TwinVQ object, with additional error resilient functionality.

5.1.2.19 Error Resilient (ER) BSAC object type

The ER BSAC object is supported by the fine grain scalability tool (BSAC: Bit-Sliced Arithmetic Coding). It provides error resilience as well as fine step scalability in the MPEG-4 General Audio (GA) coder. It is used in combination with the AAC coding tools and replaces the noiseless coding and the bitstream formatting of MPEG-4 version 1 GA coder. A large number of scalable layers are available, providing 1 kbit/s/ch enhancement layer, i.e. 2 kbit/s steps for a stereo signal.

5.1.2.20 Error Resilient (ER) AAC LD object type

The AAC LD object is supported by the low delay AAC coding tool. It also permits combinations with the PNS tool and the LTP tool. AAC LD object 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).

5.1.2.21 Error Resilient (ER) CELP object type

The ER CELP object is supported by silence compression and ER tools. It provides the ability to reduce the average bitrate thanks to a lower-bitrate compression for silence, with additional error resilient functionality.

5.1.2.22 Error Resilient (ER) HVXC object type

The ER HVXC object is supported by the parametric speech coding (HVXC) tools, which provide fixed bitrate modes (2.0-4.0 kbit/s) and variable bitrate modes (< 2.0 kbit/s and < 4.0 kbit/s) both in a scalable and a non-scalable scheme, and the functionality of pitch and speed change. The syntax to be used with the EP-Tool, and the error concealment functionality are supported for the use for error-prone channels. Only 8 kHz sampling rate and mono audio channel are supported.

5.1.2.23 Error Resilient (ER) HILN object type

The ER HILN object is supported by the parametric audio coding tools (HILN: Harmonic and Individual Lines plus Noise) which provide coding of general audio signals at very low bitrates ranging from below 4 kbit/s to above 16 kbit/s. Bitrate scalability and the functionality of speed and pitch change are available. The ER HILN object supports mono audio objects at a wide range of sampling rates.

5.1.2.24 Error Resilient (ER) Parametric object type

The ER Parametric object is supported by the parametric audio coding and speech coding tools HILN and HVXC. This integrated parametric coder combines the functionalities of the ER HILN and the ER HVXC objects. Only 8 kHz sampling rate and mono audio channel are supported.

5.2 Audio profiles and levels

5.2.1 Profiles

In ISO/IEC 14496-3 Amd 1, four new profiles are defined.

The **High Quality Audio Profile** contains the CELP speech coder and the Low Complexity AAC coder including Long Term Prediction. Scalable coding can be performed by the AAC Scalable object type. Optionally, the new error resilient (ER) bitstream syntax may be used. The **Low Delay Audio Profile** contains the HVXC and CELP speech coders (optionally using the ER bitstream syntax), the low-delay AAC coder and the Text-to-Speech interface TTSI. The **Natural Audio Profile** contains all natural audio coding tools available in MPEG-4. The **Mobile Audio Internetworking Profile** contains the low-delay and scalable AAC object types including TwinVQ and BSAC. This profile is intended to extend communication applications using non-MPEG speech coding algorithms with high quality audio coding capabilities.

Table 2: Audio Profiles definition

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

5.2.2 Complexity units

Complexity units are defined to give an approximation of the decoder complexity in terms of processing power and RAM usage required for processing MPEG-4 Audio bitstreams in dependence of specific parameters.

The approximated processing power is given in "Processor Complexity Units" (PCU), specified in integer numbers of MOPS. The approximated RAM usage is given in "RAM Complexity Units" (RCU), specified in mostly integer

numbers of kWords (1000 words). The RCU numbers do not include working buffers that can be shared between different objects and/or channels.

The following Table 3 gives complexity estimates for the different object types:

Table 3: Complexity of Audio Object Types

Object Type	Parameters	PCU (MOPS)	RCU (kWords)	Remarks
ER AAC LC	fs=48kHz	3	3	3)
ER AAC SSR	fs=48kHz	4	3	3)
ER AAC LTP	fs=48kHz	4	4	3)
ER AAC Scalable	fs=48kHz	5	4	3), 4)
ER TwinVQ	fs=24kHz	2	3	3)
ER BSAC	fs=48kHz (input buffer size=26000bits)	4	4	3)
	fs=48kHz (input buffer size=106000bits)	4	8	
ER AAC LD	fs=48kHz	3	2	3)
ER CELP	fs=8kHz	2	1	
	fs=16kHz	3	1	
ER HVXC	fs=8kHz	2	1	
ER HILN	fs=16kHz, ns=93	15	2	2)
	fs=16kHz, ns=47	8	2	
ER Parametric	fs=8kHz, ns=47	4	2	1), 2)

Definitions:

- fs = sampling frequency
- ns = max. number of sinusoids being synthesized, see below

Notes:

- 1) Parametric coder in HILN mode, for HVXC mode see ER HVXC.
- 2) PCU depends on fs and ns, see below
- 3) PCU proportional to sampling frequency
- 4) Includes core decoder

PCU for HILN:

The computational complexity of HILN depends on the sampling frequency fs and the maximum number of sinusoids ns to be synthesized simultaneously. The value of ns for a frame is the total number of harmonic and individual lines synthesized in that frame, i.e. the number of starting plus continued plus ending lines. For fs in kHz, the PCU in MOPS is calculated as follows:

$$PCU = (1 + 0.15 \cdot ns) \cdot fs / 16$$

The typical maximum values of ns are 47 for 6 kbit/s HILN and 93 for 16 kbit/s HILN bitstreams.

5.2.3 Levels within the profiles

A number of 0 stages of interleaving for the EP-tool indicates that the EP is not used in that particular level. The notation used to specify the number of audio channels indicates the number of full bandwidth channels and the

number of low-frequency enhancement channels. For example, “5.1” indicates 5 full bandwidth channels and one low-frequency enhancement channel.

Table 4: Levels for the High Quality Audio Profile

Level	Max. channels/object	Max. sampling rate [kHz]	Max PCU ^{*2}	Max RCU ^{*2}	EP-Tool: Max. redundancy by class FEC ^{*1}	EP-Tool: Max. # stages of interleaving per object
1	2	22.05	5	8	0 %	0
2	2	48	10	8	0 %	0
3	5.1	48	25	12 ^{*3}	0 %	0
4	5.1	48	100	42 ^{*3}	0 %	0
5	2	22.05	5	8	20%	9
6	2	48	10	8	20%	9
7	5.1	48	25	12 ^{*3}	20%	22
8	5.1	48	100	42 ^{*3}	20%	22

*1: This number does not cover FEC for the EP header, i. e. FEC for the EP header is always permitted. In case of several audio objects the limit is valid independently for each audio object. This value is the maximum redundancy for the Audio object, which has the longest frame length, in each profile & level.

*2: Level 5 to 8 do not include RAM and computational complexity for the EP tool.

*3: Sharing of work buffers between multiple objects or channel pair elements is assumed.

Table 5: Levels for the Low Delay Audio Profile

Level	Max. channels/object	Max. sampling rate [kHz]	Max PCU ^{*2}	Max RCU ^{*2}	EP-Tool: Max. redundancy by class FEC ^{*1}	EP-Tool: Max. # stages of interleaving per object
1	1	8	2	1	0 %	0
2	1	16	3	1	0 %	0
3	1	48	3	2	0 %	0
4	2	48	24	12 ^{*3}	0 %	0
5	1	8	2	1	100%	5
6	1	16	3	1	100%	5
7	1	48	3	2	20%	5
8	2	48	24	12 ^{*3}	20%	9

*1: This number does not cover FEC for the EP header, i. e. FEC for the EP header is always permitted. In case of several audio objects the limit is valid independently for each audio object. This value is the maximum redundancy for the Audio object, which has the longest frame length, in each profile & level.

*2: Level 5 to 8 do not include RAM and computational complexity for the EP tool.

*3: Sharing of work buffers between multiple objects or channel pair elements is assumed.

Table 6: Levels for the Natural Audio Profile

Level	Max. sampling rate [kHz]	Max PCU ^{*2}	EP-Tool: Max. redundancy by class FEC ^{*1}	EP-Tool: Max. # stages of interleaving per object
1	48	20	0 %	0
2	96	100	0 %	0
3	48	20	20%	9
4	96	100	20%	22

*1: This number does not cover FEC for the EP header, i. e. FEC for the EP header is always permitted. In case of several audio objects the limit is valid independently for each audio object. This value is the maximum redundancy for the Audio object, which has the longest frame length, in each profile & level.

*2: Level 3 and 4 do not include computational complexity for the EP tool.

No RCU limitations are specified for this profile.

Table 7: Levels for the Mobile Audio Internetworking Profile

Level	Max. channels/object	Max. sampling rate [kHz]	Max PCU ^{*3}	Max RCU ^{*2 *3}	Max. # audio objects	EP-Tool: Max. redundancy by class FEC ^{*1}	EP-Tool: Max. # stages of interleaving per object
1	1	24	2.5	4	1	0 %	0
2	2	48	10	8	2	0 %	0
3	5.1	48	25	12 ^{*4}	-	0 %	0
4	1	24	2.5	4	1	20%	5
5	2	48	10	8	2	20%	9
6	5.1	48	25	12 ^{*4}	-	20%	22

*1: This number does not cover FEC for the EP header, i. e. FEC for the EP header is always permitted. In case of several audio objects the limit is valid independently for each audio object. This value is the maximum redundancy for the Audio object, which has the longest frame length, in each profile & level.

*2: The maximum RCU for one channel in any object in this profile is 4. For the ER BSAC, this limits the input buffer size. The maximum possible input buffer size in bits for this case is given in PCU/RCU Table 3.

*3: Level 4 to 6 do not include RAM and computational complexity for the EP tool.

*4: Sharing of work buffers between multiple objects or channel pair elements are assumed.

6 Extension to interface to ISO/IEC 14496-1 (MPEG-4 Systems)

6.1 Introduction

The header streams are transported via MPEG-4 systems. These streams contain configuration information, which is necessary for the decoding process and parsing of the raw data streams. However, an update is only necessary if there are changes in the configuration.

The payloads contain all information varying on a frame to frame basis and therefore carry the actual audio information.

6.2 Extension to syntax

6.2.1 AudioSpecificInfo

Table 8: Syntax of AudioSpecificInfo()

Syntax	No. of bits	Mnemonic
AudioSpecificInfo ()		
{		
audioObjectType;	5	bslbf
samplingFrequencyIndex;	4	bslbf
if (samplingFrequencyIndex==0xf)		
samplingFrequency;	24	uimsbf
channelConfiguration;	4	bslbf
if (audioObjectType == 1 audioObjectType == 2		
audioObjectType == 3 audioObjectType == 4		
audioObjectType == 6 audioObjectType == 7)		
GASpecificConfig();		
if (audioObjectType == 8)		
CelpSpecificConfig();		
if (audioObjectType == 9)		
HvxcSpecificConfig();		
if (audioObjectType == 12)		
TTSSpecificConfig();		
if (audioObjectType == 13 audioObjectType == 14		
audioObjectType == 15 audioObjectType==16)		
StructuredAudioSpecificConfig();		
/* the following Objects are Amendment 1 Objects */		
if (audioObjectType == 17 audioObjectType == 19		
audioObjectType == 20 audioObjectType == 21		
audioObjectType == 22 audioObjectType == 23)		
GASpecificConfig();		
if (audioObjectType == 24)		
ErrorResilientCelpSpecificConfig();		
if (audioObjectType == 25)		
ErrorResilientHvxcSpecificConfig();		
if (audioObjectType == 26 audioObjectType == 27)		
ParametricSpecificConfig();		
if (audioObjectType == 17 audioObjectType == 19		
audioObjectType == 20 audioObjectType == 21		
audioObjectType == 22 audioObjectType == 23		
audioObjectType == 24 audioObjectType == 25		
audioObjectType == 26 audioObjectType == 27) {		
epConfig;	2	bslbf
if (epConfig == 2)		
ErrorProtectionSpecificConfig();		
}		
}		

6.2.1.1 HvxcSpecificConfig

Defined in ISO/IEC 14496-3 subpart 2.

6.2.1.2 CelpSpecificConfig

Defined in ISO/IEC 14496-3 subpart 3.

6.2.1.3 GASpecificConfig

Defined in subclause 8.2.1

6.2.1.4 StructuredAudioSpecificConfig

Defined in ISO/IEC 14496-3 subpart 5.

6.2.1.5 TTSSpecificConfig

Defined in ISO/IEC 14496-3 subpart 6.

6.2.1.6 ParametricSpecificConfig

Defined in subclause 7.3.1.

6.2.1.7 ErrorProtectionSpecificConfig

Defined in subclause 9.2.1.

6.2.1.8 ErrorResilientCelpSpecificConfig

Defined in subclause 11.4.

6.2.1.9 ErrorResilientHvxcSpecificConfig

Defined in subclause 12.3.1.

6.2.2 Payloads

For the NULL object the payload shall be 16 bit signed integer in the range from -32768 to +32767. The payloads for all other audio object types are defined in the corresponding parts. These are the basic entities to be carried by the systems transport layer. Note that for all natural audio coding schemes the output is scaled for a maximum of 32767/-32768. However, the MPEG-4 System compositor expects a scaling.

The following table shows an overview about where the Elementary Stream payloads for the error resilient Audio Object Types can be found and where the detailed syntax is defined.

Table 9: ER Audio Object Types

ER Audio Object Type	definition of elementary stream payloads	detailed syntax definition
ER AAC LC	subclause 10.5	ISO/IEC 14496-3 subpart 4 and subclause 8.5
ER AAC LTP	subclause 10.5	ISO/IEC 14496-3 subpart 4 and subclause 8.5
ER AAC scalable	subclause 10.5	ISO/IEC 14496-3 subpart 4 and subclause 8.5
ER Twin VQ	subclause 10.4	ISO/IEC 14496-3 subpart 4
ER BSAC	subclause 10.7	subclause 8.3.3.1
ER AAC LD	subclause 10.5	ISO/IEC 14496-3 subpart 4 and subclause 8.4
ER CELP	subclause 10.2	ISO/IEC 14496-3 subpart 3 and subclause 11.4
ER HVXC	subclause 10.3	ISO/IEC 14496-3 subpart 2 and subclause 12.3.2
ER HILN	subclause 10.6	subclause 7.3.2
ER Parametric	subclause 10.6	ISO/IEC 14496-3 subpart 2 and subclause 7.3.2

6.3 Semantics

6.3.1 audioObjectType

A five bit field indicating the audio object type. This is the master switch which selects the actual bitstream syntax of the audio data. In general, different object type use a different bitstream syntax. The interpretation of this field is given in Table 1

6.3.2 samplingFrequency

See ISO/IEC 14496-3 Subpart 1 Main.

6.3.3 samplingFrequencyIndex

See ISO/IEC 14496-3 Subpart 1 Main.

6.3.4 channelConfiguration

See ISO/IEC 14496-3 Subpart 1 Main.

6.3.5 epConfig

This variable signals what kind of error robust configuration is used, i. e. how instances of error sensitivity categories are obtained on decoder site.

Table 10: epConfig

epConfig	Description
0	All instances of all sensitivity categories belonging to one frame are stored within one access unit.
1	Each instance of each sensitivity category belonging to one frame is stored separately within a single access unit, i.e. there are as many elementary streams existent as instances defined within a frame.
2	The error protection decoder has to be applied. Its input is an error protected access unit and its output are several error protection class instances. Each instance of each sensitivity category belonging to one frame corresponds to one of these error protection class instances.
3	Reserved

6.4 Upstream

6.4.1 Introduction

Upstreams are defined to allow for user on a remote side to dynamically control the streaming of the server.

The need for an up-stream channel is signaled to the client terminal by supplying an appropriate elementary stream descriptor declaring the parameters for that stream. The client terminal opens this up-stream channel in a similar manner as it opens the downstream channels. The entities (e.g. media encoders & decoders) that are connected through an up-stream channel are known from the parameters in its elementary stream descriptor and from the association of the elementary stream descriptor to a specific object descriptor.

An up-stream can be associated to a single downstream or a group of down streams. The stream type of the downstream to which the up-stream is associated defines the scope of the up-stream. When the up-stream is associated to a single downstream it carries messages about the downstream it is associated to. The syntax and semantics of messages for MPEG-4 Audio are defined in the next subclause.

6.4.2 Syntax

Table 11: Syntax of AudioUpstreamPayload()

Syntax	No. of bits	Mnemonic
AudioUpstreamPayload() {		
upStreamType ;	4	uimsbf
switch (upStreamType) {		
case 0: /* scalability control */		
numOfLayer ;	6	uimsbf
for (layer = 0; layer < numOfLayer; layer++) {		
avgBitrate[layer] ;	24	uimsbf
}		
break;		
case 1: /* BSAC frame interleaving */		
numOfSubFrame ;	5	uimsbf
break;		
case 2: /* quality feedback */		
multiLayOrSynEle ;	1	uimsbf
if (multiLayOrSynEle) {		
layOrSynEle ;	6	uimsbf
}		
else {		
layOrSynEle = 1;		
}		
numFrameExp[layOrSynEle] ;	4	uimsbf
lostFrames[layOrSynEle] ;	numFrameExp [layOrSynEle]	uimsbf
break;		
case 3: /* bitrate control */		
avgBitrate ;	24	uimsbf
break;		
default: /* reserved for future use */		
break;		
}		
}		

6.4.3 Definitions

upStreamType A 4-bit unsigned integer value representing the type of the up-stream as defined in the following Table 12

Table 12: Definition of upStreamType

UpStreamType	Type of Audio up-stream
0	scalability control
1	BSAC frame interleaving
2	quality feedback
3	bitrate control
4 – 15	reserved for future use

avgBitrate[layer] The average bitrate in bits per second of a large step layer, which the client requests to be transmitted from the server.

numOfSubFrame	A 5-bit unsigned integer value representing the number of the frames which are grouped and transmitted in order to reduce the transmission overhead. The transmission overhead is decreased but the delay is increased as numOfSubFrame is increased.
multiLayOrSynEle	This bit signals, whether or not a multi-channel or multi-layer configuration is used. Only in that case a layer number or a syntactic element number needs to be transmitted.
layOrSynEle	A 6-bit unsigned integer value representing the number of the syntactic element (in case of multi-channel setup) or the number of the layer (in case of multi-layer setup), to which the following quality feedback information belongs. This number refers to one of the layers or one of the syntactic elements contained within the associated Audio object. If the Audio object does neither support scalability nor multi-channel capabilities, this value is implicitly set to 1.
numFrameExp[layOrSynEle]	This value indicates the number of last recently passed frames $(2^{\text{numFrameExp}} - 1)$ considered in the following lostFrames value.
lostFrames[layOrSynEle]	This field contains the number of lost frames with respect to the indicated layer or syntactic element within the last recently passed frames signalled by numFrameExp.
avgBitrate	The average bitrate in bits per second of the whole Audio object, which the client requests to be transmitted from the server.

6.4.4 Decoding process

First, **upStreamType** is parsed which represents the type of the up-stream. The remaining decoding process depends upon the type of the up-stream.

6.4.4.1 Decoding of scalability control

Next is the value **numOfLayer**. It represents the number of the data elements **avgBitrate** to be read. **avgBitrate** follows.

6.4.4.2 Decoding of BSAC frame interleaving

The data element to be read is **numOfSubFrame**. It represents the number of the sub-frames to be interleaved in BSAC tool. BSAC can allow for runtime adjustments to the quality of service. When the content of upstream is transmitted from the client to the server to implement a stream dynamically and interactively. BSAC data are split and interleaved in the server. The detailed process for implementing an AU payload in the server is described in the clause 'informative Annex: Encoder' of 14496-3 Amd 1.

6.4.4.3 Decoding of quality feedback

The real frame loss rate in percent can be derived using the following formula:

$$\text{frameLossRate}[\text{layOrSynEle}] = \frac{\text{lostFrames}[\text{layOrSynEle}]}{2^{\text{numFrameExp}[\text{layOrSynEle}] - 1}} * 100\%$$

6.4.4.4 Decoding of bitrate control

avgBitrate is parsed.

6.5 MPEG-4 Audio transport stream

6.5.1 Overview

This subclause defines a mechanism to transport ISO/IEC 14496-3 (MPEG-4 Audio) streams without using ISO/IEC 14496-1 (MPEG-4 Systems) for audio-only applications. Figure 1 shows the concept of MPEG-4 Audio transport. The transport mechanism uses a two-layer approach, namely a multiplex layer and a synchronization layer. The multiplex layer (Low-overhead MPEG-4 Audio Transport Multiplex: LATM) manages multiplexing of several MPEG-4 Audio payloads and `AudioSpecificInfo()` elements. The synchronization layer specifies a self-synchronized syntax of the MPEG-4 Audio transport stream which is called Low Overhead Audio Stream (LOAS). The Interface format to a transmission layer depends on the conditions of the underlying transmission layer as follows:

- LOAS shall be used for the transmission over channels where no frame synchronization is available.
- LOAS may be used for the transmission over channels with fixed frame synchronization.
- A multiplexed element (`AudioMuxElement()`/`EPMuxElement()`) without synchronization shall be used only for transmission channels where an underlying transport layer already provides frame synchronization that can handle arbitrary frame size.

The details in the LOAS format and the `AudioMuxElement()` format are described in subclauses 6.5.2 and 6.5.3, respectively.

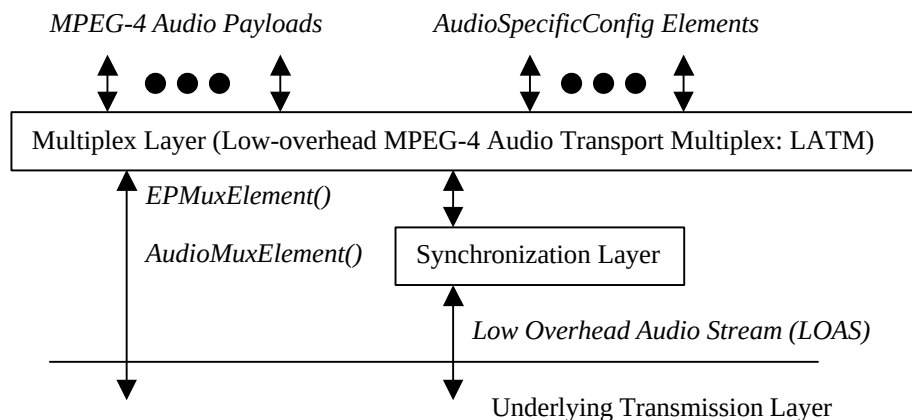


Figure 1: Concept of MPEG-4 Audio Transport

6.5.2 Synchronization Layer

The synchronization layer provides the multiplexed element with a self-synchronized mechanism to generate LOAS. The LOAS has three different types of format, namely `AudioSyncStream()`, `EPAudioSyncStream()` and `AudioPointerStream()`. The choice for one of the three formats is dependent on the underlying transmission layer.

- `AudioSyncStream()`

`AudioSyncStream()` consists of a syncword, the multiplexed element with byte alignment, and its length information. The maximum byte-distance between two syncwords is 8192 bytes. This self-synchronized stream shall be used for the case that the underlying transmission layer comes without any frame synchronization.

- EPAudioSyncStream()

For error prone channels, an alternative version to AudioSyncStream() is provided. This format has the same basic functionality as the previously described AudioSyncStream(). However, it additionally provides a longer syncword and a frame counter to detect lost frames. The length information and the frame counter are additionally protected by a FEC code.

- AudioPointerStream()

AudioPointerStream() shall be used for applications using a underlying transmission layer with fixed frame synchronization, where transmission framing can not be synchronized with the variable length multiplexed element. Figure 2 shows synchronization in AudioPointerStream(). This format utilizes a pointer indicating the start of the next multiplex element in order to synchronize the variable length payload with the constant transmission frame.

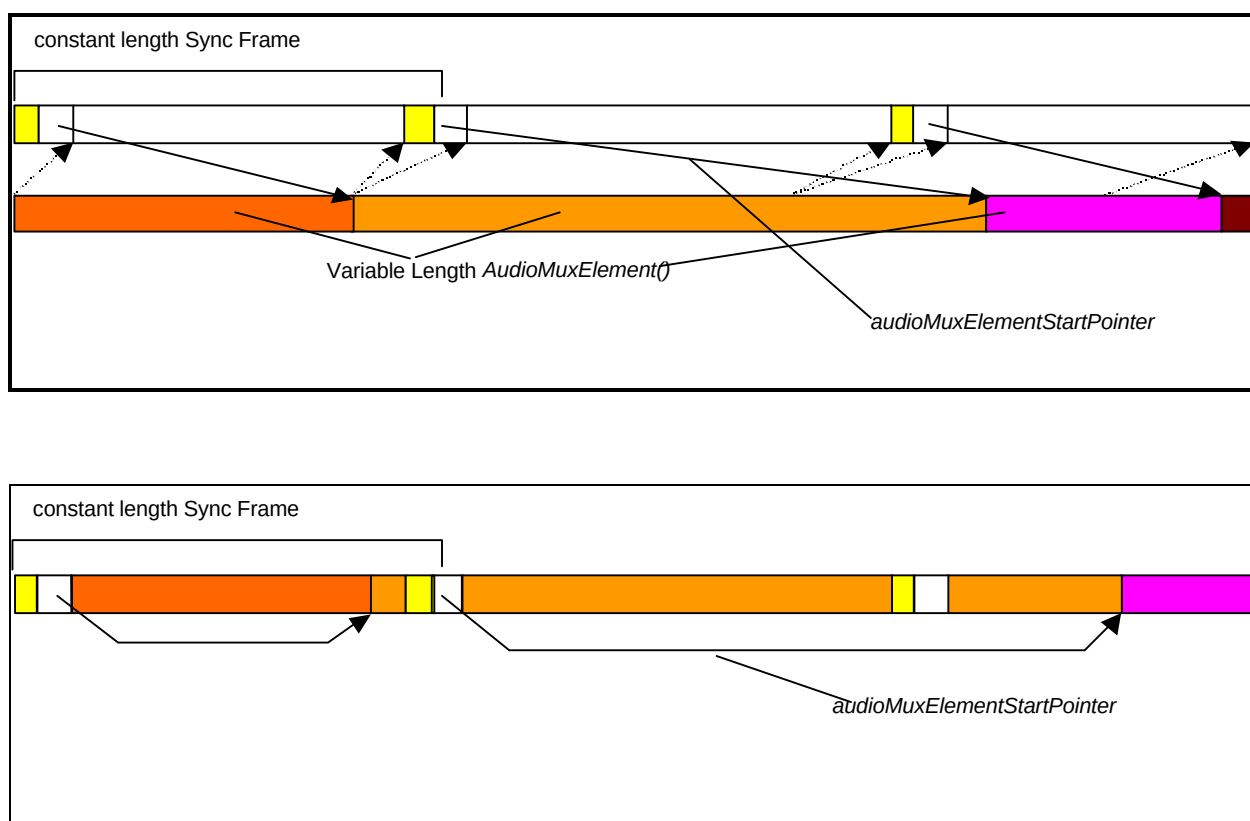


Figure 2: Synchronization in AudioPointerStream()

6.5.2.1 Syntax

Table 13: Syntax of AudioSyncStream()

Syntax	No. of bits	Mnemonic
AudioSyncStream() { while (nextbits() == 0x2B7) { /* syncword */ audioMuxLengthBytesLast ; AudioMuxElement(1); ByteAlign(); } }	11 13	bslbf uimsbf

Table 14: Syntax of EPAudioSyncStream()

Syntax	No. of bits	Mnemonic
EPAudioSyncStream() { while (nextbits() == 0x4de1) { /* syncword */ futureUse ; audioMuxLengthBytes ; frameCounter ; headerParity ; EPMuxElement(1, 1); } }	16 4 13 5 18	bslbf uimsbf uimsbf uimsbf bslbf

Table 15: Syntax of AudioPointerStream()

Syntax	No. of bits	Mnemonic
AudioPointerStream(length) { ByteAlign(); audioMuxElementStartPointer ; AudioMuxElement(1); }	 ceil(lid(length))	 uimsbf

6.5.2.2 Semantics

audioMuxLengthBytesLast	A 13-bit field indicating the byte length of the multiplexed element with byte alignment.
futureUse	A 4-bit field for future use.
audioMuxLengthBytes	A 13-bit field indicating the byte length of the multiplexed element.
frameCounter	A 5-bit field indicating a sequential number which is used to detect lost frames. The number is continuously incremented for each multiplexed element as a modulo counter.

headerParity A 18-bit field which contains a BCH (36,18) code shortened from BCH (63,45) code for the elements **audioMuxLengthBytes** and **frameCounter**. The generator polynomial is $x^{18}+x^{17}+x^{16}+x^{15}+x^9+x^7+x^6+x^3+x^2+x+1$. The value is calculated with this generation polynomial as described in subclause 9.4.3.

audioMuxElementStartPointer A field indicating the start point of the multiplexed element. The number of bits required for this field is calculated as $nbits = \text{ceil}(\log_2(\text{the transmission frame length}))$. The transmission frame length should be provided from the transmission layer.

AudioMuxElement() A multiplexed element as specified in subclause 6.5.3.

EPMuxElement() An error resilient multiplexed element as specified in subclause 6.5.3.

6.5.3 Multiplex Layer

The LATM layer multiplexes several MPEG-4 Audio payloads and AudioSpecificInfo() syntax elements into one multiplexed element. The multiplexed element format is selected between AudioMuxElement() and EPMuxElement() depending on whether error resilience is required in the multiplexed element itself, or not. EPMuxElement() is an error resilient version of AudioMuxElement() and may be used for error prone channels.

The multiplexed elements can be directly conveyed on transmission layers with frame synchronization. In this case, the first bit of the multiplexed element shall be located at the first bit of a transmission payload in the underlying transmission layer. If the transmission payload allows only byte-aligned payload, zero-padding bits for byte alignment shall follow the multiplexed element. The number of the padding bits should be less than 8. These padding bits should be removed when the multiplexed element is de-multiplexed into the MPEG-4 Audio payloads. Then, the MPEG-4 Audio payloads are forwarded to the corresponding MPEG-4 Audio decoder tool.

6.5.3.1 Syntax

Table 16: Syntax of EPMuxElement()

Syntax	No. of bits	Mnemonic
EPMuxElement(epDataPresent , muxConfigPresent)		
{		
if(epDataPresent) {		
epUsePreviousMuxConfig;	1	bslbf
epUsePreviousMuxConfigParity;	2	bslbf
if(!epUsePreviousMuxConfig) {		
epSpecificConfigLength;	10	bslbf
epSpecificConfigLengthParity;	11	bslbf
ErrorProtectionSpecificConfig();		
ErrorProtectionSpecificConfigParity();		
}		
ByteAlign();		
EPAudioMuxElement(muxConfigPresent);		
} else {		
AudioMuxElement(muxConfigPresent);		
ByteAlign();		
}		
}		

Table 17: Syntax of AudioMuxElement()

Syntax	No. of bits	Mnemonic
<pre> AudioMuxElement(muxConfigPresent) { if(muxConfigPresent) { useSameStreamMux; if (!useSameStreamMux) StreamMuxConfig(); } for(i=0; i<numSubFrames; i++) { PayloadLengthInfo(); PayloadMux(); } if(otherDataPresent) { for(i=0; i<otherDataLenBits; i++) { otherDataBit; } } } </pre>	1	bslbf
	1	bslbf

Table 18: Syntax of StreamMuxConfig()

Syntax	No. of bits	Mnemonic
StreamMuxConfig()		
{		
streamCnt = 0;		
allStreamsSameTimeFraming;	1	uimsbf
numSubFrames;	3	uimsbf
numProgram;	4	uimsbf
for (prog = 0; prog < numProgram; prog++) {		
numLayer;	3	uimsbf
for (lay = 0; lay < numLayer; lay++) {		
progSndx[streamCnt]=prog; laySndx[streamCnt]=lay;		
streamID [prog][lay] = streamCnt++;		
if (prog == 0 & lay == 0) {		
AudioSpecificInfo();		
}else {		
useSameConfig;	1	uimsbf
if (!useSameConfig)		
AudioSpecificInfo();		
}		
frameLengthType [streamID[prog][lay]];	3	uimsbf
if (frameLengthType[streamID[prog][lay] == 0) {		
blockDelay [streamID[prog][lay]];	5	uimsbf
fractionalDelayPresent [streamID[prog][lay]];	1	uimsbf
if (fractionalDelayPresent[streamID[prog][lay]]		
fractionalDelay [streamID[prog][lay]];	8	uimsbf
} else if(frameLengthType[streamID[prog][lay]] == 1) {		
frameLength [streamID[prog][lay]];	9	uimsbf
} else if (frameLengthType[streamID[prog][lay]] == 4		
frameLengthType[streamID[prog][lay]] == 5		
frameLengthType[streamID[prog][lay]] == 3) {		
CELPframeLengthTableIndex [streamID[prog][lay]];	6	uimsbf
} else if (frameLengthType[streamID[prog][lay]] == 6		
frameLengthType[streamID[prog][lay]] == 7) {		
HVXCframeLengthTableIndex [streamID[prog][lay]];	1	uimsbf
}		
}		
}		
otherDataPresent;	1	uimsbf
if(otherDataPresent) otherDataLenBits;	8	uimsbf
crcCheckPresent;	1	
if(crcCheckPresent) crcCheckSum;	8	uimsbf
}		

Table 19: Syntax of PayloadLengthInfo()

Syntax	No. of bits	Mnemonic
<pre> PayloadLengthInfo() { if(allStreamsSameTimeFraming) { for (prog = 0; prog < numProgram; prog++) { for (lay = 0; lay < numLayer; lay++) { if(frameLengthType[streamID[prog][lay]] == 0) { do { /* always one complete access unit */ tmp; MuxSlotLengthBytes[streamID[prog][lay]] += tmp; } while(tmp==255); } else { if (frameLengthType[streamID[prog][lay]] == 5 frameLengthType[streamID[prog][lay]] == 7 frameLengthType[streamID[prog][lay]] == 3) { MuxSlotLengthCoded[streamID[prog][lay]]; } } } } } } else { numChunk; for (cnt=0; cnt < numChunk; cnt++) { streamIdx; prog = progCIdx[chunkCnt] = progSIdx[streamIdx]; lay = layCIdx[chunkCnt] = laySIdx [streamIdx]; if(frameLengthType[streamID[prog][lay]] == 0) { do { /* not necessarily a complete access unit */ tmp; MuxSlotLengthBytes[streamID[prog][lay]] += tmp; } while (tmp == 255); AuEndFlag[streamID[prog][lay]]; } else { if (frameLengthType[streamID[prog][lay]] == 5 frameLengthType[streamID[prog][lay]] == 7 frameLengthType[streamID[prog][lay]] == 3) { MuxSlotLengthCoded[streamID[prog][lay]]; } } } } } </pre>	<p>8</p> <p>2</p> <p>4</p> <p>4</p> <p>8</p> <p>1</p> <p>2</p>	<p>uimsbf</p> <p>uimsbf</p> <p>uimsbf</p> <p>uimsbf</p> <p>bslbf</p> <p>uimsbf</p>

Table 20: Syntax of PayloadMux()

Syntax	No. of bits	Mnemonic
<pre> PayloadMux() { if(allStreamsSameTimeFraming) { for (prog = 0; prog < numProgram; prog++) { for (lay = 0; lay < numLayer; lay++) { payload [streamID[prog][lay]]; } } } else { for (chunkCnt=0; chunkCnt < numChunk; chunkCnt++) { prog = progCIdx[chunkCnt]; lay = layCIdx [chunkCnt]; payload [streamID[prog][lay]]; } } } </pre>		

6.5.3.2 Semantics

In order to parse an AudioMuxElement(), a muxConfigPresent flag shall be set at the underlying layer. If muxConfigPresent is set to 1, this indicates multiplexing configuration (StreamMuxConfig()) is multiplexed into AudioMuxElement(), i.e. in-band transmission. If not, StreamMuxConfig() should be conveyed through out-band means, such as session announcement/description/control protocols. For parsing of EPMuxElement(), an epDataPresent flag shall be additionally set at the underlying layer. If epDataPresent() is set to 1, this indicates EPMuxElement() has error resiliency. If not, the format of EPMuxElement() is identical to AudioMuxElement(). The default for both flags is 1.

muxConfigPresent	Description
0	out-band transmission of StreamMuxConfig()
1	in-band transmission of StreamMuxConfig()

epDataPresent	Description
0	EPMuxElement() is identical to AudioMuxElement()
1	EPMuxElement() has error resiliency

epUsePreviousMuxConfig A flag indicating whether the configuration for the MPEG-4 Audio EP tool in the previous frame is applied in the current frame.

epUsePreviousMuxConfig	Description
0	The configuration for the MPEG-4 Audio EP tool is present
1	The configuration for the MPEG-4 Audio EP tool is not present. The previous configuration should be applied

epUsePreviousMuxConfigParity A 2-bits element which contains the parity for **epUsePreviousMuxConfig**. Each bit is a repetition of **epUsePreviousMuxConfig**. Majority decides.

epHeaderLength A 10-bit field to indicate the size of ErrorProtectionSpecificConfig()

epHeaderLengthParity: A 11-bit field for epHeaderLength, calculated as described in subclause 9.4.3 with “1)Basic set of FEC codes”.

Note: This means shortened Golay(23,12) is used

ErrorProtectionSpecificConfig() Configuration information for the EP tool which is applied to AudioMuxElement() as defined in subclause 9.2.1.

ErrorProtectionSpecificConfigParity() The parity bits for **ErrorProtectionSpecificConfig()**, calculated as described in subclause 9.4.3 with “1) Basic Set of FEC codes”.

EPAudioMuxElement() Error resilient multiplexed element that is generated by applying the EP tool to AudioMuxElement() as specified by ErrorProtectionSpecificConfig(). Therefore data elements in AudioMuxElement() are subdivided into different categories depending on their error sensitivity and collected in instances of these categories. Following sensitivity categories are defined:

elements	error sensitivity category
useSameStreamMux + StreamMuxConfig()	0
PayloadLengthInfo()	1
PayloadMux()	2
otherDataBits	3

Note 1: There might be more than one instance of error sensitivity category 1 and 2 depending on the value of the variable **numSubFrames** defined in **StreamMuxConfig()**. Figure 3 shows an example for the order of the instances assuming numSubFrames is two (2).

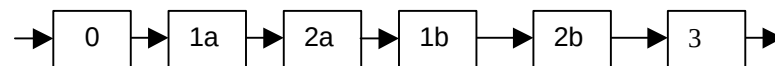


Figure 3: Instance order in EPAudioMuxElement()

Note 2: EPAudioMuxElement() has to be byte aligned, therefore **bit_stuffing** in ErrorProtectionSpecificConfig() should be always on.

useSameStreamMux A flag indicating whether the multiplexing configuration in the previous frame is applied in the current frame.

useSameStreamMux	Description
0	The multiplexing configuration is present.
1	The multiplexing configuration is not present. The previous configuration should be applied.

otherDataBit A 1-bit field indicating the other data information.

allStreamsSameTimeFraming A flag indicating whether all payloads, which are multiplexed in PayloadMux(), share a common time base.

numSubFrames A field indicating how many PayloadMux() frames are multiplexed. If more than one PayloadMux() frame are multiplexed, all PayloadMux() share a common StreamMuxConfig(). The minimum value is 1.

numProgram A field indicating how many programs are multiplexed. The minimum value is 1.

numLayer A field indicating how many scalable layers are multiplexed. The minimum value is 1.

useSameConfig

A flag indicating whether AudioSpecificInfo() for the payload in the previous layer or program is applied for the payload in the current layer or program.

useSameConfig	Description
0	AudioSpecificInfo() is present.
1	AudioSpecificInfo() is not present. AudioSpecificInfo() in the previous layer or program should be applied.

frameLengthType

A field indicating the frame length type of the payload. For CELP and HVXC objects, the frame length (bits/frame) is stored in tables and only the indexes to point out the frame length of the current payload is transmitted instead of sending the frame length value directly.

frameLengthType	Description
0	Payload with variable frame length. The payload length in bytes is directly specified with 8-bit codes in PayloadLengthInfo().
1	Payload with fixed frame length. The payload length in bits is specified with frameLength in StreamMuxConfig().
2	Reserved
3	Payload for a CELP object with one of 2 kinds of frame length. The payload length is specified by two table-indexes, namely CELPframeLengthTableIndex and MuxSlotLengthCoded.
4	Payload for a CELP or ER_CELP object with fixed frame length. CELPframeLengthTableIndex specifies the payload length.
5	Payload for an ER_CELP object with one of 4 kinds of frame length. The payload length is specified by two table-indexes, namely CELPframeLengthTableIndex and MuxSlotLengthCoded.
6	Payload for a HVXC or ER_HVXC object with fixed frame length. HVXCframeLengthTableIndex specifies the payload length.
7	Payload for an HVXC or ER_HVXC object with one of 4 kinds of frame length. The payload length is specified by two table-indexes, namely HVXCframeLengthTableIndex and MuxSlotLengthCoded.

blockDelay

A field indicating the time base of the payload with frameLengthType of 0. The time base is specified by a two-layer scheme. blockDelay provides a coarse time base in multiples of $tb1 = \text{frame_length_samples} / \text{sampling_rate}$. If enhanced time resolution is desired, fractionalDelay may specify an additional fractional value.

fractionalDelayPresent

A flag indicating the presence of fractionalDelay in the current payload.

fractionalDelayPresent	Description
0	fractionalDelay is not present.
1	fractionalDelay is present.

fractionalDelay

A field indicating the enhanced time resolution for the time base of the payload with frameLengthType of 0.

frameLength

A field indicating the frame length of the payload with frameLengthType of 1. The payload length in bits is specified as $8 * (\text{frameLength} + 20)$.

CELPframeLengthTableIndex A field indicating one of two indexes for pointing out the frame length for a CELP or ER_CELP object. (Table 22 and Table 23)

HVXCframeLengthTableIndex A field indicating one of two indexes for pointing out the frame length for a HVXC or ER_HVXC object. (Table 21)

otherDataPresent A flag indicating the presence of the other data than audio payloads.

otherDataPresent	Description
0	The other data than audio payload otherData is not multiplexed.
1	The other data than audio payload otherData is multiplexed.

otherDataLenBits A field indicating the length of the other data.

crcCheckPresent A flag indicating the presence of CRC check bits for StreamMuxConfig() elements.

crcCheckPresent	Description
0	CRC check bits are not present.
1	CRC check bits are present.

crcChecksum A field indicating the CRC check bits.

tmp A field indicating the payload length of the payload with frameLengthType of 0. The value 255 is used as an escape value and indicates that at least one more **tmp** value is following. The overall length of the transmitted payload is calculated by summing up the partial values.

MuxSlotLengthCoded A field indicating one of two indexes for pointing out the payload length for CELP, HVXC, ER_CELP, and ER_HVXC objects.

numChunk A field indicating the number of payload chunks. Each chunk may belong to an access unit with a different time base; only used if allStreamsSameTimeFraming is set to zero. The minimum value is 1.

streamIndx A field indicating the stream. Used if payloads are splitted into chunks.

chunkCnt Helper variable to count number of chunks.

progSindx,laySindx Helper variables to identify program and layer number from **streamIndx**.

progCindx,layCindx Helper variables to identify program and layer number from **chunkCnt**.

AuEndFlag A flag indicating whether the payload is the last fragment, in the case that an access unit is transmitted in pieces.

AuEndFlag	Description
0	The fragmented piece is not the last one.
1	The fragmented piece is the last one.

Table 21: Frame length of HVXC [bits]

frameLengthType[]	HVXCframeLengthTableIndex[]	MuxSlotLengthCoded			
		00	01	10	11
6	0	40			
6	1	80			
7	0	40	28	2	0
7	1	80	40	25	3

Table 22: Frame Length of CELP Layer 0 [bits]

CELPframeLengthTable Index	Fixed-Rate frameLengthType[] =4	1-of-4 Rates (Silence Compression) frameLengthType[]=5				1-of-2 Rates (FRC) frameLengthType[]=3	
		MuxSlotLengthCoded				MuxSlotLengthCoded	
		00	01	10	11	00	01
0	154	156	23	8	2	156	134
1	170	172	23	8	2	172	150
2	186	188	23	8	2	188	166
3	147	149	23	8	2	149	127
4	156	158	23	8	2	158	136
5	165	167	23	8	2	167	145
6	114	116	23	8	2	116	94
7	120	122	23	8	2	122	100
8	126	128	23	8	2	128	106
9	132	134	23	8	2	134	112
10	138	140	23	8	2	140	118
11	142	144	23	8	2	144	122
12	146	148	23	8	2	148	126
13	154	156	23	8	2	156	134
14	166	168	23	8	2	168	146
15	174	176	23	8	2	176	154
16	182	184	23	8	2	184	162
17	190	192	23	8	2	192	170
18	198	200	23	8	2	200	178
19	206	208	23	8	2	208	186
20	210	212	23	8	2	212	190
21	214	216	23	8	2	216	194
22	110	112	23	8	2	112	90
23	114	116	23	8	2	116	94
24	118	120	23	8	2	120	98
25	120	122	23	8	2	122	100
26	122	124	23	8	2	124	102
27	186	188	23	8	2	188	166
28	218	220	40	8	2	220	174
29	230	232	40	8	2	232	186
30	242	244	40	8	2	244	198
31	254	256	40	8	2	256	210
32	266	268	40	8	2	268	222
33	278	280	40	8	2	280	234
34	286	288	40	8	2	288	242
35	294	296	40	8	2	296	250
36	318	320	40	8	2	320	276
37	342	344	40	8	2	344	298
38	358	360	40	8	2	360	314
39	374	376	40	8	2	376	330
40	390	392	40	8	2	392	346
41	406	408	40	8	2	408	362

42	422	424	40	8	2	424	378
43	136	138	40	8	2	138	92
44	142	144	40	8	2	144	98
45	148	150	40	8	2	150	104
46	154	156	40	8	2	156	110
47	160	162	40	8	2	162	116
48	166	168	40	8	2	168	122
49	170	172	40	8	2	172	126
50	174	176	40	8	2	176	130
51	186	188	40	8	2	188	142
52	198	200	40	8	2	200	154
53	206	208	40	8	2	208	162
54	214	216	40	8	2	216	170
55	222	224	40	8	2	224	178
56	230	232	40	8	2	232	186
57	238	240	40	8	2	240	194
58	216	218	40	8	2	218	172
59	160	162	40	8	2	162	116
60	280	282	40	8	2	282	238
61	338	340	40	8	2	340	296
62-63	reserved						

Table 23: Frame Length of CELP Layer 1-5 [bits]

CELPframeLengthTableIndex	Fixed-Rate frameLengthType []=4	1-of-4 Rates (Silence Compression) frameLengthType[]=5			
		MuxSlotLengthCoded			
		00	01	10	11
0	80	80	0	0	0
1	60	60	0	0	0
2	40	40	0	0	0
3	20	20	0	0	0
4	368	368	21	0	0
5	416	416	21	0	0
6	464	464	21	0	0
7	496	496	21	0	0
8	284	284	21	0	0
9	320	320	21	0	0
10	356	356	21	0	0
11	380	380	21	0	0
12	200	200	21	0	0
13	224	224	21	0	0
14	248	248	21	0	0
15	264	264	21	0	0
16	116	116	21	0	0
17	128	128	21	0	0
18	140	140	21	0	0
19	148	148	21	0	0
20-63	reserved				

7 Parametric audio coding (HILN)

7.1 Overview of the tools

MPEG-4 parametric audio coding uses the HILN technique (Harmonic and Individual Lines plus Noise) to code audio signals like music at bitrates of 4 kbit/s and higher using a scalable parametric representation of the audio signal. HILN allows independent change of speed and pitch during decoding. Furthermore HILN can be combined with MPEG-4 parametric speech coding (HVXC) to form an integrated parametric coder covering a wider range of signals and bitrates.

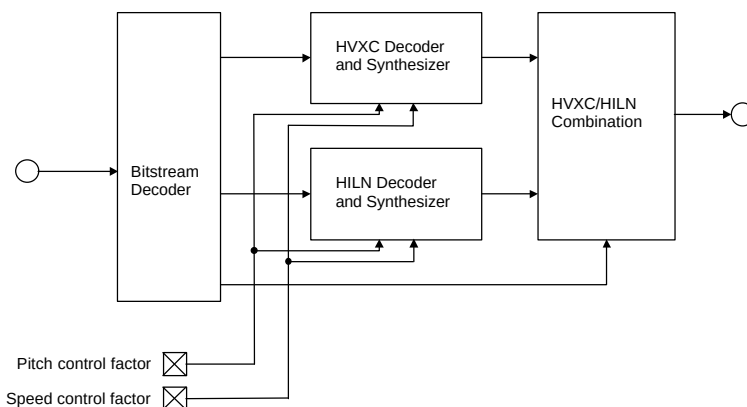


Figure 4: Block diagram of the integrated parametric decoder

The integrated parametric coder can operate in the following modes:

Table 24: Parametric coder operation modes

PARAMode	Description
0	HVXC only
1	HILN only
2	switched HVXC / HILN
3	mixed HVXC / HILN

PARAModes 0 and 1 represent the fixed HVXC and HILN modes. PARAMode 2 permits automatic switching between HVXC and HILN depending on the current input signal type. In PARAMode 3 the HVXC and HILN decoders can be used simultaneously and their output signals are added (mixed) in the parametric decoder.

In “switched HVXC / HILN” and “mixed HVXC / HILN” modes both HVXC and HILN decoder tools are operated alternatively or simultaneously according to the PARASwitchMode or PARAMixMode of the current frame. To obtain proper time alignment of both HVXC and HILN decoder output signals before they are added, a FIFO buffer compensates for the time difference between HVXC and HILN decoder delay.

To avoid hard transitions at frame boundaries when the HVXC or HILN decoders are switched on or off, the respective decoder output signals fade in and out smoothly. For the HVXC decoder a 20 ms linear fade is applied when it is switched on or off. The HILN decoder requires no additional fading because of the smooth synthesis windows utilized in the HILN synthesizer. It is only necessary to reset the HILN decoder (numLine = 0) if the current bitstream frame contains no HILNframe().

7.2 Terms and definitions

For the purposes of Subclause 7 the following definitions apply:

HVXC:	Harmonic Vector Excitation Coding (parametric speech coding).
HILN:	Harmonic and Individual Lines plus Noise (parametric audio coding).
individual line:	A spectral component described by frequency, amplitude and phase.
harmonic lines:	A set of spectral components having a common fundamental frequency.
noise component:	A signal component modeled as noise.
pi:	The constant $\pi = 3.14159...$

A general glossary and list of symbols and abbreviations is located in clause 0.

7.3 Bitstream syntax

7.3.1 Decoder configuration (ParametricSpecificConfig)

The decoder configuration information for parametric coding is transmitted in the DecoderConfigDescriptor() of the base layer and the optional enhancement layer Elementary Stream (see Subclause 6.2.1).

Parametric Base Layer -- Configuration

For the parametric coder in unscalable mode or for the base layer in HILN scalable mode the following ParametricSpecificConfig() is required:

```
ParametricSpecificConfig() {
    PARAconfig();
}
```

Parametric HILN Enhancement / Extension Layer -- Configuration

To use HILN as core in an "T/F scalable with core" mode, in addition to the HILN base layer an HILN enhancement layer is required. In HILN bitrate scalable operation, in addition to the HILN base layer one or more HILN extension layers are permitted. Both the enhancement layer and the extension layer have the following ParametricSpecificConfig():

```
ParametricSpecificConfig() {
    HILNenexConfig();
}
```

An MPEG-4 Natural Audio Object using Parametric Coding is transmitted in one or more Elementary Streams: The base layer stream, an optional enhancement layer stream, and one or more optional extension layer streams.

The bitstream syntax is described in pseudo-C code.

The mnemonics LARH1, LARH2, LARH3, LARN1, LARN2, DIA, DIF, DHF, DFS indicate that a "vclbf" codeword is used. The corresponding codebooks are given in Subclause 7.3.2.3.

The mnemonic SDC indicates that a "vclbf" codeword is used which is decoded by the HILN SubDivisionCode described in Subclause 7.3.2.4 using the parameters for SDCdecode() as given in the bitstream syntax description.

7.3.1.1 Parametric Audio decoder configuration

Table 25: Syntax of PARAconfig()

Syntax	No. of bits	Mnemonic
<pre> PARAconfig() { PARAMode; if (PARAMode != 1) { ErHVXCconfig(); } if (PARAMode != 0) { HILNconfig(); } extensionFlag; if (extensionFlag) { /* to be defined in MPEG-4 Phase 3 */ } } </pre>	<p>2</p> <p>1</p>	<p>uimbsbf</p> <p>uimbsbf</p>

Table 26: PARAMode

PARAMode	frameLength	Description
0	20 ms (N = 160 samples)	HVXC only
1	see Subclause 7.3.1.2 and Subclause 7.5.1.4.3.3	HILN only
2	40 ms (N = 320 samples)	HVXC/HILN switching
3	40 ms (N = 320 samples)	HVXC/HILN mixing

7.3.1.2 HILN decoder configuration

Table 27: Syntax of HILNconfig()

Syntax	No. of bits	Mnemonic
HILNconfig() { HILNquantMode; HILNmaxNumLine; HILNsampleRateCode; HILNframeLength; HILNcontMode; }	 1 8 4 12 2	 uimsbf uimsbf uimsbf uimsbf uimsbf

Table 28: Syntax of HILNenexConfig()

Syntax	No. of bits	Mnemonic
HILNenexConfig() {		
HILNenhaLayer;	1	uimsbf
if (HILNenhaLayer) {		
HILNenhaQuantMode;	2	uimsbf
}		
}		

Table 29: HILNsampleRateCode

HILNsampleRateCode	sampleRate	maxFIndex
0	96000	890
1	88200	876
2	64000	825
3	48000	779
4	44100	765
5	32000	714
6	24000	668
7	22050	654
8	16000	603
9	12000	557
10	11025	544
11	8000	492
12	7350	479
13	reserved	reserved
14	reserved	reserved
15	reserved	reserved

Table 30: linebits

HILNmaxNumLine	0	1	2..3	4..7	8..15	16..31	32..63	64..127	128..255
linebits	0	1	2	3	4	5	6	7	8

Table 31: HILNcontMode

HILNcontMode	additional decoder line continuation (see subclause 7.5.1.4.3.1)
0	harmonic lines <-> individual lines and harmonic lines <-> harmonic lines
1	mode 0 plus individual lines <-> individual lines
2	no additional decoder line continuation
3	(reserved)

The number of frequency enhancement bits (fEnhbits[i]) in HILNenhaFrame() (see Subclause 7.3.2.2) is calculated as follows:

- individual line, see INDlenhaPara():

$$\text{fEnhbits}[i] = \max(0, \text{fEnhbitsBase}[\text{ILFreqIndex}[i]] + \text{fEnhbitsMode}[\text{HILNenhaQuantMode}])$$

- harmonic line, see HARMenhaPara():

$$\text{fEnhbits}[i] = \max(0, \text{fEnhbitsBase}[\text{harmFreqIndex}] + \text{fEnhbitsMode}[\text{HILNenhaQuantMode}] + \text{fEnhbitsHarm}[i])$$

Table 32: fEnhbitsBase

ILFreqIndex	harmFreqIndex	fEnhbitsBase
0..159	0..1243	0
160..269	1244..1511	1
270..380	1512..1779	2
381..491	1780..2047	3
492..602		4
603..713		5
714..890		6

Table 33: fEnhbitsMode

HILNenhQuantMode	0	1	2	3
fEnhbitsMode	-3	-2	-1	0

Table 34: fEnhbitsHarm

i	0	1	2..3	4..7	8..9
fEnhbitsHarm[i]	0	1	2	3	4

Table 35: HILN constants

tmBits	4
atkBits	4
decBits	4
tmEnhbits	3
atkEnhbits	2
decEnhbits	2
phasebits	5

7.3.2 Bitstream frame (alPduPayload)

The dynamic data for parametric coding is transmitted as AL-PDU payload in the base layer and the optional enhancement or extension layer Elementary Stream.

Parametric Base Layer -- Access Unit payload

For the parametric coder in unscalable mode or for the base layer in HILN scalable mode the following bitstream frame payload is defined:

```
alPduPayload {
    PARAFrame();
}
```

Parametric HILN Enhancement / Extension Layer -- Access Unit payload

To parse and decode the HILN enhancement layer, information decoded from the HILN base layer is required.

To parse and decode an HILN extension layer, information decoded from the HILN base layer and possible lower HILN extension layers is required. The bitstream syntax of the HILN extension layers is described in a way which requires the HILN base and extension bitstream frames being parsed in proper order:

1. HILNbasicFrame() base bitstream frame
2. HILNNextFrame(1) 1st extension bitstream frame (if base bitstream frame available)
3. HILNNextFrame(2) 2nd extension bitstream frame (if base and 1st extension bitstream frame available)

4. etc.

For the enhancement layer and the extension layer in HILN scalable mode the following bitstream frame payload is defined:

```
alPduPayload {
    HILNenexFrame();
}
```

7.3.2.1 Parametric Audio bitstream frame

Table 36: Syntax of PARAframe()

Syntax	No. of bits	Mnemonic
<pre>PARAframe() { if (PARAMode == 0) { ErHVXCframe(HVXCrate); } else if (PARAMode == 1) { HILNframe(); } else if (PARAMode == 2) { switchFrame(); } else if (PARAMode == 3) { mixFrame(); } }</pre>		

Table 37: Syntax of switchFrame()

Syntax	No. of bits	Mnemonic
<pre>switchFrame() { PARAswitchMode; if (PARAswitchMode == 0) { ErHVXCdoubleframe(HVXCrate); } else { HILNframe(); } }</pre>	1	uimsbf

One of the following PARAswitchModes is selected in each frame:

Table 38: PARAswitchMode

PARAswitchMode	Description
0	HVXC only
1	HILN only

Table 39: Syntax of mixFrame()

Syntax	No. of bits	Mnemonic
<pre> mixFrame() { PARAMixMode; if (PARAMixMode == 0) { ErHVXCdoubleframe(HVXCrate); } else if (PARAMixMode == 1) { HILNframe(); ErHVXCdoubleframe(2000); } else if (PARAMixMode == 2) { HILNframe(); ErHVXCdoubleframe(4000); } else if (PARAMixMode == 3) { HILNframe(); } } </pre>	2	uimsbf

One of the following PARAMixModes is selected in each frame:

Table 40: PARAMixMode

PARAMixMode	Description
0	HVXC only
1	HVXC 2 kbit/s & HILN
2	HVXC 4 kbit/s & HILN
3	HILN only

Table 41: Syntax of HVXCdoubleframe()

Syntax	No. of bits	Mnemonic
<pre> ErHVXCdoubleframe(rate) { if (rate >= 3000) { ErHVXCfixframe(4000); ErHVXCfixframe(rate); } else { ErHVXCfixframe(2000); ErHVXCfixframe(rate); } } </pre>		

7.3.2.2 HILN bitstream frame

Table 42: Syntax of HILNframe()

Syntax	No. of bits	Mnemonic
<pre> HILNframe() { numLayer = 0; HILNbasicFrameESC0(); HILNbasicFrameESC1(); HILNbasicFrameESC2(); HILNbasicFrameESC3(); HILNbasicFrameESC4(); layNumLine[0] = numLine; layPrevNumLine[0] = prevNumLine; for (k=0; k<prevNumLine; k++) { layPrevLineContFlag[0][k] = prevLineContFlag[k]; } } </pre>		

Table 43: Syntax of HILNbasicFrameESC0()

Syntax	No. of bits	Mnemonic
HILNbasicFrameESC0()		
{		
prevNumLine = numLine;		
/* prevNumLine is set to the number of lines */		
/* in the previous frame */		
/* prevNumLine = 0 for the first bitstream frame */		
numLine;	linebits	uimsbf
harmFlag;	1	uimsbf
noiseFlag;	1	uimsbf
envFlag;	1	uimsbf
phaseFlag;	1	uimsbf
maxAmplIndexCoded;	4	uimsbf
maxAmplIndex = 4*maxAmplIndexCoded;		
if (harmFlag) {		
HARMbasicParaESC0();		
}		
if (noiseFlag) {		
NOISEbasicParaESC0();		
}		
}		

Table 44: Syntax of HILNbasicFrameESC1()

Syntax	No. of bits	Mnemonic
<pre> HILNbasicFrameESC1() { if (harmFlag) { HARMbasicParaESC1(); } if (noiseFlag) { NOISEbasicParaESC1(); } INDbasicParaESC1(); } </pre>		

Table 45: Syntax of HILNbasicFrameESC2()

Syntax	No. of bits	Mnemonic
<pre> HILNbasicFrameESC2() { INDbasicParaESC2 (); } </pre>		

Table 46: Syntax of HILNbasicFrameESC3()

Syntax	No. of bits	Mnemonic
<pre> HILNbasicFrameESC3() { if (envFlag) { envTmax; envRatk; envRdec; } if (harmFlag) { HARMbasicParaESC3(); } if (noiseFlag) { NOISEbasicParaESC3(); } INDbasicParaESC3(); if (harmFlag) { harmFreqStretch; } } </pre>	tmbits atkbits decbits	uimbsf uimbsf uimbsf
	1..7	HFS