INTERNATIONAL STANDARD

ISO/IEC 14496-3

Third edition
2005-12-01

Information technology — Coding of audio-visual objects —

Part 3: **Audio**

Technologies de l'information — Codage des objets audiovisuels —

Partie 3: Codage audio

Reference number
ISO/IEC 14496-3:2005(E)

Foreword

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

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

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

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

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

This third edition cancels and replaces the second edition (ISO/IEC 14496-3:2001), which has been technically revised. It also incorporates the Amendments ISO/IEC 14496-3:2001/Amd.1:2003, ISO/IEC 14496-3:2001/ Amd.2:2004, ISO/IEC 14496-3:2001/Amd.3:2005 and ISO/IEC 14496-3:2001/Amd.6:2005, and the Technical Corrigenda ISO/IEC 14496-3:2001/Cor.1:2002 and ISO/IEC 14496-3:2001/Cor.2:2004.

ISO/IEC 14496 consists of the following parts, under the general title *Information technology — Coding of audio-visual objects*:

— *Part 1: Systems*

— *Part 2: Visual*

— *Part 3: Audio*

— *Part 4: Conformance testing*

— *Part 5: Reference software*

— *Part 6: Delivery Multimedia Integration Framework (DMIF)*

— *Part 7: Optimized reference software for coding of audio-visual objects* [Technical Report]

— *Part 8: Carriage of ISO/IEC 14496 contents over IP networks*

— *Part 9: Reference hardware description* [Technical Report]

— *Part 10: Advanced Video Coding*

— *Part 11: Scene description and application engine*

— *Part 12: ISO base media file format*

— *Part 13: Intellectual Property Management and Protection (IPMP) extensions*

— *Part 14: MP4 file format*

— *Part 15: Advanced Video Coding (AVC) file format*

— *Part 16: Animation Framework eXtension (AFX)*

— *Part 17: Streaming text format*

— *Part 18: Font compression and streaming*

— *Part 19: Synthesized texture stream*

— *Part 20: Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF)*

0 Introduction

0.1 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 remainder of this Introduction gives a more detailed overview of the capabilities and functioning of MPEG-4 Audio. First a discussion of concepts, that have changed since the MPEG-2 audio standards, is presented. Then the MPEG-4 Audio toolset is outlined.

0.2 Concepts of MPEG-4 Audio

As with previous MPEG standards, MPEG-4 does not standardize methods for encoding sound. Thus, content authors are left to their own decisions as to the best method of creating bitstream payloads. At the present time, methods to automatically convert natural sound into synthetic or multi-object descriptions are not mature; therefore, most immediate solutions will involve interactively-authoring the content stream in some way. This process is similar to current schemes for MIDI-based and multi-channel mixdown authoring of soundtracks.

Many concepts in MPEG-4 Audio are different from those in previous MPEG Audio standards. For the benefit of readers who are familiar with MPEG-1 and MPEG-2, we provide a brief overview here.

0.2.1 Audio storage and transport facilities

In all of the MPEG-4 tools for audio coding, the coding standard ends at the point of constructing access units that contain the compressed data. The MPEG-4 Systems (ISO/IEC 14496-1) specification describes how to convert these individually coded access units into elementary streams.

There is no standard transport mechanism of these elementary streams over a channel. This is because the broad range of applications that can make use of MPEG-4 technology have delivery requirements that are too wide to easily characterize with a single solution. Rather, what is standardized is an interface (the Delivery Multimedia Interface Format, or DMIF, specified in ISO/IEC 14496-6) that describes the capabilities of a transport layer and the communication between transport, multiplex, and demultiplex functions in encoders and decoders. The use of DMIF and the MPEG-4 Systems specification allows transmission functions that are much more sophisticated than are possible with previous MPEG standards.

However, LATM and LOAS were defined to provide a low overhead audio multiplex and transport mechanism for natural audio applications, which do not require sophisticated object-based coding or other functions provided by MPEG-4 Systems.

The following table gives an overview about the multiplex, storage and transmission formats currently available for MPEG-4 Audio within the MPEG-4 framework:

	Format	Functionality defined in MPEG-4:	Functionality originally defined in:	Description
Multiplex	M4Mux	ISO/IEC 14496-1 (normative)	-	MPEG-4 Multiplex scheme
	LATM	ISO/IEC 14496-3 (normative)	-	Low Overhead Audio Transport Multiplex
Storage	ADIF	ISO/IEC 14496-3 (informative)	ISO/IEC 13818-7 (normative)	Audio Data Interchange Format, (AAC only)
	MP4FF	ISO/IEC 14496-12 (normative)	-	MPEG-4 File Format
Transmission	ADTS	ISO/IEC 14496-3 (informative)	ISO/IEC 13818-7 (normative, exemplarily)	Audio Data Transport Stream, (AAC only)
	LOAS	ISO/IEC 14496-3 (normative, exemplarily)	-	Low Overhead Audio Stream, based on LATM, three versions are available: AudioSyncStream() EPAudioSyncStream() AudioPointerStream()

To allow for a user on the remote side of a channel to dynamically control a server streaming MPEG-4 content, MPEG-4 defines backchannel streams that can carry user interaction information.

0.2.2 MPEG-4 Audio supports low-bitrate coding

Previous MPEG Audio standards have focused primarily on transparent (undetectable) or nearly transparent coding of high-quality audio at whatever bitrate was required to provide it. MPEG-4 provides new and improved tools for this purpose, but also standardizes (and has tested) tools that can be used for transmitting audio at the low bitrates suitable for Internet, digital radio, or other bandwidth-limited delivery. The new tools specified in MPEG-4 are the state-of-the-art tools that support low-bitrate coding of speech and other audio.

0.2.3 MPEG-4 Audio is an object-based coding standard with multiple tools

Previous MPEG Audio standards provided a single toolset, with different configurations of that toolset specified for use in various applications. MPEG-4 provides several toolsets that have no particular relationship to each other, each with a different target function. The profiles of MPEG-4 Audio specify which of these tools are used together for various applications.

Further, in previous MPEG standards, a single (perhaps multi-channel or multi-language) piece of content was transmitted. In contrast, MPEG-4 supports a much more flexible concept of a *soundtrack*. Multiple tools may be used to transmit several *audio objects*, and when using multiple tools together an *audio composition* system is provided to create a single soundtrack from the several audio substreams. User interaction, terminal capability, and speaker configuration may be used when determining how to produce a single soundtrack from the component objects. This capability gives MPEG-4 significant advantages in quality and flexibility when compared to previous audio standards.

0.2.4 MPEG-4 Audio provides capabilities for synthetic sound

In natural sound coding, an existing sound is compressed by a server, transmitted and decompressed at the receiver. This type of coding is the subject of many existing standards for sound compression. In contrast, MPEG-4 standardizes a novel paradigm in which synthetic sound descriptions, including synthetic speech and synthetic music, are transmitted and then *synthesized* into sound at the receiver. Such capabilities open up new areas of very-low-bitrate but still very-high-quality coding.

0.2.5 MPEG-4 Audio provides capabilities for error robustness

Improved error robustness capabilities for all coding tools are provided through the error-resilient bitstream payload syntax. This tool supports advanced channel coding techniques, which can be adapted to the special needs of given coding tools and a given communications channel. This error-resilient bitstream payload syntax is mandatory for all error resillient 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.

Improved error robustness for AAC is provided by a set of error resilience tools. These tools reduce the perceived degradation of the decoded audio signal that is caused by corrupted bits in the bitstream payload.

0.2.6 MPEG-4 Audio provides capabilities for scalability

Previous MPEG Audio standards provided a single bitrate, single bandwidth toolset, with different configurations of that toolset specified for use in various applications. MPEG-4 provides several bitrate and bandwidth options within a single stream, providing a scalability functionality that permits a given stream to scale to the requirement of different channels and applications or to be responsive to a given channel that has dynamic throughput characteristics. The tools specified in MPEG-4 are the state-of-the-art tools providing scalable compression of speech and audio signals.

0.3 The MPEG-4 Audio tool set

0.3.1 Speech coding tools

0.3.1.1 Overview

Speech coding tools are designed for the transmission and decoding of synthetic and natural speech.

Two types of speech coding tools are provided in MPEG-4. The *natural* speech tools allow the compression, transmission, and decoding of human speech, for use in telephony, personal communication, and surveillance applications. The *synthetic* speech tool provides an interface to text-to-speech synthesis systems; using synthetic speech provides very-low-bitrate operation and built-in connection with facial animation for use in low-bitrate video teleconferencing applications.

0.3.1.2 Natural speech coding

The MPEG-4 speech coding toolset covers the compression and decoding of natural speech sound at bitrates ranging between 2 and 24 kbit/s. When variable bitrate coding is allowed, coding at even less than 2 kbit/s, for example an average bitrate of 1.2 kbit/s, is also supported. Two basic speech coding techniques are used: One is a parametric speech coding algorithm, HVXC (Harmonic Vector eXcitation Coding), for very low bit rates; and the other is a CELP (Code Excited Linear Prediction) coding technique. The MPEG-4 speech coders target applications range from mobile and satellite communications, to Internet telephony, to packaged media and speech databases. It meets a wide range of requirements encompassing bitrate, functionality and sound quality.

MPEG-4 HVXC operates at fixed bitrates between 2.0 kbit/s and 4.0 kbit/s using a bitrate scalability technique. It also operates at lower bitrates, typically 1.2 - 1.7 kbit/s, using a variable bitrate technique. HVXC provides communications-quality to near-toll-quality speech in the 100 Hz – 3800 Hz band at 8 kHz sampling rate. HVXC also allows independent change of speed and pitch during decoding, which is a powerful functionality for fast access to speech databases. HVXC functionalities including 2.0 - 4.0 kbit/s fixed bitrate modes and a 2.0 kbit/s maximum variable bitrate mode.

Error Resilient (ER) HVXC extends operation of the variable bitrate mode to 4.0 kbit/s to allow higher quality variable rate coding. The ER HVXC therefore provides fixed bitrate modes of 2.0 - 4.0 kbit/s and a variable bitrate of either less than 2.0 kbit/s or less than 4.0 kbit/s, both in scalable and non-scalable modes. In the variable bitrate modes, non-speech parts are detected in unvoiced signals, and a smaller number of bits are used for these non-speech parts to reduce the average bitrate. ER HVXC provides communications-quality to near-toll-quality speech in the 100 Hz - 3800 Hz band at 8 kHz sampling rate. When the variable bitrate mode is allowed, operation at lower average bitrate is possible. Coded speech using variable bitrate mode at typical bitrates of 1.5 kbit/s average, and at typical bitrate of 3.0 kbit/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 a bitstream payload syntax with the error sensitivity classes to be used with the EP-Tool, and some error concealment functionality is supported for use in error-prone channels such as mobile communication channels. The ER HVXC speech coder target applications range from mobile and satellite communications, to Internet telephony, to packaged media and speech databases.

MPEG-4 CELP is a well-known coding algorithm with new functionality. Conventional CELP coders offer compression at a single bit rate and are optimized for specific applications. Compression is one of the functionalities provided by MPEG-4 CELP, but MPEG-4 also enables the use of one basic coder in multiple applications. It provides scalability in bitrate and bandwidth, as well as the ability to generate bitstream payloads at arbitrary bitrates. The MPEG-4 CELP coder supports two sampling rates, namely, 8 kHz and 16 kHz. The associated bandwidths are 100 Hz – 3800 Hz for 8 kHz sampling and 50 Hz – 7000 Hz for 16 kHz sampling. 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 bitrate during the non-active-voice (silent) frames. During the active-voice (speech) frames, MPEG-4 CELP encoding and decoding are used.

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 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 methods similar to those used in a normal CELP frame. These parameters are optionally re-transmitted in the SID and thus can be updated as required.

MPEG has conducted extensive verification testing in realistic listening conditions in order to prove the efficacy of the speech coding toolset.

0.3.1.3 Text-to-speech interface

Text-to-speech (TTS) capability is becoming a rather common media type and plays an important role in various multi-media application areas. For instance, by using TTS functionality, multimedia content with narration can be easily created without recording natural speech. Before MPEG-4, however, there was no way for a multimedia content provider to easily give instructions to an unknown TTS system. With **MPEG-4 TTS Interface**, a single common interface for TTS systems is standardized. This interface allows speech information to be transmitted in the international phonetic alphabet (IPA), or in a textual (written) form of any language.

The **MPEG-4 Hybrid/Multi-Level Scalable TTS Interface** is a superset of the conventional TTS framework. This extended TTS Interface can utilize prosodic information taken from natural speech in addition to input text and can thus generate much higher-quality synthetic speech. The interface and its bitstream payload format is scalable in terms of this added information; for example, if some parameters of prosodic information are not available, a decoder can generate the missing parameters by rule. Normative algorithms for speech synthesis and text-to-phoneme translation are not specified in MPEG-4, but to meet the goal that underlies the MPEG-4 TTS Interface, a decoder should fully utilize all the provided information according to the user's requirements level.

As well as an interface to text-to-speech synthesis systems, MPEG-4 specifies a joint coding method for phonemic information and facial animation (FA) parameters and other animation parameters (AP). Using this technique, a single bitstream payload may be used to control both the text-to-speech interface and the facial animation visual object decoder (see ISO/IEC 14496-2, Annex C). The functionality of this extended TTS thus ranges from conventional TTS to natural speech coding and its application areas, from simple TTS to audio presentation with TTS and motion picture dubbing with TTS.

0.3.2 Audio coding tools

0.3.2.1 Overview

Audio coding tools are designed for the transmission and decoding of recorded music and other audio soundtracks.

0.3.2.2 General audio coding tools

MPEG-4 standardizes the coding of natural audio at bitrates ranging from 6 kbit/s up to several hundred kbit/s per audio channel for mono, two-channel-, and multi-channel-stereo signals. General high-quality compression is provided by incorporating the MPEG-2 AAC standard (ISO/IEC 13818-7), with certain improvements, as MPEG-4 AAC. At 64 kbit/s/channel and higher ranges, this coder has been found in verification testing under rigorous conditions to meet the criterion of “indistinguishable quality” as defined by the European Broadcasting Union.

General audio (GA) coding tools comprise the AAC tool set expanded by alternative quantization and coding schemes (Twin-VQ and BSAC). The general audio coder uses a perceptual filterbank, a sophisticated masking model, noise-shaping techniques, channel coupling, and noiseless coding and bit-allocation to provide the maximum compression within the constraints of providing the highest possible quality. Psychoacoustic coding standards developed by MPEG have represented the state-of-the-art in this technology since MPEG-1 Audio; MPEG-4 General Audio coding continues this tradition.

For bitrates ranging from 6 kbit/s to 64 kbit/s per channel, the MPEG-4 standard provides extensions to the GA coding tools, that allow the content author to achieve the highest quality coding at the desired bitrate. Furthermore, various bit rate scalability options are available within the GA coder. The low-bitrate techniques and scalability modes provided within this tool set have also been verified in formal tests by MPEG.

The **MPEG-4 low delay** coding functionality provides the ability to extend the usage of generic low bitrate audio coding to applications requiring a very low delay in 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 from the proven architecture of MPEG-2/4 Advanced Audio Coding (AAC) and all capabilities for coding of 2 (stereo) or more sound channels (multi-channel) are available within the low delay coder. 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 to enable coding of general audio signals with an algorithmic delay not exceeding 20 ms. 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 the case of transient signals, window shape switching is provided instead. For non-transient portions of the signal a sine window is used, while a so-called low overlap window is used for transient portions. 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.

The **MPEG-4 BSAC** is used in combination with the AAC coding tools and replaces the noiseless coding of the quantized spectral data and the scalefactors. The MPEG-4 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 stream and many small enhancement layer streams are used. 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 all most significant bits (MSB) of the quantized values in a group are processed first. These bit-slices are then encoded using an arithmetic coding scheme to obtain entropy coding with minimal redundancy. In order to implement fine grain scalability efficiently using MPEG-4 Systems tools, the fine grain audio data can be grouped into large-step layers and these large-step layers can be further grouped by concatenating large-step layers from several sub-frames. Furthermore, the configuration of the payload transmitted over an Elementary Stream (ES) can be changed dynamically (by means of the MPEG-4 backchannel capability) depending on the environment, such as network traffic or user interaction. This means that BSAC can allow for real-time adjustments to the quality of service. In addition to fine grain scalablity, it can improve the quality of an audio signal that is decoded from a stream transmitted over an error-prone channel, such as a mobile communication networks or Digital Audio Broadcasting (DAB) channel.

MPEG-4 SBR (Spectral Band Replication) is a bandwidth extension tool used in combination with the AAC general audio codec. When integrated into the MPEG AAC codec, a significant improvement of the performance is available, which can be used to lower the bitrate or improve the audio quality. This is achieved by replicating the highband, i.e. the high frequency part of the spectrum. A small amount of data representing a parametric description of the highband is encoded and used in the decoding process. The data rate is by far below the data rate required when using conventional AAC coding of the highband.

0.3.2.3 Parametric audio coding tools

The parametric audio coding tool **MPEG-4 HILN** (Harmonic and Individual Lines plus Noise) codes non-speech signals like music at bitrates of 4 kbit/s and higher 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. HILN allows independent change of speed and pitch during decoding.

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 in MPEG-4, improved overall coding efficiency is expected for applications of object based coding allowing selection and/or switching between different coding techniques.

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 in e.g. 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, especially 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 tones are described using LPC modeling as known from speech coding. Correlation between the parameters of one frame and those of consecutive frames is exploited by parameter prediction. Finally, the quantized parameters are entropy coded and multiplexed to form a bitstream payload.

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 MPEG-4 standard.

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

0.3.3 Lossless audio coding tools

MPEG-4 DST (Direct Stream Transfer) provides lossless coding of oversampled audio signals.

0.3.4 Synthesis tools

Synthesis tools are designed for very low bitrate description and transmission, and terminal-side synthesis, of synthetic music and other sounds.

The MPEG-4 toolset providing general audio synthesis capability is called **MPEG-4 Structured Audio**, and it is described in subpart 5 of ISO/IEC 14496-3. MPEG-4 Structured Audio (the SA coder) provides very general capabilities for the description of synthetic sound, and the normative creation of synthetic sound in the decoding terminal. High-quality stereo sound can be transmitted at bitrates from 0 kbit/s (no continuous cost) to 2-3 kbit/s for extremely expressive sound using these tools.

Rather than specify a particular method of synthesis, SA specifies a flexible language for describing methods of synthesis. This technique allows content authors two advantages. First, the set of synthesis techniques available is not limited to those that were envisioned as useful by the creators of the standard; any current or future method of synthesis may be used in MPEG-4 Structured Audio. Second, the creation of synthetic sound from structured descriptions is normative in MPEG-4, so sound created with the SA coder will sound the same on any terminal.

Synthetic audio is transmitted via a set of *instrument* modules that can create audio signals under the control of a *score*. An instrument is a small network of signal-processing primitives that control the parametric generation of sound according to some algorithm. Several different instruments may be transmitted and used in a single Structured Audio bitstream payload. A score is a time-sequenced set of commands that invokes various instruments at specific times to contribute their output to an overall music performance. The format for the description of instruments is SAOL, the Structured Audio Orchestra Language. The format for the description of scores is SASL, the Structured Audio Score Language.

Efficient transmission of sound samples, also called *wavetables*, for use in sampling synthesis is accomplished by providing interoperability with the MIDI Manufacturers Association Downloaded Sounds Level 2 (DLS-2) standard, which is normatively referenced by the Structured Audio standard. By using the DLS-2 format, the simple and popular technique of wavetable synthesis can be used in MPEG-4 Structured Audio soundtracks, either by itself or in conjunction with other kinds of synthesis using the more general-purpose tools. To further enable interoperability with existing content and authoring tools, the popular MIDI (Musical Instrument Digital Interface) control format can be used instead of, or in addition to, scores in SASL for controlling synthesis.

Through the inclusion of compatibility with MIDI standards, MPEG-4 Structured Audio thus represents a unification of the current technique for synthetic sound description (MIDI-based wavetable synthesis) with that of the future (general-purpose algorithmic synthesis). The resulting standard solves problems not only in very-low-bitrate coding, but also in virtual environments, video games, interactive music, karaoke systems, and many other applications.

0.3.5 Composition tools

Composition tools are designed for object-based coding, interactive functionality, and audiovisual synchronization.

The tools for audio composition, like those for visual composition, are specified in the MPEG-4 Systems standard (ISO/IEC 14496-1). However, since readers interested in audio functionality are likely to look here first, a brief overview is provided.

Audio composition is the use of multiple individual “audio objects” and mixing techniques to create a single soundtrack. It is analogous to the process of recording a soundtrack in a multichannel mix, with each musical instrument, voice actor, and sound effect on its own channel, and then “mixing down” the multiple channels to a single channel or single stereo pair. In MPEG-4, the multichannel mix itself may be transmitted, with each audio source using a different coding tool, and a set of instructions for mixdown also transmitted in the bitstream payload. As the multiple audio objects are received, they are decoded separately, but not played back to the listener; rather, the instructions for mixdown are used to prepare a single soundtrack from the “raw material” given in the objects. This final soundtrack is then played for the listener.

An example serves to illustrate the efficacy of this approach. Suppose, for a certain application, we wish to transmit the sound of a person speaking in a reverberant environment over stereo background music, at very high quality. A traditional approach to coding would demand the use of a general audio coding at 32 kbit/s/channel or above; the sound source is too complex to be well-modeled by a simple model-based coder. However, in MPEG-4 we can represent the soundtrack as the conjunction of several objects: a speaking person passed through a reverberator added to a synthetic music track. We transmit the speaker’s voice using the CELP tool at 16 kbit/s, the synthetic music using the SA tool at 2 kbit/s, and allow a small amount of overhead (only a few hundreds of bytes as a fixed cost) to describe the stereo mixdown and the reverberation. Using MPEG-4 and an object-based approach thus allows us to describe in less than 20 kbit/s total a stream that might require 64 kbit/s to transmit with traditional coding, at equivalent quality.

Additionally, having such structured soundtrack information present in the decoding terminal allows more sophisticated client-side interaction to be included. For example, the listener can be allowed (if the content author desires) to request that the background music be muted. This functionality would not be possible if the music and speech were coded into the same audio track.

With the **MPEG-4 Binary Format for Scenes (BIFS),** specified in MPEG-4 Systems, a subset tool called AudioBIFS allows content authors to describe sound scenes using this object-based framework. Multiple sources may be mixed and combined, and interactive control provided for their combination. Sample-resolution control over mixing is provided in this method. Dynamic download of custom signal-processing routines allows the content author to exactly request a particular, normative, digital filter, reverberator, or other effects-processing routine. Finally, an interface to terminal-dependent methods of 3-D audio spatialisation is provided for the description of virtual-reality and other 3-D sound material.

As AudioBIFS is part of the general BIFS specification, the same framework is used to synchronize audio and video, audio and computer graphics, or audio with other material. Please refer to ISO/IEC 14496-1 (MPEG-4 Systems) for more information on AudioBIFS and other topics in audiovisual synchronization.

0.3.6 Scalability tools

Scalability tools are designed for the creation of bitstream payloads that can be transmitted, without recoding, at several different bitrates.

Many of the stream types in MPEG-4 are *scalable* in one manner or another. Several types of scalability in the standard are discussed below.

Bitrate scalability allows a bitstream payload to be parsed into a bitstream payload of lower bitrate such that the combination can still be decoded into a meaningful signal. The bitstream payload parsing can occur either during transmission or in the decoder. Scalability is available within each of the natural audio coding schemes, or by a combination of different natural audio coding schemes.

Bandwidth scalability is a particular case of bitrate scalability, whereby part of a bitstream payload representing a part of the frequency spectrum can be discarded during transmission or decoding. This is available for the CELP speech coder, where an extension layer converts the narrow band base layer speech coder into a wide band speech coder. Also the general audio coding tools which all operate in the frequency domain offer a very flexible bandwidth control for the different coding layers.

Encoder complexity scalability allows encoders of different complexity to generate valid and meaningful bitstream payloads. An example for this is the availability of a high quality and a low complexity excitation module for the wideband CELP coder allowing to choose between significant lower encoder complexity or optimized coding quality.

Decoder complexity scalability allows a given bitstream payload to be decoded by decoders of different levels of complexity. A subtype of decoder complexity scalability is *graceful degradation*, in which a decoder dynamically monitors the resources available, and scales down the decoding complexity (and thus the audio quality) when resources are limited. The Structured Audio decoder allows this type of scalability; a content author may provide (for example) several different algorithms for the synthesis of piano sounds, and the content itself decides, depending on available resources, which one to use.

0.3.7 Upstream

Upstream tools are designed for the dynamic control the streaming of the server for bitrate control and quality feedback control.

The **MPEG-4 upstream** or backchannel allows a user on a remote side to dynamically control the streaming of MPEG-4 content from a server. Backchannel streams carrying the user interaction information.

0.3.8 Error robustness facilities

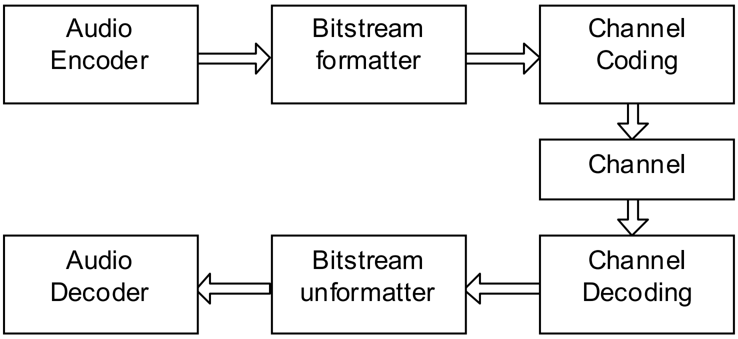
0.3.8.1 Overview

Error robustness facilities include tools for error resilience as well as for error protection.

The error robustness facilities provide improved performance on error-prone transmission channels. They are comprised of error resilient bitstream payload reordering, a common error protection tool and codec specific error resilience tools.

0.3.8.2 Error resilient bitstream payload reordering

Error resilient bitstream payload reordering allows the effective use of advanced channel coding techniques like unequal error protection (UEP), which 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 can be either data element-wise or even bit-wise. An error resilient bitstream payload frame is build by concatenating these instances.



The basic principle is depicted in the figure above. A bitstream payload is reordered according to the error sensitivity of single bitstream payload elements or even single bits. This new arranged bitstream payload is channel coded, transmitted and channel decoded. Prior to audio decoding, the bitstream payload is rearranged to its original order.

0.3.8.3 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 set of 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, both in performance and in redundancy
- providing a generic and bandwidth-efficient error protection framework, which covers both fixed-length frame streams and variable-length frame streams
- providing a UEP configuration control with low overhead.

0.3.8.4 Error resilience tools for AAC

Several tools are provided to increase the error resilience for AAC. These tools improve the perceived audio quality of the decoded audio signal in case of corrupted bitstream payloads, 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 payload. This permits the detection of serious errors within the spectral data of an MPEG-4 AAC bitstream payload.

Virtual codebooks are used to limit the largest absolute value possible within any scalefactor band that uses escape values. While all virtual codeboocks use the codebook 11, the sixteen virtual codebooks introduced by VCB11 provide sixteen different limitations of the spectral values belonging to the corresponding subclause. Therefore, errors in the transmission of spectral data that result 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 payload. 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 payload 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, the PCWs are determined by sorting the codewords according to their importance.

Information technology — Coding of audio-visual objects —

Part 3: **Audio**

ISO/IEC 14496-3 contains ten subparts:

Subpart 1: Main

Subpart 2: Speech coding — HVXC

Subpart 3: Speech coding — CELP

Subpart 4: General Audio coding (GA) — AAC, TwinVQ, BSAC

Subpart 5: Structured Audio (SA)

Subpart 6: Text To Speech Interface (TTSI)

Subpart 7: Parametric Audio Coding — HILN

Subpart 8: Parametric coding for high quality audio — SSC

Subpart 9: MPEG-1/2 Audio in MPEG-4

Subpart 10: Lossless coding of oversampled audio — DST

Content for Subpart 1

Subpart 1: Main

1.1 Scope

ISO/IEC 14496-3 (MPEG-4 Audio) is a new kind of audio International 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 International 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 International 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. A multimedia terminal that can decode the Natural Audio Profile of MPEG-4 Audio has audio capabilities that cover the entire spectrum of audio functionality available today and into the future.

1.2 Normative references

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

ISO/IEC 11172-3:1993, *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 13818-1:2000, *Information technology — Generic coding of moving pictures and associated audio information — Part 1: Systems*

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

ISO/IEC 13818-7:2004, *Information technology – Generic coding of moving pictures and associated audio information — Part 7: Advanced Audio Coding (AAC)*

ISO/IEC 14496-1, *Information technology — Coding of audio-visual objects — Part 1: Systems*

ISO/IEC 14496-11, *Information technology — Coding of audio-visual objects — Part 11: Scene description and application engine*

ITU-T Recommendation H.223/Annex C: MULTIPLEXING PROTOCOL FOR LOW BITRATE MULTIMEDIA COMMUNICATION OVER HIGHLY ERROR_PRONE CHANNELS, April 1998.

(c) 1996 MIDI Manufacturers Association, *The Complete MIDI 1.0 Detailed Specification* v. 96.2

(c) 1998 MIDI Manufacturers Association, *The MIDI Downloadable Sounds Specification* v. 98.2

1.3 Terms and definitions

1.3.1. **AAC:** Advanced Audio Coding.

1.3.2. **AAC program:** A set of main audio channels, coupling channel, lfe channel and associated data streams intended to be decoded and played back simultaneously. A program may be defined by default, or specifically by a program_config_element(). A given single_channel_element(), channel_pair_element(), coupling_channel_element(), lfe_channel_element() or data_stream_element() may accompany one or more programs in any given stream.

1.3.3. **Audio access unit**: An individually accessible portion of audio data within an elementary stream.

1.3.4. **Audio composition unit**: An individually accessible portion of the output that an audio decoder produces from audio access units.

1.3.5. **Absolute time:** The time at which sound corresponding to a particular event is really created; time in the real-world. Contrast score time.

1.3.6. **Actual parameter:** The expression which, upon evaluation, is passed to an opcode as a parameter value.

1.3.7. **A-cycle:** See audio cycle.

1.3.8. **Adaptive codebook:** An approach to encode the long-term periodicity of the signal. The entries of the codebook consists of overlapping segments of past excitations.

1.3.9. **Alias:** Mirrored spectral component resulting from sampling.

1.3.10. **Analysis filterbank:** Filterbank in the encoder that transforms a broadband PCM audio signal into a set of spectral coefficients.

1.3.11. **Ancillary data:** Part of the bitstream payload that might be used for transmission of ancillary data.

1.3.12. **API:** Application Programming Interface.

1.3.13. **A-rate:** See audio rate.

1.3.14. **Asig:** The lexical tag indicating an a-rate variable.

1.3.15. **Audio buffer:** A buffer in the system target decoder (see ISO/IEC 13818-1) for storage of compressed audio data.

1.3.16. **Audio cycle:** The sequence of processing which computes new values for all a-rate expressions in a particular code block.

1.3.17. **Audio rate:** The rate type associated with a variable, expression or statement which may generate new values as often as the sampling rate.

1.3.18. **Audio sample:** A short snippet or clip of digitally represented sound. Typically used in wavetable synthesis.

1.3.19. **AudioBIFS:** The set of tools specified in ISO/IEC 14496-1 (MPEG-4 Systems) for the composition of audio data in interactive scenes.

1.3.20. **Authoring:** In Structured Audio, the combined processes of creatively composing music and sound control scripts, creating instruments which generate and alter sound, and encoding the instruments, control scripts, and audio samples in MPEG-4 Structured Audio format.

1.3.21. **Backus-Naur Format:** (BNF) A format for describing the syntax of programming languages, used here to specify the SAOL and SASL syntax.

1.3.22. **Backward compatibility:** A newer coding standard is backward compatible with an older coding standard if decoders designed to operate with the older coding standard are able to continue to operate by decoding all or part of a bitstream payload produced according to the newer coding standard.

1.3.23. **Bandwidth scalability:** The possibility to change the bandwidth of the signal during transmission.

1.3.24. **Bank:** A set of samples used together to define a particular sound or class of sounds with wavetable synthesis.

1.3.25. **Bark:** The Bark is the standard unit corresponding to one critical band width of human hearing.

1.3.26. **Beat:** The unit in which score time is measured.

1.3.27. **Bitrate:** The rate at which the compressed stream is delivered to the input of a decoder.

1.3.28. **Bitrate scalability:** The possibility to transmit a subset of the bitstream payload and still decode the bitstream payload with the same decoder.

1.3.29. **Bitstream payload verifier:** A process by which it is possible to test and verify that all the requirements specified in ISO/IEC 14496-3 are met by the bitstream payload.

1.3.30. **Bitstream; stream:** An ordered series of bits that forms the coded representation of the data.

1.3.31. **Block companding:** Normalizing of the digital representation of an audio signal within a certain time period.

1.3.32. **BNF:** See Backus-Naur Format.

1.3.33. **BSAC:** Bit Sliced Arithmetic Coding

1.3.34. **Bus:** An area in memory which is used to pass the output of one instrument into the input of another.

1.3.35. **Byte:** Sequence of 8-bits.

1.3.36. **Byte aligned:** A bit in a data function is byte-aligned if its position is a multiple of 8-bits from the first bit of this data function.

1.3.37. **CELP:** Code Excited Linear Prediction.

1.3.38. **Center channel:** An audio presentation channel used to stabilize the central component of the frontal stereo image.

1.3.39. **Channel:** A sequence of data representing an audio signal intended to be reproduced at one listening position.

1.3.40. **Coded audio bitstream:** A coded representation of an audio signal.

1.3.41. **Coded representation:** A data element as represented in its encoded form.

1.3.42. **Composition (compositing):** Using a scene description to mix and combine several separate audio tracks into a single presentation.

1.3.43. **Compression:** Reduction in the number of bits used to represent an item of data.

1.3.44. **Constant bitrate:** Operation where the bitrate is constant from start to finish of the coded bitstream.

1.3.45. **Context:** See state space.

1.3.46. **Control:** An instruction used to describe how to use a particular synthesis method to produce sound.

EXAMPLES:
"Using the piano instrument, play middle C at medium volume for 2 seconds."
"Glissando the violin instrument up to middle C."
"Turn off the reverberation for 8 seconds."

1.3.47. **Control cycle:** The sequence of processing which computes new values for all control-rate expressions in a particular code block.

1.3.48. **Control period:** The length of time (typically measured in audio samples) corresponding to the control rate.

1.3.49. **Control rate:** (1) The rate at which instantiation and termination of instruments, parametric control of running instrument instances, sharing of global variables, and other non-sample-by-sample computation occurs in a particular orchestra. (2) The rate type of variables, expressions, and statements that can generate new values as often as the control rate.

1.3.50. **Core coder:** The term core coder is used to denote a base layer coder in certain scalability configurations. A core coder does not code the spectral samples of the MDCT filterbank of the subsequent AAC coding layers, but operates on a time domain signal. The output of the core decoder has to be up-sampled and transformed into the spectral domain, before it can be combined with the output of the AAC coding layers. Within the MPEG-4 Audio standard only the MPEG-4 CELP coder is a valid core coder. However, in principal, also another AAC coding layer, operating at a lower sampling rate, could be used on the time domain signal, and then combined with the other coding layer in exactly the same way as described for the CELP coder, and would therefore be called a core coder.

1.3.51. **CRC:** The Cyclic Redundancy Check to verify the correctness of data.

1.3.52. **Critical band:** This unit of bandwidth represents the standard unit of bandwidth expressed in human auditory terms, corresponding to a fixed length on the human cochlea. It is approximately equal to 100 Hz at low frequencies and 1/3 octave at higher frequencies, above approximately 700 Hz.

1.3.53. **Data element:** An item of data as represented after encoding and before decoding.

1.3.54. **Data function:** An encapsulation of data elements forming a logic unit.

1.3.55. **Decoded stream:** The decoded reconstruction of a compressed bitstream.

1.3.56. **Decoder:** An embodiment of a decoding process.

1.3.57. **Decoding (process):** The process that reads an input coded bitstream payload and outputs decoded audio samples.

1.3.58. **Demultiplexing:** Splitting one bitstream into several.

1.3.59. **DFT:** Discrete Fourier Transform.

1.3.60. **Digital storage media; DSM:** A digital storage or transmission device or system.

1.3.61. **Dimension conversion:** A method to convert a dimension of a vector by a combination of low pass filtering and linear interpolation.

1.3.62. **Discrete cosine transform; DCT:** Either the forward discrete cosine transform or the inverse discrete cosine transform. The DCT is an invertible, discrete orthogonal transformation.

1.3.63. **Downmix:** A matrixing of n channels to obtain less than n channels.

1.3.64. **DST:** Direct Stream Transfer

1.3.65. **Duration:** The amount of time between instantiation and termination of an instrument instance.

1.3.66. **Editing:** The process by which one or more coded bitstreams are manipulated to produce a new coded bitstream. Conforming edited bitstreams must meet the requirements defined in part 3 of ISO/IEC 14496.

1.3.67. **Elementary stream (ES):** A sequence of data that originates from a single producer in the transmitting MPEG-4 Terminal and terminates at a single recipient, e.g. an AVObject or a Control Entity in the receiving MPEG-4 Terminal. It flows through one FlexMux Channel.

1.3.68. **Encoder:** An embodiment of an encoding process.

1.3.69. **Encoding (process):** A process, not specified in ISO/IEC 14496, that reads a stream of input audio samples and produces a valid coded bitstream payload as defined in part 3 of ISO/IEC 14496.

1.3.70. **Enhancement layer(s):** The part(s) of the bitstream payload that is possible to drop in a transmission and still decode the bitstream payload.

1.3.71. **Entropy coding:** Variable length lossless coding of the digital representation of a signal to reduce statistical redundancy.

1.3.72. **Envelope:** A loudness-shaping function applied to a sound, or more generally, any function controlling a parametric aspect of a sound

1.3.73. **EP:** Error Protection

1.3.74. **ER:** Error resilience or Error Resilient (as appropriate)

1.3.75. **Event:** One control instruction.

1.3.76. **Excitation:** The excitation signal represents the input to the LPC module. The signal consists of contributions that cannot be covered by the LPC model.

1.3.77. **Expression:** A mathematical or functional combination of variable values, symbolic constants, and opcode calls.

1.3.78. **FFT:** Fast Fourier Transformation. A fast algorithm for performing a discrete Fourier transform (an orthogonal transform).

1.3.79. **Filterbank:** A set of band-pass filters covering the entire audio frequency range.

1.3.80. **Fine rate control:** The possibility to change the bitrate by, under some circumstances, skipping transmission of the LPC indices.

1.3.81. **Fixed codebook:** The fixed codebook contains excitation vectors for the speech synthesis filter. The contents of the codebook are non-adaptive (i.e. fixed).

1.3.82. **Flag:** A variable which can take one of only the two values defined in this specification.

1.3.83. **Formal parameter:** The syntactic element that gives a name to one of the parameters of an opcode.

1.3.84. **Forward compatibility:** A newer coding standard is forward compatible with an older coding standard if decoders designed to operate with the newer coding standard are able to decode bitstream payloads of the older coding standard.

1.3.85. **Frame**: A part of the audio signal that corresponds to a certain number of audio PCM samples.

1.3.86. **Fs:** Sampling frequency.

1.3.87. **FSS:** Frequency Selective Switch. Module which selects one of two input signals independently in each scalefactor band.

1.3.88. **Fundamental frequency:** A parameter which represents signal periodicity in frequency domain.

1.3.89. **Future wavetable:** A wavetable that is declared but not defined in the SAOL orchestra; its definition must arrive in the bitstream payload before it is used.

1.3.90. **Global block:** The section of the orchestra that describes global variables, route and send statements, sequence rules, and global parameters.

1.3.91. **Global context:** The state space used to hold values of global variables and wavetables.

1.3.92. **Global parameters:** The sampling rate, control rate, and number of input and output channels of audio associated with a particular orchestra.

1.3.93. **Global variable:** A variable that can be accessed and/or changed by several different instruments.

1.3.94. **Grammar:** A set of rules that describes the set of allowable sequences of lexical elements comprising a particular language.

1.3.95. **Guard expression:** The expression standing at the front of an if, while, or else statement that determines whether or how many times a particular block of code is executed.

1.3.96. **Hann window:** A time function applied sample-by-sample to a block of audio samples before Fourier transformation.

1.3.97. **Harmonic lines:** A set of spectral components having a common fundamental frequency.

1.3.98. **Harmonic magnitude:** Magnitude of each harmonic.

1.3.99. **Harmonic synthesis:** A method to obtain a periodic excitation from harmonic magnitudes.

1.3.100. **Harmonics:** Samples of frequency spectrum at multiples of the fundamental frequency.

1.3.101. **HCR:** Huffman codebook reordering

1.3.102. **HILN:** Harmonic and Individual Lines plus Noise (parametric audio coding).

1.3.103. **Huffman coding:** A specific method for entropy coding.

1.3.104. **HVXC:** Harmonic Vector eXcitation Coding (parametric speech coding).

1.3.105. **Hybrid filterbank:** A serial combination of subband filterbank and MDCT. Used in MPEG-1 and MPEG-2 Audio.

1.3.106. **I-cycle:** See initialisation cycle.

1.3.107. **IDCT:** Inverse Discrete Cosine Transform.

1.3.108. **Identifier:** A sequence of characters in a textual SAOL program that denotes a symbol.

1.3.109. **IFFT:** Inverse Fast Fourier Transform.

1.3.110. **IMDCT:** Inverse Modified Discrete Cosine Transform.

1.3.111. **Index:** Number indicating the quantized value(s).

1.3.112. **Individual line:** A spectral component described by frequency, amplitude and phase.

1.3.113. **Informative:** Aspects of a standards document that are provided to assist implementers, but are not required to be implemented in order for a particular system to be compliant to the standard.

1.3.114. **Initial phase:** A phase value at the onset of voiced signal in harmonic synthesis.

1.3.115. **Initialisation cycle:** See initialisation pass.

1.3.116. **Initialisation pass:** The sequence of processing that computes new values for each i-rate expression in a particular code block.

1.3.117. **Initialisation rate:** The rate type of variables, expressions, and statements that are set once at instrument instantiation and then do not change.

1.3.118. **Instance:** See instrument instantiation.

1.3.119. **Instantiation:** The process of creating a new instrument instantiation based on an event in the score or statement in the orchestra.

1.3.120. **Instrument:** An algorithm for parametric sound synthesis, described using SAOL. An instrument encapsulates all of the algorithms needed for one sound-generation element to be controlled with a score.

NOTE - An MPEG-4 Structured Audio instrument does not necessarily correspond to a real-world instrument. A single instrument might be used to represent an entire violin section, or an ambient sound such as the wind. On the other hand, a single real-world instrument that produces many different timbres over its performance range might be represented using several SAOL instruments.

1.3.121. **Instrument instantiation:** The state space created as the result of executing a note-creation event with respect to a SAOL orchestra.

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

1.3.123. **Interframe prediction:** A method to predict a value in the current frame from values in the previous frames. Interframe prediction is used in VQ of LSP.

1.3.124. **International Phonetic Alphabet; IPA :** The worldwide agreed symbol set to represent various phonemes appearing in human speech.

1.3.125. **I-pass:** See initialisation pass.

1.3.126. **IPQF:** inverse polyphase quadrature filter

1.3.127. **I-rate:** See initialisation rate.

1.3.128. **Ivar:** The lexical tag indicating an i-rate variable.

1.3.129. **Joint stereo coding:** Any method that exploits stereophonic irrelevance or stereophonic redundancy.

1.3.130. **Joint stereo mode:** A mode of the audio coding algorithm using joint stereo coding.

1.3.131. **K-cycle:** See control cycle.

1.3.132. **K-rate:** See control rate.

1.3.133. **Ksig:** The lexical tag indicating a k-rate variable.

1.3.134. **Lexical element:** See token.

1.3.135. **Lip shape pattern :** A number that specifies a particular pattern of the preclassified lip shape.

1.3.136. **Lip synchronization :** A functionality that synchronizes speech with corresponding lip shapes.

1.3.137. **Looping:** A typical method of wavetable synthesis. Loop points in an audio sample are located and the sound between those endpoints is played repeatedly while being simultaneously modified by envelopes, modulators, etc.

1.3.138. **Low frequency enhancement (LFE) channel:** A limited bandwidth channel for low frequency audio effects in a multichannel system.

1.3.139. **LPC:** Linear Predictive Coding.

1.3.140. **LPC residual signal:** A signal filtered by the LPC inverse filter, which has a flattened frequency spectrum.

1.3.141. **LPC synthesis filter:** An IIR filter whose coefficients are LPC coefficients. This filter models the time varying vocal tract.

1.3.142. **LSP:** Line Spectral Pairs.

1.3.143. **LTP:** Long Term Prediction.

1.3.144. **M/S stereo:** A method of removing imaging artifacts as well as exploiting stereo irrelevance or redundancy in stereophonic audio programs based on coding the sum and difference signal instead of the left and right channels.

1.3.145. **Main audio channels:** All single_channel_elements or channel_pair_elements in one program.

1.3.146. **Mapping:** Conversion of an audio signal from time to frequency domain by subband filtering and/or by MDCT.

1.3.147. **Masking:** A property of the human auditory system by which an audio signal cannot be perceived in the presence of another audio signal.

1.3.148. **Masking threshold:** A function in frequency and time below which an audio signal cannot be perceived by the human auditory system.

1.3.149. **MIDI:** The Musical Instrument Digital Interface standards. Certain aspects of the MPEG-4 Structured Audio tools provide interoperability with MIDI standards.

1.3.150. **Mixed voiced frame:** A speech segment which has both voiced and unvoiced components.

1.3.151. **Modified discrete cosine transform (MDCT):** A transform which has the property of time domain aliasing cancellation.

1.3.152. **Moving picture dubbing :** A functionality that assigns synthetic speech to the corresponding moving picture while utilizing lip shape pattern information for synchronization.

1.3.153. **MPE:** Multi Pulse Excitation.

1.3.154. **MPEG-4 Audio Text-to-Speech Decoder :** A device that produces synthesized speech by utilizing the M-TTS bitstream payload while supporting all the M-TTS functionalities such as speech synthesis for FA and MP dubbing.

1.3.155. **M-TTS sentence :** This defines the information such as prosody, gender, and age for only the corresponding sentence to be synthesized.

1.3.156. **M-TTS sequence :** This defines the control information which affects all M-TTS sentences that follow this M-TTS sequence.

1.3.157. **Multichannel:** A combination of audio channels used to create a spatial sound field.

1.3.158. **Multilingual:** A presentation of dialogue in more than one language.

1.3.159. **Multiplexing:** Combining several bitstream payloads into one.

1.3.160. **Natural Sound:** A sound created through recording from a real acoustic space. Contrasted with synthetic sound.

1.3.161. **NCC:** Number of Considered Channels. In case of AAC, it is the number of channels represented by the elements SCE, independently switched CCE and CPE, i.e. once the number of SCEs plus once the number of independently switched CCEs plus twice the number of CPEs. With respect to the naming conventions of the MPEG-AAC decoders and payloads, NCC=A+I. This number is used to derive the required decoder input buffer size (see subpart 4, subclause 4.5.3.1). In case of other codecs, it tis the total number of channels.

1.3.162. **Noise component:** A signal component modeled as noise.

1.3.163. **Non-tonal component:** A noise-like component of an audio signal.

1.3.164. **Normative:** Those aspects of a standard that must be implemented in order for a particular system to be compliant to the standard.

1.3.165. **Nyquist sampling:** Sampling at or above twice the maximum bandwidth of a signal.

1.3.166. **OD:** Object Descriptor.

1.3.167. **Opcode:** A parametric signal-processing function that encapsulates a certain functionality so that it may be used by several instruments.

1.3.168. **Orchestra:** The set of sound-generation and sound-processing algorithms included in an MPEG-4 bitstream payload. Includes instruments, opcodes, routing, and global parameters.

1.3.169. **Orchestra cycle:** A complete pass through the orchestra, during which new instrument instantiations are created, expired ones are terminated, each instance receives one k-cycle and one control period worth of a-cycles, and output is produced.

1.3.170. **Padding:** A method to adjust the average length of an audio frame in time to the duration of the corresponding PCM samples, by conditionally adding a slot to the audio frame.

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

1.3.172. **Parameter fields:** The names given to the parameters to an instrument.

1.3.173. **Parser:** Functional stage of a decoder which extracts from a coded bitstream payload a series of bits representing coded elements.

1.3.174. **P-fields:** See parameter fields.

1.3.175. **Phoneme/bookmark-to-FAP converter :** A device that converts phoneme and bookmark information to FAPs.

1.3.176. **Pi:** The constant π = 3.14159...

1.3.177. **Pitch:** A parameter which represents signal periodicity in the time domain. It is expressed in terms of the number of samples.

1.3.178. **Pitch control:** A functionality to control the pitch of the synthesized speech signal without changing its speed.

1.3.179. **PNS:** Perceptual Noise Substitution.

1.3.180. **Polyphase filterbank:** A set of equal bandwidth filters with special phase interrelationships, allowing an efficient implementation of the filterbank.

1.3.181. **Postfilter:** A filter to enhance the perceptual quality of the synthesized speech signal.

1.3.182. **PQF:** polyphase quadrature filter

1.3.183. **Prediction:** The use of a predictor to provide an estimate of the sample value or data element currently being decoded.

1.3.184. **Prediction error:** The difference between the actual value of a sample or data element and its predictor.

1.3.185. **Predictor:** A linear combination of previously decoded sample values or data elements.

1.3.186. **Presentation channel:** An audio channel at the output of the decoder.

1.3.187. **Production rule:** In Backus-Naur Form grammars, a rule that describes how one syntactic element may be expressed in terms of other lexical and syntactic elements.

1.3.188. **PSNR:** Peak Signal to Noise Ratio.

1.3.189. **Psychoacoustic model:** A mathematical model of the masking behaviour of the human auditory system.

1.3.190. **Random access:** The process of beginning to read and decode the coded stream at an arbitrary point.

1.3.191. **Rate semantics:** The set of rules describing how rate types are assigned to variables, expressions, statements, and opcodes, and the normative restrictions that apply to a bitstream regarding combining these elements based on their rate types.

1.3.192. **Rate type:** The “speed of execution” associated with a particular variable, expression, statement, or opcode.

1.3.193. **Rate-mismatch error:** The condition that results when the rate semantics rules are violated in a particular SAOL construction. A type of syntax error.

1.3.194. **Reserved:** The term "reserved" when used in the subclauses defining the coded bitstream payload indicates that the value may be used in the future for ISO/IEC defined extensions.

1.3.195. **Route statement:** A statement in the global block that describes how to place the output of a certain set of instruments onto a bus.

1.3.196. **RPE:** Regular Pulse Excitation.

1.3.197. **Run-time error:** The condition that results from improper calculations or memory accesses during execution of a SAOL orchestra.

1.3.198. **RVLC:** Reversible Variable Length Coding

1.3.199. **Sample:** See Audio sample.

1.3.200. **Sample Bank Format:** A component format of MPEG-4 Structured Audio that allows the description of a set of samples for use in wavetable synthesis and processing methods to apply to them.

1.3.201. **Sampling Frequency (Fs):** Defines the rate in Hertz which is used to digitize an audio signal during the sampling process.

1.3.202. **SAOL:** The Structured Audio Orchestra Language, pronounced like the English word “sail”. SAOL is a digital-signal processing language that allows for the description of arbitrary synthesis and control algorithms as part of the content bitstream payload.

1.3.203. **SAOL orchestra:** See orchestra.

1.3.204. **SASBF:** The MPEG-4 Structured Audio Sample Bank Format, an efficient format for the transmission of blocks of wavetable (sample data) compatible with the MIDI method for the same.

1.3.205. **SASL:** The Structured Audio Score Language. SASL is a simple format that allows for powerful and flexible control of music and sound synthesis.

1.3.206. **SBA:** Segmented Binary Arithmetic Coding which is the error resilient tool for BSAC.

1.3.207. **SBR**: Spectral Band Replication.

1.3.208. **Scalefactor:** Factor by which a set of values is scaled before quantization.

1.3.209. **Scalefactor band:** A set of spectral coefficients which are scaled by one scalefactor.

1.3.210. **Scalefactor index:** A numerical code for a scalefactor.

1.3.211. **Scheduler:** The component of MPEG-4 Structured Audio that describes the mapping from control instructions to sound synthesis using the specified synthesis techniques. The scheduler description provides normative bounds on event-dispatch times and responses.

1.3.212. **Scope :** The code within which access to a particular variable name is allowed.

1.3.213. **Score:** A description in some format of the sequence of control parameters needed to generate a desired music composition or sound scene. In MPEG-4 Structured Audio, scores are described in SASL and/or MIDI.

1.3.214. **Score time:** The time at which an event happens in the score, measured in beats. Score time is mapped to absolute time by the current tempo.

1.3.215. **Semantics:** The rules describing what a particular instruction or bitstream payload element should do. Most aspects of bitstream payload and SAOL semantics are normative in MPEG-4.

1.3.216. **Send statement:** A statement in the global block that describes how to pass a bus on to an effect instrument for post-processing.

1.3.217. **Sequence rules:** The set of rules, both default and explicit, given in the global block that define in what order to execute instrument instantiations during an orchestra cycle.

1.3.218. **SIAQ**: Scalable Inverse AAC Quantization Module.

1.3.219. **Side information:** Information in the bitstream payload necessary for controlling the decoder.

1.3.220. **Signal variable:** A unit of memory, labelled with a name, that holds intermediate processing results. Each signal variable in MPEG-4 Structured Audio is instantaneously representable by a 32-bit floating point value.

1.3.221. **Sinusoidal synthesis:** A method to obtain a time domain waveform by a sum of amplitude modulated sinusoidal waveforms.

1.3.222. **Spatialisation:** The process of creating special sounds that a listener perceives as emanating from a particular direction.

1.3.223. **Spectral coefficients:** Discrete frequency domain data output from the analysis filterbank.

1.3.224. **Spectral envelope:** A set of harmonic magnitudes.

1.3.225. **Speed control:** A functionality to control the speed of the synthesized speech signal without changing its pitch or phonemes.

1.3.226. **Spreading function:** A function that describes the frequency spread of masking effects.

1.3.227. **SQ:** Scalar Quantization.

1.3.228. **SSC**: SinuSoidal Coding, parametric coder for high quality audio.

1.3.229. **State space:** A set of variable-value associations that define the current computational state of an instrument instantiation or opcode call. All the “current values” of the variables in an instrument or opcode call.

1.3.230. **Statement:** “One line” of a SAOL orchestra.

1.3.231. **Stereo-irrelevant:** A portion of a stereophonic audio signal which does not contribute to spatial perception.

1.3.232. **Structured audio:** Sound-description methods that make use of high-level models of sound generation and control. Typically involving synthesis description, structured audio techniques allow for ultra-low bitrate description of complex, high-quality sounds.

1.3.233. **Stuffing (bits); stuffing (bytes):** Code-words that may be inserted at particular locations in the coded bitstream payload that are discarded in the decoding process. Their purpose is to increase the bitrate of the stream which would otherwise be lower than the desired bitrate.

1.3.234. **Surround channel:** An audio presentation channel added to the front channels (L and R or L, R, and C) to enhance the spatial perception.

1.3.235. **Symbol:** A sequence of characters in a SAOL program, or a symbol token in a MPEG-4 Structured Audio bitstream payload, that represents a variable name, instrument name, opcode name, table name, bus name, etc.

1.3.236. **Symbol table:** In an MPEG-4 Structured Audio bitstream payload, a sequence of data that allows the tokenised representation of SAOL and SASL code to be converted back to a readable textual representation. The symbol table is an optional component.

1.3.237. **Symbolic constant:** A floating-point value explicitly represented as a sequence of characters in a textual SAOL orchestra, or as a token in a bitstream payload.

1.3.238. **Syncword:** A code embedded in audio transport streams that identifies the start of a transport frame.

1.3.239. **Syntax:** The rules describing what a particular instruction or bitstream payload element should look like. All aspects of bitstream payload and SAOL syntax are normative in MPEG-4.

1.3.240. **Syntax error:** The condition that results when a bitstream payload element does not comply with its governing rules of syntax.

1.3.241. **Synthesis:** The process of creating sound based on algorithmic descriptions.

1.3.242. **Synthesis filterbank:** Filterbank in the decoder that reconstructs a PCM audio signal from subband samples.

1.3.243. **Synthetic Sound:** Sound created through synthesis.

1.3.244. **Tempo:** The scaling parameter that specifies the relationship between score time and absolute time. A tempo of 60 beats per minute means that the score time measured in beats is equivalent to the absolute time measured in seconds; higher numbers correspond to faster tempi, so that 120 beats per minute is twice as fast.

1.3.245. **Terminal:** The “client side” of an MPEG transaction; whatever hardware and software are necessary in a particular implementation to allow the capabilities described in this document.

1.3.246. **Termination:** The process of destroying an instrument instantiation when it is no longer needed.

1.3.247. **Text-to-speech synthesizer :** A device producing synthesized speech according to the input sentence character strings.

1.3.248. **Timbre:** The combined features of a sound that allow a listener to recognise such aspects as the type of instrument, manner of performance, manner of sound generation, etc. Those aspects of sound that distinguish sounds equivalent in pitch and loudness.

1.3.249. **TNS:** Temporal Noise Shaping

1.3.250. **Token:** A lexical element of a SAOL orchestra a keyword, punctuation mark, symbol name, or symbolic constant.

1.3.251. **Tokenisation:** The process of converting a orchestra in textual SAOL format into a bitstream payload representation consisting of a stream of tokens.

1.3.252. **Tonal component:** A sinusoid-like component of an audio signal.

1.3.253. **Trick mode :** A set of functions that enables stop, play, forward, and backward operations for users.

1.3.254. **TTSI:** Text to Speech Interface.

1.3.255. **TwinVQ:** Transform domain Weighted Interleave Vector Quantization.

1.3.256. **Unvoiced frame:** Frame containing unvoiced speech which looks like random noise with no periodicity.

1.3.257. **V/UV decision:** Decision whether the current frame is voiced or unvoiced or mixed voiced.

1.3.258. **Variable:** See signal variable.

1.3.259. **Variable bitrate:** Operation where the bitrate varies with time during the decoding of a coded stream.

1.3.260. **Variable length code (VLC):** A code word assigned by variable length encoder (See variable length coding).

1.3.261. **Variable length coding:** A reversible procedure for coding that assigns shorter code-words to frequent symbols and longer code-words to less frequent symbols.

1.3.262. **Variable length decoder:** A procedure to obtain the symbols encoded with a variable length coding technique.

1.3.263. **Variable length encoder:** A procedure to assign variable length codewords to symbols.

1.3.264. **VCB11:** Virtual Codebooks for codebook 11.

1.3.265. **Vector quantizer:** Tool that quantizes several values to one index.

1.3.266. **Virtual codebook:** If several codebook values refer to one and the same physical codebook, these values are called virtual codebooks.

1.3.267. **Voiced frame:** A voiced speech segment is known by its relatively high energy content, but more importantly it contains periodicity which is called the pitch of the voiced speech.

1.3.268. **VQ:** Vector Quantization.

1.3.269. **VXC:** Vector eXcitation Coding. It is also called CELP (Coded Excitation Linear Prediction). In HVXC, no adaptive codebook is used.

1.3.270. **Wavetable synthesis:** A synthesis method in which sound is created by simple manipulation of audio samples, such as looping, pitch-shifting, enveloping, etc.

1.3.271. **White Gaussian noise:** A noise sequence which has a Gaussian distribution.

1.3.272. **Width:** The number of channels of data that an expression represents.

1.4 Symbols and abbreviations

The mathematical operators used in this part of ISO/IEC 14496 are similar to those used in the C programming language. However, integer division with truncation and rounding are specifically defined. The bitwise operators are defined assuming two's-complement representation of integers. Numbering and counting loops generally begin from zero.

1.4.1 Arithmetic operators

+	Addition.
–	Subtraction (as a binary operator) or negation (as a unary operator).
++	Increment.
– –	Decrement.
*	Multiplication.
^	Power.
/	Integer division with truncation of the result toward zero. For example, 7/4 and –7/–4 are truncated to 1 and –7/4 and 7/–4 are truncated to –1.
//	Integer division with rounding to the nearest integer. Half-integer values are rounded away from zero unless otherwise specified. For example 3//2 is rounded to 2, and –3//2 is rounded to –2.
DIV	Integer division with truncation of the result towards $-\infty$.
| |	Absolute value. $\vert x \vert = x$ when $x > 0$ $\vert x \vert = 0$ when $x == 0$ $\vert x \vert = -x$ when $x < 0$
%	Modulus operator. Defined only for positive numbers.

Sign( )	Sign.	Sign(x) = 1 when x > 0 Sign(x) = 0 when x == 0 Sign(x) = –1 when x < 0

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

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

sin Sine.

cos Cosine.

exp Exponential.

√ Square root.

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

$\log_{e}$ Logarithm to base e.

$\log_{2}$ Logarithm to base 2.

ceil( ) Ceiling operator. Returns the smallest integer that is greater than or equal to the real-valued argument.

1.4.2 Logical operators

|| Logical OR.

&& Logical AND.

! Logical NOT

1.4.3 Relational operators

> Greater than.

>= Greater than or equal to.

< Less than.

<= Less than or equal to.

== Equal to.

!= Not equal to.

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

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

1.4.4 Bitwise operators

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

& AND

| OR

>> Shift right with sign extension.

<< Shift left with zero fill.

1.4.5 Assignment

= Assignment operator.

1.4.6 Mnemonics

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

bslbf	Bit string, left bit first, where "left" is the order in which bit strings are written in ISO/IEC 14496. Bit strings are written as a string of 1s and 0s within single quote marks, e.g. '1000 0001'. Blanks within a bit string are for ease of reading and have no significance.
L, C, R, LS, RS	Left, center, right, left surround and right surround audio signals
rpchof	Remainder polynomial coefficients, highest order first. (Audio)
uimsbf	Unsigned integer, most significant bit first.
vlclbf	Variable length code, left bit first, where "left" refers to the order in which the VLC codes are written.
window	Number of the actual time slot in case of block_type == 2, 0 <= window <= 2. (Audio)

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

1.4.7 Constants

π 3,14159265358...

e 2,71828182845...

1.4.8 Method of describing bitstream payload syntax

The bitstream payload retrieved by the decoder is described in the syntax section of each subpart. Each data item in the bitstream payload is in bold type.

It is described by

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

Data elements forming a logic unit are encapsulated in data functions.

data_function ( ); Data function call.

data_function ( ) {
. . .
}
	Data function entity.

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

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

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

do { **data_element** . . . } while (condition)	The data element always occurs at least once. The data element is repeated until the condition is not true.

if (condition) {
data_element
. . .
}
	If the condition is true, then the first group of data elements occurs next in the data stream

else {
data_element
. . .
}
	If the condition is not true, then the second group of data elements occurs next in the data stream.

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

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

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

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

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

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

While the syntax is expressed in procedural terms, it should not be assumed that this implements a satisfactory decoding procedure. In particular, it defines a correct and error-free input bitstream payload. Actual decoders must include a means to deal with incorrect bitstream payloads and to find the start of the described elements.

Definition of nextbits function

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

1.5 Technical overview

1.5.1 MPEG-4 audio object types

1.5.1.1 Audio object type definition

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	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	DST	Remark
0	Null																																							
1	AAC main		X	X		X		X		X	X	X	X	X				X																						2)
2	AAC LC		X	X		X		X		X	X		X	X				X																						
3	AAC SSR	X	X	X			X	X		X	X		X	X				X																						
4	AAC LTP		X	X		X		X	X	X	X		X	X				X																						2)
5	SBR																																X							
6	AAC Scalable		X	X		X		X	X	X			X	X	X	X	X	X																						6)

Object Type ID	Audio Object Type	gain control	block switching	window shapes - standard	window shapes – AAC LD	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	DST	Remark
7	TwinVQ		X	X		X		X	X					X					X																					
8	CELP																							X																
9	HVXC																									X														
10	(reserved)																																							
11	(reserved)																																							
12	TTSI																															X								
13	Main synthetic																											X	X	X										3)
14	Wavetable synthesis																												X	X										4)
15	General MIDI																													X										
16	Algorithmic Synthesis and Audio FX																											X												
17	ER AAC LC		X	X		X		X		X			X	X				X			X	X	X																	
18	(reserved)																																							
19	ER AAC LTP		X	X		X		X	X	X			X	X				X			X	X	X																	5)
20	ER AAC scalable		X	X		X		X		X			X	X	X	X	X	X			X	X	X																	6)
21	ER TwinVQ		X	X		X		X						X					X			X	X																	
22	ER BSAC		X	X		X		X		X			X	X						X		X	X																	
23	ER AAC LD				X	X		X	X	X			X	X				X			X	X	X																	
24	ER CELP																					X	X	X	X															
25	ER HVXC																					X	X			X	X													
26	ER HILN																					X	X								X									
27	ER Parametric																					X	X			X	X				X									
28	SSC																																				X	X		
29	(reserved)																																							
30	(reserved)																																							
31	(escape)																																							
32	Layer-1																																	X						
33	Layer-2																																		X					
34	Layer-3																																			X				
35	DST																																						X	
36 - 95	(reserved)																																							

Notes:

1) The bit parsing function is mandatory on decoder site. However, the error detection and error correction functions are optional.

2) Contains AAC LC.

3) Contains Wavetable synthesis and Algorithmic Synthesis and Audio FX.

4) Contains General MIDI.

5) Contains ER AAC LC.

6) The upsampling filter tool is only required in combination with a core coder.

1.5.1.2 Description

1.5.1.2.1 NULL object type

The NULL object provides the possibility to feed raw PCM data directly to the audio compositor. No decoding is involved, However, an audio object descriptor is used to specify the sampling rate and the audio channel configuration.

1.5.1.2.2 AAC - Main object type

The AAC Main object is very similar to the AAC Main Profile that is defined in ISO/IEC 13818-7. However, additionally the PNS tool is available. The restrictions of the AAC Main profile with respect to multiple programs and mixdown elements also apply to the AAC Main object type. The AAC Main object type bitstream payload syntax is compatible with the syntax defined in ISO/IEC 13818-7. All the MPEG-2 AAC multi-channel capabilities are available. A decoder capable to decode a MPEG-4 main object stream can also parse and decode a MPEG-2 AAC raw data stream. On the other hand, although a MPEG-2 AAC coder can parse an MPEG-4 AAC Main bitstream payload, decoding may fail, since PNS might have been used.

1.5.1.2.3 AAC - Low Complexity (LC) object type

The MPEG-4 AAC Low Complexity object type is the counterpart to the MPEG-2 AAC Low Complexity Profile, with exactly the same restrictions as mentioned above for the AAC Main object type.

1.5.1.2.4 AAC - Scalable Sampling Rate (SSR) object type

The MPEG-4 AAC Scalable Sampling Rate object type is the counterpart to the MPEG-2 AAC Scalable Sampling Rate Profile, with exactly the same restrictions as mentioned above for the AAC Main object type.

1.5.1.2.5 AAC - Long Term Predictor (LTP) object type

The MPEG-4 AAC LTP object type is similar to the AAC Main object type. However, a Long Term Predictor replaces the MPEG-2 AAC predictor. The LTP achieves a similar coding gain, but requires significantly lower implementation complexity. The bitstream payload syntax for this object type is very similar to the syntax defined in ISO/IEC 13818-7. An MPEG-2 AAC LC profile bitstream payload can be decoded without restrictions using an MPEG-4 AAC LTP object decoder.

1.5.1.2.6 SBR object type

The SBR Object contains the SBR-Tool and can be combined with the audio object types indicated in Table 1.2.

Table 1.2 — Audio object types that can be combined with the SBR Tool

Audio Object Type	Combination with SBR Tool permitted	Object Type ID
Null		0
AAC main	X	1
AAC LC	X	2
AAC SSR	X	3
AAC LTP	X	4
SBR		5
AAC Scalable	X	6
TwinVQ		7
CELP		8
HVXC		9
(Reserved)		10
(Reserved)		11
TTSI		12
Main synthetic		13
Wavetable synthesis		14
General MIDI		15
Algorithmic Synthesis and Audio FX		16
ER AAC LC	X	17
(Reserved)		18
ER AAC LTP	X	19
ER AAC scalable	X	20
ER TwinVQ		21
ER BSAC		22
ER AAC LD		23
ER CELP		24
ER HVXC		25
ER HILN		26
ER Parametric		27
SSC		28
(Reserved)		29
(Reserved)		30
(Reserved)		31

1.5.1.2.7 AAC Scalable object type

The scalable AAC object uses a different bitstream payload syntax to support bitrate- and bandwidth- scalability. A large number of scalable combinations are available, including combinations with TwinVQ and CELP coder tools. However, only mono, or 2-channel stereo objects are supported.

1.5.1.2.8 TwinVQ object type

The TwinVQ object belongs to the GA coding scheme that quantizes the MDCT coefficients. This coding scheme is based on fixed rate vector quantization instead of Huffman coding in AAC.

Low bit rate mono and stereo audio coding is available. Scalable audio coding schemes are also available in the Scalable Audio Profile combined with AAC scalable object type.

1.5.1.2.9 CELP object type

The CELP object is supported by the CELP speech coding tools, which provide coding at 8 kHz and 16 kHz sampling rate at bit rates in the range 4-24 kbit/s. Additionally, bit rate scalability and bandwidth scalability are available in order to provide scalable decoding of CELP streams. CELP Object always contains exactly one mono audio signal.

1.5.1.2.10 HVXC object type

The HVXC object is supported by the parametric speech coding (HVXC) tools, which provide fixed bitrate modes (2.0 - 4.0 kbit/s) in a scalable and a non-scalable scheme, a variable bitrate mode (< 2.0 kbit/s) and the functionality of pitch and speed change. Only 8 kHz sampling rate and mono audio channel are supported.

1.5.1.2.11 TTSI object type

The TTSI object is supported by the TTSI tools described in subpart 6. It allows very-low-bitrate phonemic descriptions of speech to be transmitted in the bitstream payload and then synthesized into sound. No particular speech synthesis method is specified in MPEG-4; rather, the TTSI tools specify an *interface* to non-normative synthesis methods. This method has a bit rate ranging 200 ~ 1200 bit/s. The TTSI object also supports synchronization of the synthesized speech with the facial animation defined in ISO/IEC 14496-2.

1.5.1.2.12 Main Synthetic object type

The main synthetic object allows the use of all MPEG-4 Structured Audio tools (described in subpart 5 of this standard). It supports flexible, high-quality algorithmic synthesis using the SAOL music-synthesis language; efficient wavetable synthesis with the SASBF sample-bank format; and enables the use of high-quality mixing and postproduction in the Systems AudioBIFS toolset. Sound can be described at 0 kbit/s (no continuous cost) to 3-4 kbit/s for extremely expressive sounds in the MPEG-4 Structured Audio format.

1.5.1.2.13 Wavetable Synthesis object type

The wavetable synthesis object is supported only by the SASBF format and MIDI tools. It allows the use of simple „sampling synthesis“ in presentations where the quality and flexibility of the full synthesis toolset is not required.

1.5.1.2.14 General Midi object type

The General MIDI object is included only to provide interoperability with existing content. Normative sound quality and decoder behavior are not provided with the General MIDI object.

1.5.1.2.15 Algorithmic Synthesis and Audio FX object type

The Algorithmic Synthesis object provides SAOL-based synthesis capabilities for very low-bitrate terminals. It is also used to support the AudioBIFS **AudioFX** node in content where sound synthesis capability is not needed.

1.5.1.2.16 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.

1.5.1.2.17 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.

1.5.1.2.18 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.

1.5.1.2.19 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.

1.5.1.2.20 Error Resilient (ER) BSAC object type

The ER BSAC object is supported by the fine grain scalablility 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 payload formatting of MPEG-4 AAC 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.

1.5.1.2.21 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).

1.5.1.2.22 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.

1.5.1.2.23 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.

1.5.1.2.24 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.

1.5.1.2.25 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.

1.5.1.2.26 SSC Audio object type

The SSC (SinuSoidal Coding) object combines the SSC parametric coding tools: transients, sinusoids, noise and parametric stereo. Only 44.1kHz and coding of mono, dual mono and (parametric) stereo are supported.

1.5.1.2.27 Layer-1 Audio object type

The Layer-1 object is the counterpart of the audio coding scheme Layer-1 specified in ISO/IEC 11172-3 and 13818-3.

1.5.1.2.28 Layer-2 Audio object type

The Layer-2 object is the counterpart of the audio coding scheme Layer-2 specified in ISO/IEC 11172-3 and ISO/IEC 13818-3.

1.5.1.2.29 Layer-3 Audio object type

The Layer-3 object is very similar to the audio coding scheme Layer-3 specified in ISO/IEC 11172-3 and ISO/IEC 13818-3. However, the use of Layer 3 encoded data as defined in ISO/IEC 13818-3 is limited to the “Lower

Sampling Frequencies" case, i.e. the Layer 3 multi-channel syntax defined in ISO/IEC 13818-3 is not permitted in this scope. Furthermore, additional sampling rates have been specified.

1.5.2 Audio profiles and levels

1.5.2.1 Profiles

The following Audio Profiles have been defined:

1. The **Speech Audio Profile** provides a parametric speech coder, a CELP speech coder and a Text-To-Speech interface.
2. The **Synthetic Audio Profile** provides the capability to generate sound and speech at very low bitrates.
3. The **Scalable Audio Profile,** a superset of the Speech Audio Profile, is suitable for scalable coding of speech and music, for transmission methods such as Internet and Digital Broadcasting.
4. The **Main Audio Profile** is a superset of the scalable profile, the speech profile and the synthesis profile, containing tools for natural and synthetic audio.
5. The **High Quality Audio Profile** contains the CELP speech coder and the Low Complexity AAC coder including Long Term Prediction. Scalable coding coding can be performed by the AAC Scalable object type. Optionally, the new error resilient (ER) bitstream payload syntax may be used.
6. The **Low Delay Audio Profile** contains the HVXC and CELP speech coders (optionally using the ER bitstream payload syntax), the low-delay AAC coder and the Text-to-Speech interface TTSI.
7. The **Natural Audio Profile** contains all natural audio coding tools available in MPEG-4.
8. 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.
9. The **AAC Profile** contains the audio object type 2 (AAC-LC).
10. The **High Efficiency AAC Profile** contains the audio object types 5 (SBR) and 2 (AAC LC). The High Efficiency AAC Profile is a superset of the AAC Profile.

Table 1.3 — Audio Profiles definition

Object Type ID	Audio Object Type	Speech Audio Profile	Synthetic Audio Profile	Scalable Audio Profile	Main Audio Profile	High Quality Audio Profile	Low Delay Audio Profile	Natural Audio Profile	Mobile Audio Internetworking Profile	AAC Profile	High Efficiency AAC Profile
0	Null										
1	AAC main				X			X			
2	AAC LC			X	X	X		X		X	X
3	AAC SSR				X			X			
4	AAC LTP			X	X	X		X			
5	SBR										X
6	AAC Scalable			X	X	X		X			
7	TwinVQ			X	X			X			
8	CELP	X		X	X	X	X	X			
9	HVXC	X		X	X		X	X			
10	(reserved)										
11	(reserved)										
12	TTSI	X	X	X	X		X	X			
13	Main synthetic		X		X						
14	Wavetable synthesis		(Subset of Main Synthetic)		(Subset of Main Synthetic)						

15	General MIDI		(Subset of Main Synthetic)		(Subset of Main Synthetic)						
16	Algorithmic Synthesis and Audio FX		(Subset of Main Synthetic)		(Subset of Main Synthetic)						
17	ER AAC LC					X		X	X		
18	(reserved)										
19	ER AAC LTP					X		X			
20	ER AAC Scalable					X		X	X		
21	ER TwinVQ							X	X		
22	ER BSAC							X	X		
23	ER AAC LD						X	X	X		
24	ER CELP					X	X	X			
25	ER HVXC						X	X			
26	ER HILN							X			
27	ER Parametric							X			
28	SSC										
29	(reserved)										
30	(reserved)										
31	(escape)										
32	Layer-1										
33	Layer-2										
34	Layer-3										
35	DST										

In addition to the profile descriptions given above it is stated that AAC Scalable objects using wide-band CELP core layer (with or without ER bitstream payload syntax) are **not** part of any Audio Profile.

1.5.2.2 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 bitstream payloads 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.

If a profile level is specified by the maximum number of complexity units, then a flexible configuration of the decoder handling different types of objects is allowed under the constraint that both values for the total complexity for decoding and sampling rate conversion (if needed) do not exceed this limit.

The following table gives complexity estimates for the different object types. PCU values are given in MOPS per channel, RCU values in kWords per channel (with respect to AAC, channel refers to Main channel, e. g. the channel of a SCE, one channel of a CPE, or the channel of an independently switched CCE).

Table 1.4 — Complexity of Audio Object Types and SR conversion

Object Type	Parameters	PCU (MOPS)	RCU	Remark
AAC Main	fs = 48 kHz	5	5	1)
AAC LC	fs = 48 kHz	3	3	1)
AAC SSR	fs = 48 kHz	4	3	1)
AAC LTP	fs = 48 kHz	4	4	1)
SBR	fs = 24/48 kHz (in/out) (SBR tool)	3	2.5	1)
	fs = 24/48 kHz (in/out) (Low Power SBR tool)	2	1.5	1)
	fs = 48/48 kHz (in/out) (Down Sampled SBR tool)	4.5	2.5	1)
	fs = 48/48 kHz (in/out) (Low Power Down Sampled SBR tool)	3	1.5	1)
AAC Scalable	fs = 48 kHz	5	4	1), 2)
TwinVQ	fs = 24 kHz	2	3	1)
CELP	fs = 8 kHz	1	1	
CELP	fs = 16 kHz	2	1	
CELP	fs = 8/16 kHz (bandwidth scalable)	3	1	
HVXC	fs = 8 kHz	2	1	
TTSI		-	-	4)
General MIDI		4	1	
Wavetable Synthesis	fs = 22.05 kHz	depends on bitstream payloads (3)	depends on bitstream payloads (3)	
Main Synthetic		depends on bitstream	depends on bitstream	
Algorithmic Synthesis and AudioFX		depends on bitstream	depends on bitstream	
Sampling Rate Conversion	rf = 2, 3, 4, 6, 8, 12	2	0.5	7)
ER AAC LC	fs = 48 kHz	3	3	1)
ER AAC LTP	fs = 48 kHz	4	4	1)
ER AAC Scalable	fs = 48 kHz	5	4	1), 2)
ER TwinVQ	fs = 24 kHz	2	3	1)
ER BSAC	fs = 48 kHz (input buffer size=26000bits)	4	4	1)
	fs = 48 kHz (input buffer size=106000bits)	4	8	
ER AAC LD	fs = 48 kHz	3	2	1)
ER CELP	fs = 8 kHz	2	1	
	fs = 16 kHz	3	1	
ER HVXC	fs = 8 kHz	2	1	
ER HILN	fs = 16 kHz, ns=93	15	2	6)
	fs = 16 kHz, ns=47	8	2	
ER Parametric	fs = 8 kHz, ns=47	4	2	5),6)

Definitions:

- fs = sampling frequency
- rf = ratio of sampling rates

Notes:

1) PCU proportional to sampling frequency.
2) Includes core decoder.

3) See ISO/IEC 14496-4.
4) The complexity for speech synthesis is not taken into account.
5) Parametric coder in HILN mode, for HVXC mode see ER HVXC.
6) PCU depends on fs and ns, see below.
7) Sampling Rate Conversion is needed, if objects of different sampling rates are combined in a scene (see ISO/IEC 14496-1:2000, subclause 9.2.2.13.2.1). The specified values have to be added for each required conversion.

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*ns) * fs/16

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

PCU and RCU for AAC:

For AAC object types, PCU and RCU depend on sampling frequency and channel configuration in the following way:

PCUs

PCU = (fs / fs_ref) * PCU_ref * (2 * #CPE + #SCE + #LFE + #IndepCouplingCh + 0.3 * #DepCouplingCh)

fs: actual sampling frequency

fs_ref: reference sampling frequency (sampling freqency for the given PCU_ref)

PCU_ref: reference PCU given in Table 1.4

#SCE: number of SCEs

#CPE: ...

RCUs

#CPE < 2:

RCU = RCU_ref * [#SCE + 0.5 * #LFE + 0.5 * #IndepCouplingCh + 0.4 * #DepCouplingCh]
+ [RCU_ref + (RCU_ref – 1)] * #CPE

#CPE >= 2:

RCU = RCU_ref * [#SCE + 0.5 * #LFE + 0.5 * #IndepCouplingCh + 0.4 * #DepCouplingCh]
+ [RCU_ref + (RCU_ref – 1) * (2 * #CPE – 1)]

RCU_ref: reference RCU given in Table 1.4

#SCE: number of SCEs

#CPE: ...

1.5.2.3 Levels within the profiles

The notation used to specify the number of audio channels indicates the number of main audio channels. Based on the number of main audio channels (A), Table 1.5 indicates for object types derived from the AAC multichannel syntax the number of LFE channels (L), the number of independently switched coupling channels (I), and the number of dependently switched coupling channels (D) in the form A.L.I.D.

Table 1.5 — Maximum number of the individual AAC channel types depending on the specified number of main audio channels

	Number of Main Audio Channels		
AOT	1	2	5
1 (AAC main)	1.0.0.0	2.0.0.0	5.1.1.1
2 (AAC LC)	1.0.0.0	2.0.0.0	5.1.0.1
3 (AAC SSR)	1.0.0.0	2.0.0.0	5.1.0.0
4 (AAC LTP)	1.0.0.0	2.0.0.0	5.1.0.1
17 (ER AAC LC)	1.0.0.0	2.0.0.0	5.1.0.0
19 (ER AAC LTP)	1.0.0.0	2.0.0.0	5.1.0.0
23 (ER AAC LD)	1.0.0.0	2.0.0.0	5.1.0.0

Note: In the case of scalable coding schemes, only the first instantiation of each object type will be counted to determine the number of objects relevant to the level definition and complexity metric. For example, in a scalable coder consisting of a CELP core coder and two enhancement layers implemented by means of AAC scalable objects, one CELP object and one AAC scalable object and their associated complexity metrics are counted since there is almost no overhead associated with the second (and any further) GA enhancement layer.

- **Levels for Speech Audio Profile**

Two levels are defined by number of objects:

1. One speech object.
2. Up to 20 speech objects.

- **Levels for Synthetic Audio Profile**

Three levels are defined :

1. Synthetic Audio 1: All bitstream payload elements may be used with:
   - “Low processing” (exact numbers in ISO/IEC 14496-4:2000)
   - Only core sample rates may be used
   - No more than one TTSI object
2. Synthetic Audio 2: All bitstream payload elements may be used with:
   - ”Medium processing” (exact numbers in ISO/IEC 14496-4:2000).
   - Only core sample rates may be used.
   - no more than four TTSI objects.
3. Synthetic Audio 3: All bitstream payload elements may be used with:
   - ”High processing” (exact numbers in ISO/IEC 14496-4:2000).
   - no more than twelve TTSI objects.

- **Levels for Scalable Audio Profile**

Four levels are defined by configuration; complexity units define the fourth level:

1. Maximum 24 kHz of sampling rate, one mono object (all object Types).
2. Maximum 24 kHz of sampling rate, one stereo object or two mono objects (all object Types).
3. Maximum 48 kHz of sampling rate, one stereo object or two mono objects (all object Types).
4. Maximum 48 kHz of sampling rate, one object with 5 main channels or multiple objects with at maximum one integer factor sampling rate conversion for a maximum of two channels.
   Flexible configuration is allowed with PCU < 30 and RCU < 19.

For the audio object types 2 (AAC LC), and 4 (AAC LTP), only a frame length of 1024 samples is permitted with respect to level 1, 2, 3 and 4. For the audio object types 2 (AAC LC) and 4 (AAC LTP), mono or stereo mixdown elements are not permitted with respect to level 1, 2, 3 and 4. For the audio object type 6 (AAC Scalable) the following restrictions apply. The number of AAC layers shall not exceed 8 in any scalable configuration. If audio object type 8 (CELP) is used as core layer coder, the number of CELP layers shall not exceed 2. If audio object type 7 (TwinVQ) is used as base layer coder, only one mono TwinVQ layer is permitted.

- **Levels for Main Audio Profile**

Main Audio Profile contains all natural and synthetic object types. Levels are then defined as a combination of the two different types of levels from the two different metrics defined for natural tools (computation-based metrics) and synthetic tools (macro-oriented metrics).

For Object Types not belonging to the Synthetic Profile four levels are defined:

1. Natural Audio 1: PCU < 40, RCU < 20
2. Natural Audio 2: PCU < 80, RCU < 64
3. Natural Audio 3: PCU < 160, RCU < 128
4. Natural Audio 4: PCU < 320, RCU < 256

For Object Types belonging to the Synthetic Profile the same three Levels are defined as above, i.e. Synthetic Audio 1, Synthetic Audio 2 and Synthetic Audio 3.

Four Levels are then defined for Main Profile:

1. Natural Audio 1 + Synthetic Audio 1
2. Natural Audio 2 + Synthetic Audio 1
3. Natural Audio 3 + Synthetic Audio 2
4. Natural Audio 4 + Synthetic Audio 3

For the audio object types 1 (AAC main), 2 (AAC LC), 3 (AAC SSR), and 4 (AAC LTP), only a frame length of 1024 samples is permitted with respect to level 1, 2 , 3 and 4. For the audio object types 1 (AAC Main), 2 (AAC LC), 3 (AAC SSR) and 4 (AAC LTP), mono or stereo mixdown elements are not permitted with respect to level 1, 2 , 3 and 4. For the audio object type 6 (AAC Scalable) the following restrictions apply. The number of AAC layers shall not exceed 8 in any scalable configuration. If audio object type 8 (CELP) is used as core layer coder, the number of CELP layers shall not exceed 2. If audio object type 7 (TwinVQ) is used as base layer coder, only one mono TwinVQ layer is permitted.

- **Levels for the High Quality Audio Profile**

Table 1.6 — 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	48	25	12 [*3]	0 %	0
4	5	48	100	42 [*3]	0 %	0
5	2	22.05	5	8	20%	9
6	2	48	10	8	20%	9
7	5	48	25	12 [*3]	20%	22
8	5	48	100	42 [*3]	20%	22

*1: The value specifies the maximum redundancy based on the available audio object with the longest maximum frame length. The redundancy might be larger in the case of smaller frame lengths. However, the application of any class FEC is not permitted if 0 % is specified. The limit is valid independently for each audio object. Since this value does neither cover the EP header and its protection nor any CRC, another 5 % must always be added to this value in order to derive the necessary increase of the minimum decoder input buffer. This does imply, that not more than those 5 % might be spent for the EP header and its protection or any CRC.
*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.

For the audio object types 2 (AAC LC), 4 (AAC LTP), 17 (ER AAC LC), and 19 (ER AAC LTP) only a frame length of 1024 samples is permitted with respect to level 1, 2, 3, 4, 5, 6, 7 and 8. For the audio object types 2 (AAC LC) and 4 (AAC LTP), mono or stereo mixdown elements are not permitted with respect to level 1, 2, 3, 4, 5, 6, 7 and 8. For the audio object type 6 and 20 ((ER) AAC Scalable) the following restrictions apply. The number of AAC layers shall not exceed 8 in any scalable configuration. If audio object type 8 or 24 ((ER) CELP) is used as core layer coder, the number of CELP layers shall not exceed 2.

- **Levels for the Low Delay Audio Profile**

Table 1.7 — 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: The value specifies the maximum redundancy based on the available audio object with the longest maximum frame length. The redundancy might be larger in the case of smaller frame lengths. However, the application of any class FEC is not permitted if 0 % is specified. The limit is valid independently for each audio object. Since this value does neither cover the EP header and its protection nor any CRC, another 5 % must always be added to this value in order to derive the necessary increase of the minimum decoder input buffer. This does imply, that not more than those 5 % might be spent for the EP header and its protection or any CRC.
*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.

- **Levels for the Natural Audio Profile**

Table 1.8 — 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: The value specifies the maximum redundancy based on the available audio object with the longest maximum frame length. The redundancy might be larger in the case of smaller frame lengths. However, the application of any class FEC is not permitted if 0 % is specified. The limit is valid independently for each audio object.

Since this value does neither cover the EP header and its protection nor any CRC, another 5 % must always be added to this value in order to derive the necessary increase of the minimum decoder input buffer. This does imply, that not more than those 5 % might be spent for the EP header and its protection or any CRC.

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

No RCU limitations are specified for this profile.

For the audio object types 1 (AAC main), 2 (AAC LC), 3 (AAC SSR), 4 (AAC LTP), 17 (ER AAC LC), and 19 (ER AAC LTP) only a frame length of 1024 samples is permitted with respect to level 1, 2 , 3 and 4. For the audio object types 1 (AAC Main), 2 (AAC LC), 3 (AAC SSR) and 4 (AAC LTP), mono or stereo mixdown elements are not permitted with respect to level 1, 2 , 3 and 4. For the audio object type 6 and 20 ((ER)AAC Scalable) the following restrictions apply. The number of AAC layers shall not exceed 8 in any scalable configuration. If audio object type 8 or 24 ((ER) CELP) is used as core layer coder, the number of CELP layers shall not exceed 2. If audio object type 7 or 21 ((ER) TwinVQ) is used as base layer coder, only one mono TwinVQ layer is permitted.

- **Levels for the Mobile Audio Internetworking Profile**

Table 1.9 — 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	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	48	25	12 *4	-	20%	22

*1: The value specifies the maximum redundancy based on the available audio object with the longest maximum frame length. The redundancy might be larger in the case of smaller frame lengths. However, the application of any class FEC is not permitted if 0 % is specified. The limit is valid independently for each audio object. Since this value does neither cover the EP header and its protection nor any CRC, another 5 % must always be added to this value in order to derive the necessary increase of the minimum decoder input buffer. This does imply, that not more than those 5 % might be spent for the EP header and its protection or any CRC.

*2: The maximum RCU for one channnel 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 1.4).

*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.

For the audio object type 17 (ER AAC LC) only a frame length of 1024 samples is permitted with respect to level 1,2,3,4,5 and 6. For the audio object type 20 (ER AAC Scalable) the following restrictions apply. The number of AAC layers shall not exceed 8 in any scalable configuration. If audio object type 21 (ER TwinVQ) is used as base layer coder, only one mono TwinVQ layer is permitted.

- **Levels for the AAC Profile**

Table 1.10 — Levels for the AAC Profile

Level	Max. channels/ object	Max. sampling rate [kHz]	Max. PCU	Max. RCU
1	2	24	3	5
2	2	48	6	5
3	NA	NA	NA	NA
4	5	48	19	15
5	5	96	38	15

For the audio object type 2 (AAC LC), mono or stereo mixdown elements are not permitted.

The NA (Not Applicable) levels are introduced to emphasize the hierarchical structure of the AAC Profile and the High Efficiency AAC Profile. Hence, a decoder supporting the High Efficiency AAC Profile at a given level can decode an AAC Profile stream of the same or a lower level. The NA levels are not indicated in the audioProfileLevelIndication table (Table 1.12).

- **Levels for the High Efficiency AAC Profile**

Table 1.11 — Levels for the High Efficiency AAC Profile

Level	Max. channels/ object	Max. AAC sampling rate, SBR not present [kHz]	Max. AAC sampling rate, SBR present [kHz]	Max. SBR sampling rate [kHz] (in/out)	Max. PCU	Max. RCU	Max. PCU Low power SBR	Max. RCU Low power SBR
1	NA	NA	NA	NA	NA	NA	NA	NA
2	2	48	24	24/48	9	10	7	8
3	2	48	48	48/48 (Note 1)	15	10	12	8
4	5	48	24/48 (Note 2)	48/48 (Note 1)	25	28	20	23
5	5	96	48	48/96	49	28	39	23
Note 1: For level 3 and level 4 decoders, it is mandatory to operate the SBR tool in downsampled mode if the sampling rate of the AAC core is higher than 24kHz. Hence, if the SBR tool operates on a 48kHz AAC signal, the internal sampling rate of the SBR tool will be 96kHz, however, the output signal will be downsampled by the SBR tool to 48kHz. Note 2: For one or two channels the maximum AAC sampling rate, with SBR present, is 48kHz. For more than two channels the maximum AAC sampling rate, with SBR present, is 24kHz.								

For the audio object type 2 (AAC LC), mono or stereo mixdown elements are not permitted.

1.5.2.4 audioProfileLevelIndication

audioProfileLevelIndication is a data element of the InitialObjectDescriptor defined in ISO/IEC 14496-1. It indicates the profile and level required to process the content associated with this InitialObjectDesriptor as defined in Table 1.12.

Table 1.12 — audioProfileLevelIndication values

Value	Profile	Level
0x00	Reserved for ISO use	-
0x01	Main Audio Profile	L1
0x02	Main Audio Profile	L2
0x03	Main Audio Profile	L3
0x04	Main Audio Profile	L4
0x05	Scalable Audio Profile	L1
0x06	Scalable Audio Profile	L2
0x07	Scalable Audio Profile	L3
0x08	Scalable Audio Profile	L4
0x09	Speech Audio Profile	L1
0x0A	Speech Audio Profile	L2
0x0B	Synthetic Audio Profile	L1
0x0C	Synthetic Audio Profile	L2
0x0D	Synthetic Audio Profile	L3
0x0E	High Quality Audio Profile	L1
0x0F	High Quality Audio Profile	L2
0x10	High Quality Audio Profile	L3
0x11	High Quality Audio Profile	L4
0x12	High Quality Audio Profile	L5
0x13	High Quality Audio Profile	L6
0x14	High Quality Audio Profile	L7
0x15	High Quality Audio Profile	L8
0x16	Low Delay Audio Profile	L1
0x17	Low Delay Audio Profile	L2
0x18	Low Delay Audio Profile	L3
0x19	Low Delay Audio Profile	L4
0x1A	Low Delay Audio Profile	L5
0x1B	Low Delay Audio Profile	L6
0x1C	Low Delay Audio Profile	L7
0x1D	Low Delay Audio Profile	L8
0x1E	Natural Audio Profile	L1
0x1F	Natural Audio Profile	L2
0x20	Natural Audio Profile	L3
0x21	Natural Audio Profile	L4
0x22	Mobile Audio Internetworking Profile	L1
0x23	Mobile Audio Internetworking Profile	L2
0x24	Mobile Audio Internetworking Profile	L3
0x25	Mobile Audio Internetworking Profile	L4
0x26	Mobile Audio Internetworking Profile	L5
0x27	Mobile Audio Internetworking Profile	L6
0x28	AAC Profile	L1
0x29	AAC Profile	L2
0x2A	AAC Profile	L4
0x2B	AAC Profile	L5
0x2C	High Efficiency AAC Profile	L2
0x2D	High Efficiency AAC Profile	L3
0x2E	High Efficiency AAC Profile	L4
0x2F	High Efficiency AAC Profile	L5
0x30 - 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.

1.6 Interface to ISO/IEC 14496-1 (MPEG-4 Systems)

1.6.1 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.

1.6.2 Syntax

1.6.2.1 AudioSpecificConfig

AudioSpecificConfig() extends the abstract class DecoderSpecificInfo, as defined in ISO/IEC 14496-1, when DecoderConfigDescriptor.objectTypeIndication refers to streams complying with ISO/IEC 14496-3 in this case the existence of AudioSpecificConfig() is mandatory.

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;		
if (audioObjectType == 5) {		
extensionAudioObjectType = audioObjectType;		
sbrPresentFlag = 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;
default:
    /* reserved */
}
switch (audioObjectType) {
```

case 17:		
case 19:		
case 20:		
case 21:		
case 22:		
case 23:		
case 24:		
case 25:		
case 26:		
case 27:		
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**
}		
}		
}		
}		
}		

Table 1.14 — Syntax of GetAudioObjectType()

Syntax	No. of bits	Mnemonic
GetAudioObjectType()		
{		
audioObjectType;	**5**	**uimsbf**
if (audioObjectType == 31) {		
audioObjectType = 32 + **audioObjectTypeExt**;	**6**	**uimsbf**
}		
return audioObjectType;		
}		

The classes defined in subclauses 1.6.2.1.1 to 1.6.2.1.9 do not extend the BaseDescriptor class (see ISO/IEC 14496-1) and consequently their length shall be derived by difference from the length of AudioSpecificConfig().

1.6.2.1.1 HvxcSpecificConfig

Defined in ISO/IEC 14496-3 subpart 2.

1.6.2.1.2 CelpSpecificConfig

Defined in ISO/IEC 14496-3 subpart 3.

1.6.2.1.3 GASpecificConfig

Defined in ISO/IEC 14496-3 subpart 4.

1.6.2.1.4 StructuredAudioSpecificConfig

Defined in ISO/IEC 14496-3 subpart 5.

1.6.2.1.5 TTSSpecificConfig

Defined in ISO/IEC 14496-3 subpart 6.

1.6.2.1.6 ParametricSpecificConfig

Defined in ISO/IEC 14496-3 subpart 7.

1.6.2.1.7 ErrorProtectionSpecificConfig

Defined in subclause 1.8.2.1.

1.6.2.1.8 ErrorResilientCelpSpecificConfig

Defined in ISO/IEC 14496-3 subpart 3.

1.6.2.1.9 ErrorResilientHvxcSpecificConfig

Defined in ISO/IEC 14496-3 subpart 2.

1.6.2.1.10 MPEG_1_2_SpecificConfig

Defined in ISO/IEC 14496-3 subpart 9.

1.6.2.1.11 SSCSpecificConfig

Defined in ISO/IEC 14496-3 subpart 8.

1.6.2.2 Payloads

1.6.2.2.1 Overview

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.

Payloads that are not byte aligned are padded at the end for transport schemes which require byte alignment.

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

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
0	NULL		
1	AAC MAIN	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
2	AAC LC	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
3	AAC SSR	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
4	AAC LTP	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
5	SBR	ISO/IEC 14496-3 subpart 4	
6	AAC scalable	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.3
7	TwinVQ	ISO/IEC 14496-3 subpart 4	
8	CELP	ISO/IEC 14496-3 subpart 3	
9	HVXC	ISO/IEC 14496-3 subpart 2	
10	(reserved)		
11	(reserved)		
12	TTSI	ISO/IEC 14496-3 subpart 6	
13	Main synthetic	ISO/IEC 14496-3 subpart 5	
14	Wavetable synthesis	ISO/IEC 14496-3 subpart 5	
15	General MIDI	ISO/IEC 14496-3 subpart 5	
16	Algorithmic Synthesis and Audio FX	ISO/IEC 14496-3 subpart 5	
17	ER AAC LC	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
18	(reserved)		
19	ER AAC LTP	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
20	ER AAC scalable	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
21	ER Twin VQ	ISO/IEC 14496-3 subpart 4	
22	ER BSAC	ISO/IEC 14496-3 subpart 4	
23	ER AAC LD	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
24	ER CELP	ISO/IEC 14496-3 subpart 3	
25	ER HVXC	ISO/IEC 14496-3 subpart 2	
26	ER HILN	ISO/IEC 14496-3 subpart 7	
27	ER Parametric	ISO/IEC 14496-3 subpart 2 and 7	
28	SSC	ISO/IEC 14496-3 subpart 8	
29	(reserved)		
30	(reserved)		
31	(escape)		
32	Layer-1	ISO/IEC 14496-3 subpart 9	
33	Layer-2	ISO/IEC 14496-3 subpart 9	
34	Layer-3	ISO/IEC 14496-3 subpart 9	
35	DST	ISO/IEC 14496-3 subpart 10	

1.6.2.2.2 Mapping of audio payloads to access units and elementary streams

1.6.2.2.2.1 AAC Main, AAC LC, AAC SSR, AAC LTP

One top level payload (raw_data_block()) is mapped into one access unit. Subsequent access units form one elementary stream.

1.6.2.2.2.2 AAC scalable

One top level payload (aac_scalable_main_element(), ASME) is mapped into one access unit. Subsequent access units form one elementary stream.

One top level payload (aac_scalable_extension_element(), ASEE) is mapped into one access unit. Subsequent access units of the same enhancement layer form one elementary stream. This results in individual elementary streams for each layer.

The streams of subsequent layers depend on each other.

1.6.2.2.2.3 ER AAC LC, ER AAC SSR, ER AAC LTP, ER AAC scalable, ER AAC LD

The data elements of the according top level payload (er_raw_data_block(), aac_scalable_main_element(), aac_scalable_extension_element()) are subdivided into different categories depending on its error sensitivity and collected in instances of these categories (see subpart 4, subclause 4.5.2.4). Depending on the value of epConfig, there are several ways to map these instances to access units to form one or several elementary streams (see subclause 1.6.3.6). Subsequent elementary streams depend on each other.

Note: The bits of the data function byte_alignment() terminating the AAC top level payloads may be omitted if the data elements of the according top level payload is not directly maped into one access unit. Hence, it might be ommitted if the top level payload is split (e.g. in case of epConfig=1) or post-processed (e.g. in case of epConfig=3).

1.6.3 Semantics

1.6.3.1 AudioObjectType

A five bit field indicating the audio object type. This is the master switch which selects the actual bitstream payload syntax of the audio data. In general, different object type use a different bitstream payload syntax. The interpretation of this field is given in Table 1.1.

1.6.3.2 AudioObjectTypeExt

This data element extends the range of audio object types.

1.6.3.3 samplingFrequency

The sampling frequency used for this audio object. Either transmitted directly, or coded in the form of **samplingFrequencyIndex.**

1.6.3.4 samplingFrequencyIndex

A four bit field indicating the sampling rate used. If samplingFrequencyIndex equals 15 then the actual sampling rate is signaled directly by the value of **samplingFrequency**. In all other cases **samplingFrequency** is set to the value of the corresponding entry in Table 1.16.

Table 1.16 — Sampling Frequency Index

samplingFrequencyIndex	Value
0x0	96000
0x1	88200
0x2	64000
0x3	48000
0x4	44100
0x5	32000
0x6	24000
0x7	22050
0x8	16000
0x9	12000
0xa	11025
0xb	8000
0xc	7350
0xd	reserved
0xe	reserved
0xf	escape value

1.6.3.5 channelConfiguration

A four bit field indicating the audio output channel configuration:

Table 1.17 — Channel Configuration

value	number of channels	audio syntactic elements, listed in order received	channel to speaker mapping
0	-	-	defined in GASpecificConfig
1	1	single_channel_element()	center front speaker
2	2	channel_pair_element()	left, right front speakers
3	3	single_channel_element(), channel_pair_element()	center front speaker, left, right front speakers
4	4	single_channel_element(), channel_pair_element(), single_channel_element()	center front speaker, left, right center front speakers, rear surround speakers
5	5	single_channel_element(), channel_pair_element(), channel_pair_element()	center front speaker, left, right front speakers, left surround, right surround rear speakers
6	5+1	single_channel_element(), channel_pair_element(), channel_pair_element(), lfe _element()	center front speaker, left, right front speakers, left surround, right surround rear speakers, front low frequency effects speaker
7	7+1	single_channel_element(), channel_pair_element(), channel_pair_element(), channel_pair_element(), lfe_element()	center front speaker left, right center front speakers, left, right outside front speakers, left surround, right surround rear speakers, front low frequency effects speaker
8-15	-	-	reserved

1.6.3.6 epConfig

This data element signals what kind of error robust configuration is used.

Table 1.18 — epConfig

epConfig	Description
0	All instances of all error sensitivity categories belonging to one frame are stored within one access unit. There exists one elementary stream per scalability layer, or just one elementary stream in case of non-scalable configurations.
1	Each instance of each sensitivity category belonging to one frame is stored separately within a single access unit, i.e. there exist as many elementary streams as instances defined within a frame.
2	The error protection decoder has to be applied. The definition of EP classes is not normatively defined, but defined at application level. However, the restictions imposed on EP classes as defined in subclause 1.7 must be satisfied.
3	The error protection decoder has to be applied. The mapping between EP classes and ESC instances is signaled by the data element directMapping.

1.6.3.7 direct mapping

This data element identifies the mapping between error protection classes and error sensitivity category instances.

Table 1.19 — directMapping

directMapping	Description
0	Reserved
1	Each error protection class is treated as an instance of an error sensitivity category (one to one mapping), so that the error protection decoder output is equivalent to epConfig=1 data.

1.6.3.8 extensionSamplingFrequencyIndex

A four bit field indicating the output sampling frequency of the extension tool corresponding to the extensionAudioObjectType, according to Table 1.16

1.6.3.9 extensionSamplingFrequency

The output sampling frequency of the extension tool corresponding to the extensionAudioObjectType. Either transmitted directly, or coded in the form of extensionSamplingFrequencyIndex.

1.6.3.10 bits_to_decode

A helper function; returns the number of bits not yet decoded in the current AudioSpecificConfig(), if the length of this element has been signaled by a system/transport layer. If the length of this element is unknown, bits_to_decode() returns 0.

1.6.3.11 syncExtensionType

Syncword which marks the beginning of appended extension configuration data. This configuration data corresponds to an extension tool of which the coded data is embedded (in a backward compatible manner) in that of the underlying audioObjectType. If syncExtensionType is present, the configuration data of the extension tool is separated from that of the underlying audioObjectType, which allows for backward compatible signaling (see subclause 1.6.5). Decoders that do not support the extension tool can ignore the extension tool configuration data. Note that this backward compatible signaling can only be used in MPEG-4 based systems that convey the length of the AudioSpecificConfig().

1.6.3.12 sbrPresentFlag

A flag indicating the presence or absence of SBR data in case of extensionAudioObjectType==5 (i.e. explicit SBR signaling, see subclause 1.6.5). The value –1 indicates that the sbrPresentFlag was not conveyed in the

AudioSpecificConfig(). In this case, a High Efficiency AAC Profile decoder shall be able to detect the presence of SBR data in the Elementary Stream (i.e. implicit SBR signaling, see subclause 1.6.5).

1.6.3.13 extensionAudioObjectType

A five bit field indicating the extension audio object type. This object type corresponds to an extension tool, which is used to enhance the underlying audioObjectType.

1.6.4 Upstream

1.6.4.1 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.

1.6.4.2 Syntax

Table 1.20 — Syntax of AudioUpstreamPayload()

Syntax	No. of bits	Mnemonic
AudioUpstreamPayload()		
{		
upStreamType;	**4**	**uimsbf**
switch (upStreamType) {		
case 0: /* scalablity 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;		


```
    }
}
```

1.6.4.3 Definitions

upStreamType A 4-bit unsigned integer value representing the type of the up-stream as defined in the following Table 1.21

Table 1.21 — 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 is transmitted.

layOrSynEle A 6-bit unsigned integer value representing the number of the syntactic elements (in case of multi-channel setup) or the number of the layers (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 $\left(2^{numFrameExp}-1\right)$ 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.

1.6.4.4 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.

1.6.4.4.1 Decoding of scalability control

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

1.6.4.4.2 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 Annex 4.B.17.

1.6.4.4.3 Decoding of quality feedback

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

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

1.6.4.4.4 Decoding of bitrate control

avgBitrate is parsed**.**

1.6.5 Signaling of SBR

1.6.5.1 Generating and signaling AAC+SBR content

The SBR tool in combination with the AAC coder provides a significant increase of audio compression efficiency. At the same time it allows for compatibility with existing AAC-only decoders. However, the audio quality for decoders without the SBR tool will of course be significantly lower than for those supporting the SBR tool. Therefore, depending on the application, a content provider or content creator will want to choose between the two alternatives given below. In general, the SBR data is always embedded in the AAC stream in an AAC compatible way (in the extension_payload), and SBR is a pure post processing step in the decoder. Therefore, compatibility can be achieved. However, by means of different signaling the content creator can select between the full-quality mode and the backward compatibility mode as follows.

1.6.5.1.1 Ensuring full audio quality of AAC+SBR for the listener

To ensure that all listeners get the full audio quality of AAC+SBR, the stream generated shall only play on SBR capable decoders (decoders that support the HE AAC Profile, hereinafter referred to as HE AAC Profile decoders). This is achieved by indicating the HE AAC profile and using the explicit, hierarchical signalling (signaling 2.A. as described below). As a result, decoders without SBR support will not play such streams. With regard to AAC-only streams, an HE AAC Profile decoders will decode all AAC Profile streams of the appropriate level, as the HE AAC Profile is a superset of the AAC Profile.

1.6.5.1.2 Achieving backward compatibility with existing AAC-only decoders

The aim of this mode is to get all AAC-based decoders to play the stream, even if they don't support the SBR tool. Compatible streams can be created using the following two signaling methods:

a) indicating a profile containing AAC (e.g. the AAC Profile), except the HE AAC Profile, and using the explicit backward compatible signalling (2.B. as described below). This method is recommended for all MPEG-4 based systems in which the length of the AudioSpecificConfig() is known in the decoder. As this is not the case for LATM with audioMuxVersion==0 (see subclause 1.7), this method cannot be used for LATM with audioMuxVersion==0. In explicit backward compatible signaling, SBR-specific configuration data is added at the end of the AudioSpecificConfig(). Decoders that do not know about SBR will ignore these parts, while HE AAC Profile decoders will detect its presence and configure the decoder accordingly.

b) indicating a profile containing AAC (e.g. the AAC Profile, or an MPEG-2 AAC profile), except the HE AAC Profile, and using implicit signalling. In this mode, there is no explicit indication of the presence of SBR data. Instead, decoders check the presence while decoding the stream and use the SBR tool if SBR data is found. This is possible because SBR can be decoded without SBR-specific configuration data if a certain way of handling decoder output sample rate is obeyed, as described below for HE AAC Profile decoders.

Both methods lead to the result that the AAC part of an AAC+SBR streams will be decoded by AAC-only decoders. AAC+SBR decoders will detect the presence of SBR and decode the full quality AAC+SBR stream.

1.6.5.2 Implicit and explicit signaling of SBR

This subclause outlines the different signaling methods of SBR, and the decoder behavior for different types of signaling.

There are several ways to signal the presence of SBR data:

1. **implicit signaling:** If EXT_SBR_DATA or EXT_SBR_DATA_CRC extension_payload() elements are detected in the bitstream payload, this implicitly signals the presence of SBR data. The ability to detect and decode implicitly signaled SBR is mandatory for all High Efficiency AAC Profile (HE AAC Profile) decoders.

2. **explicit signaling:** The presence of SBR data is signaled explicitly by means of the SBR Audio Object Type in the AudioSpecificConfig(). When explicit signaling is used, implicit signaling shall not occur. Two different types of explicit signaling are available:

   2.A. **hierarchical signaling:** If the first audioObjectType (AOT) signaled is the SBR AOT, a second audio object type is signaled which indicates the underlying audio object type. This signaling method is not backward compatible.

   2.B. **backward compatible signaling:** The extensionAudioObjectType is signaled at the end of the AudioSpecificConfig(). This method shall only be used in systems that convey the length of the AudioSpecificConfig(). Hence, it shall not be used for LATM with audioMuxVersion==0.

Table 1.22 shows the decoder behavior depending on profile and audio object type indication when implicit or explicit signaling is used.

Table 1.22 — SBR Signaling and Corresponding Decoder Behavior

Bitstream payload characteristics				Decoder behavior (Note 4)	
Profile indication	**extension AudioObjectType**	**sbrPresent Flag**	**raw_data_block**	**AAC decoders not supporting HE AAC Profile**	**AAC decoders supporting HE AAC Profile**
Profiles with AAC support other than High Efficiency AAC Profile	!= SBR (signaling 1)	-1 (Note 1)	AAC	Play AAC	Play AAC
			AAC+SBR	Play AAC	Play at least AAC, should play AAC+SBR
	== SBR (signaling 2.B)	0 (Note 2)	AAC	Play AAC	Play AAC
		1 (Note 3)	AAC+SBR	Play AAC	Play at least AAC, should play AAC+SBR
High Efficiency AAC Profile	== SBR (signaling 2.A or 2.B)	1 (Note 3)	AAC+SBR	Unsupported Profile - Don't play	Play AAC+SBR
Note 1: Implicit signaling, check payload in order to determine output sampling frequency, or assume the presence of SBR data in the payload, giving an output sampling frequency of twice the sampling frequency indicated by samplingFrequency in the AudioSpecificConfig() (unless the down sampled SBR Tool is operated, or twice the sampling frequency indicated by samplingFrequency exceeds the maximum allowed output sampling frequency of the current level, in which case the output sampling frequency is the same as indicated by samplingFrequency). Note 2: Explicitly signals that there is no SBR data, hence no implicit signaling is present, and the output sampling frequency is given by samplingFrequency in the AudioSpecificConfig(). Note 3: Output sampling frequency is the extensionSamplingFrequency in AudioSpecificConfig(). Note 4: In all cases a decoder has to support the Profile and Level indicated in the bitstream payload in order to be able to decode and play the content of the bitstream payload.					

The upper part of Table 1.22 displays bitstream payload characteristics and decoder behavior if the profile indication is any profile with AAC, apart from the High Efficiency AAC Profile. The lower part displays bitstream payload characteristics and decoder behavior if the profile indication is the High Efficiency AAC Profile.

1.6.5.3 HE AAC profile decoder behavior in case of implicit signaling

If the presence of SBR data is backward compatible implicitly signaled (signaling 1 in the list above) the extensionAudioObjectType is not the SBR AOT, and the sbrPresentFlag is set to –1, indicating that implicit signaling may occur.

Since the HE AAC Profile decoder is a dual rate system, with the SBR Tool operating at twice the sample rate of the underlying AAC decoder, the output sample rate cannot be assumed to be that of the AAC decoder just because SBR is not explicitly signaled. The decoder shall determine the output sample rate by either of the following two methods:

- Check for the presence of SBR data in the bitstream payload prior to decoding. If no SBR data is found, the output sample rate is equal to that signaled as samplingFrequency in the AudioSpecificConfig(). If SBR data is found the output sample rate is twice that signaled as samplingFrequency in the AudioSpecificConfig

- Assume that the SBR data is available and decide the output sample rate to be twice that signaled in the AudioSpecificConfig(). If no SBR data is found once the decoding process has started, the SBR Tool can be used for upsampling only, as described in subclause 4.6.18.5.

The above only applies if twice the sample rate signaled in the AudioSpecificConfig() does not exceed the maximum output sample rate allowed for the current level. Hence, for a HE AAC Profile decoder of levels 2, 3, or 4, the output sample rate is equal to the sample rate signaled in the AudioSpecificConfig() if the latter exceeds 24kHz.

The down sampled SBR Tool shall be used when needed to ensure that the output sample rate does not exceed the maximum allowed sample rate of the present level of the High Efficiency AAC Profile decoder.

1.6.5.4 HE AAC profile decoder behavior in case of explicit signaling

If the presence of SBR data is explicitly signaled (signaling 2, in the list above) the presence of SBR data is backward compatible explicitly signaled (signaling 2.B) or non-backward explicitly signaled (signaling 2.A).

For the backward compatible explicit signaling (signaling 2.B) the extensionAudioObjectType signaled is the SBR AOT. For this backward compatible explicit signaling the sbrPresentFlag is transmitted and can be either zero or one. If the sbrPresentFlag is zero, this indicates that SBR data is not present, and hence the HE AAC Profile decoder does not have to check the extension_payload() for the presence of SBR data or make assumptions on the output sample rate in anticipation of SBR data. If the sbrPresentFlag is one, SBR data is present and the HE AAC Profile decoder shall operate the SBR Tool.

For the non-backward compatible explicit signaling of SBR (signaling 2.A) the extensionAudioObjectType signaled is the SBR AOT. For this hierarchical explicit signaling, the sbrPresentFlag is set to one if the extensionAudioObjectType is SBR. The sbrPresentFlag is not transmitted and hence it is not possible to explicitly signal the absence of implicit signaling. Hence, for the hierarchical explicit signaling, SBR data is always present and the HE AAC Profile decoder shall operate the SBR Tool.

The down sampled SBR Tool shall be operated if the output sample rate would otherwise exceed the maximum allowed output sample rate for the present level, or if the extensionSamplingFrequency is the same as the samplingFrequency.

1.7 MPEG-4 Audio transport stream

1.7.1 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 1.1 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 their AudioSpecificConfig() 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 of the LOAS and the LATM formats are described in subclauses 1.7.2 and 1.7.3, respectively.

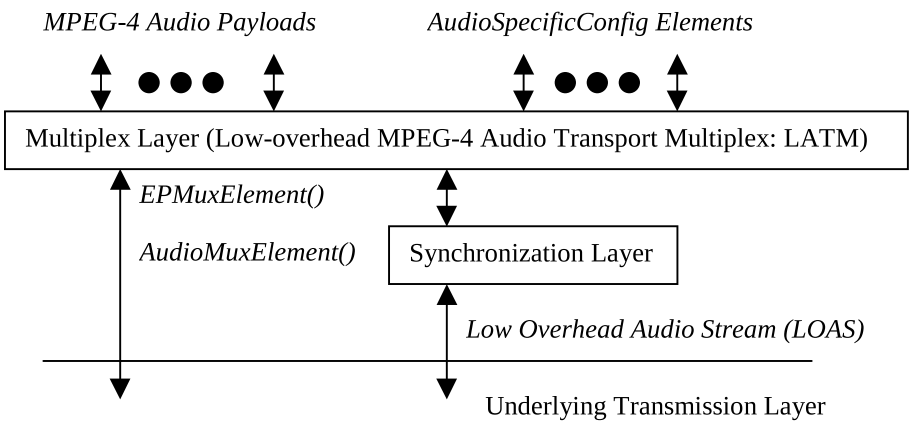


Figure 1.1 — Concept of MPEG-4 Audio Transport

The mechanism defined in this subclause should not be used for transmission of TTSI objects (12), Main Synthetic objects (13), Wavetable Synthesis objects (14), General MIDI objects (15) and Algorithmic Synthesis and Audio FX objects (16). It should further not be used for transmission of any object in conjunction with (epConfig==1). For those objects, other multiplex and transport mechanisms might be used, e.g. those defined in MPEG-4 Systems.

1.7.2 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 an underlying transmission layer with fixed frame synchronization, where transmission framing cannot be synchronized with the variable length multiplexed element. Figure 1.2 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.

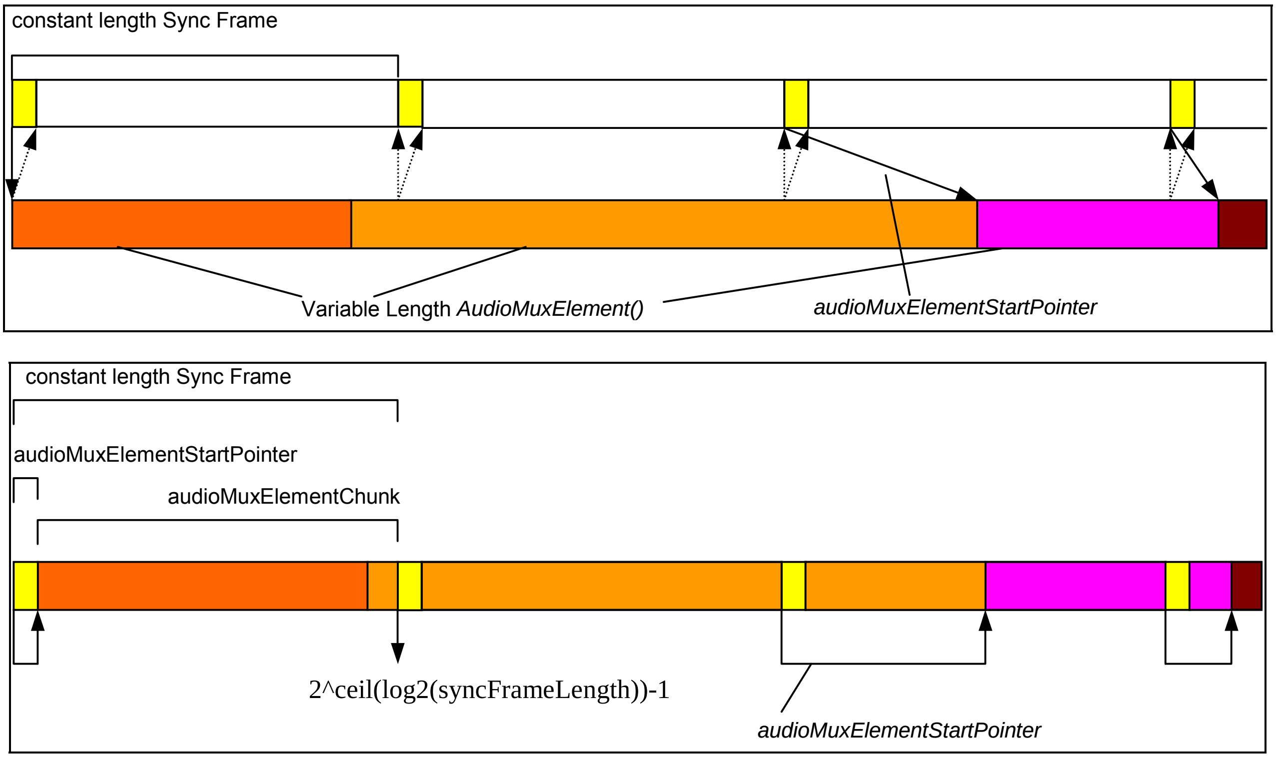


Figure 1.2 — Synchronization in AudioPointerStream()

1.7.2.1 Syntax

Table 1.23 — Syntax of AudioSyncStream()

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

Table 1.24 — Syntax of EPAudioSyncStream()

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

Table 1.25 — Syntax of AudioPointerStream()

Syntax	No. of bits	Mnemonic
AudioPointerStream (syncFrameLength)		
{		
while (! EndOfStream) {		
AudioPointerStreamFrame (syncFrameLength);		
}		
}		

Table 1.26 — Syntax of AudioPointerStreamFrame()

Syntax	No. of bits	Mnemonic
AudioPointerStreamFrame(length)		
{		
audioMuxElementStartPointer;	**ceil(log2(length))**	**uimsbf**
audioMuxElementChunk;	**length – ceil(log2(length))**	**bslbf**
}		

1.7.2.2 Semantics

1.7.2.2.1 AudioSyncStream()

audioMuxLengthBytes A 13-bit data element indicating the byte length of the subsequent AudioMuxElement() with byte alignment (AudioSyncStream) or the subsequent EPMuxElement() (EPAudioSyncStream).

AudioMuxElement() A multiplexed element as specified in subclause 1.7.3.2.2.

1.7.2.2.2 EPAudioSyncStream()

futureUse A 4-bit data element for future use, which shall be set to ‘0000’.

audioMuxLengthBytes see subclause 1.7.2.2.1.

frameCounter A 5-bit data element 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 data element 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 generator polynomial as described in subclause 1.8.4.3.

EPMuxElement() An error resilient multiplexed element as specified in subclause 1.7.3.2.1.

1.7.2.2.3 AudioPointerStream()

AudioPointerStreamFrame() A sync frame of fixed length provided by an underlying transmission layer.

audioMuxElementStartPointer A data element indicating the starting point of the first AudioMuxElement() within the current AudioPointerStreamFrame(). The number of bits required for this data element is calculated as ceil(log2(syncFrameLength)). The transmission frame length has to be provided from the underlying transmission layer. The maximum possible value of this data element is reserved to signal that there is no start of an AudioMuxElement() in this sync frame.

audioMuxElementChunk A part of a concatenation of subsequent AudioMuxElement()’s (see Figure 1.2).

1.7.3.1 Syntax

Table 1.27 — 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);		
}		
}		

Table 1.28 — Syntax of AudioMuxElement()

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

Table 1.29 – Syntax of StreamMuxConfig()

Syntax	No. of bits	Mnemonic
StreamMuxConfig()		
{		
audioMuxVersion;	**1**	**bslbf**
if (audioMuxVersion == 1) {		
audioMuxVersionA;	**1**	**bslbf**
}		

1.7.3 Multiplex Layer

The LATM layer multiplexes several MPEG-4 Audio payloads and AudioSpecificConfig() 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, 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.

Usage of LATM in case of scalable configurations with CELP core and AAC enhancement layer(s):

- *Instances of the AudioMuxElement() are transmitted in equidistant manner.*
- *The represented timeframe of one AudioMuxElement() is similar to a multiple of a super-frame timeframe.*
- *The relative number of bits for a certain layer within any AudioMuxElement() compared to the total number of bits within this AudioMuxElement() is equal to the relative bitrate of that layer compared to the bitrate of all layers.*
- *In case of coreFrameOffset = 0 and latmBufferFullness = 0, all core coder frames and all AAC frames of a certain super-frame are stored within the same instance of AudioMuxElement().*
- *In case of coreFrameOffset > 0, several or all core coder frames are stored within previous instances of AudioMuxElement().*
- *Any core layer related configuration information refers to the core frames transmitted within the current instance of the AudioMuxElement(), independent of the value of coreFrameOffset.*
- *A specified latmBufferFullness is related to the first AAC frame of the first super-frame stored within the current AudioMuxElement().*
- *The value of latmBufferFullness can be used to* **determine the location of the first bit of the first AAC frame of the current layer of the first super-frame stored within the current AudioMuxElement() by means of a backpointer***:*

  $$backPointer = -meanFrameLength + latmBufferFullness + currentFrameLength$$

  The backpointer value specifies the location as a negative offset from the current AudioMuxElement(), i. e. it points backwards to the beginning of an AAC frame located in already received data. Any data not belonging to the payload of the current AAC layer is not taken into account. If (latmBufferFullness == '0'), then the AAC frame starts after the current AudioMuxElement().

Note that the possible LATM configurations are restricted due to limited signalling capabilities of certain data elements as follows:

- Number of layers: 8 (numLayer has 3 bit)
- Number of streams: 16 (streamIndx has 4 bit)
- Number of chunks: 16 (numChunk has 4 bit)

```
else {
    audioMuxVersionA = 0;
}
if (audioMuxVersionA == 0) {
    if (audioMuxVersion == 1 ) {
        taraBufferFullness = LatmGetValue();
    }
    streamCnt = 0;
    allStreamsSameTimeFraming;                              1        uimsbf
    numSubFrames;                                           6        uimsbf
    numProgram;                                             4        uimsbf
    for (prog = 0; prog <= numProgram; prog++) {
        numLayer;                                           3        uimsbf
        for (lay = 0; lay <= numLayer; lay++) {
            progSIndx[streamCnt] = prog; laySIndx[streamCnt] = lay;
            streamID [ prog][ lay] = streamCnt++;
            if (prog == 0 && lay == 0) {
                useSameConfig = 0;
            } else {
                useSameConfig;                              1        uimsbf
            }
            if (! useSameConfig) {
                if ( audioMuxVersion == 0 ) {
                    AudioSpecificConfig();
                }
                else {
                    ascLen = LatmGetValue();
                    ascLen -= AudioSpecificConfig();                 Note 1
                    fillBits;                               ascLen   bslbf
                }
            }
            frameLengthType[streamID[prog][ lay]];          3        uimsbf
            if (frameLengthType[streamID[prog][lay]] == 0) {
                latmBufferFullness[streamID[prog][ lay]];   8        uimsbf
                if (! allStreamsSameTimeFraming) {
                    if ((AudioObjectType[lay] == 6 ||
                        AudioObjectType[lay] == 20) &&
                        (AudioObjectType[lay-1] == 8 ||
                        AudioObjectType[lay-1] == 24)) {
                            coreFrameOffset;                6        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) {
        if ( audioMuxVersion == 1 ) {
```

otherDataLenBits = LatmGetValue();		
}		
else {		
otherDataLenBits = 0; /* helper variable 32bit */		
do {		
otherDataLenBits *= 2^8;		
otherDataLenEsc;	**1**	**uimsbf**
otherDataLenTmp;	**8**	**uimsbf**
otherDataLenBits += otherDataLenTmp;		
} while (otherDataLenEsc);		
}		
}		
crcCheckPresent;	**1**	**uimsbf**
if (crcCheckPresent) **crcCheckSum;**	**8**	**uimsbf**
}		
else {		
/* tbd */		
}		
}		
Note 1: AudioSpecificConfig() returns the number of bits read.		

Table 1.30 — Syntax of LatmGetValue()

Syntax	No. of bits	Mnemonic
LatmGetValue()		
{		
bytesForValue;	**2**	**uimsbf**
value = 0; /* helper variable 32bit */		
for (i = 0; i <= bytesForValue; i++) {		
value *= 2^8;		
valueTmp;	**8**	**uimsbf**
value += valueTmp;		
}		
return value;		
}		

Table 1.31 — Syntax of PayloadLengthInfo()

Syntax	No. of bits	Mnemonic
PayloadLengthInfo()		
{		
if (allStreamsSameTimeFraming) {		
for (prog = 0; prog <= numProgram; prog++) {		
for (lay = 0; lay <= numLayer; lay++) {		
if (frameLengthType[streamID[prog][lay]] == 0) {		
MuxSlotLengthBytes[streamID[prog][lay]] = 0;		
do { /* always one complete access unit */		
tmp;	**8**	**uimsbf**
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]];	**2**	**uimsbf**
}		
}		
}		

```
        }
    } else {
        numChunk;                                                          4          uimsbf
        for (chunkCnt = 0; chunkCnt <= numChunk; chunkCnt++) {
            streamIndx;                                                    4          uimsbf
            prog = progCIndx[chunkCnt] = progSIndx[streamIndx];
             lay   = layCIndx[chunkCnt]   = laySIndx   [streamIndx];
            if (frameLengthType[streamID[prog][lay]] == 0) {
                MuxSlotLengthBytes[streamID[prog][ lay]] = 0;
                do {  /* not necessarily a complete access unit */
                    tmp;                                                   8          uimsbf
                    MuxSlotLengthBytes[streamID[prog][lay]] += tmp;
                } while (tmp == 255);
                AuEndFlag[streamID[prog][lay]];                            1          bslbf
            } else  {
                if (frameLengthType[streamID[prog][lay]] == 5 ||
                    frameLengthType[streamID[prog][lay]] == 7 ||
                    frameLengthType[streamID[prog][lay]] == 3) {
                    MuxSlotLengthCoded[streamID[prog][lay]];               2          uimsbf
                }
            }
        }
    }
}
```

Table 1.32 — Syntax of PayloadMux()

Syntax	No. of bits	Mnemonic
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 = progCIndx[chunkCnt];		
lay = layCIndx [chunkCnt];		
payload [streamID[prog][lay]];		
}		
}		
}		

1.7.3.2 Semantics

1.7.3.2.1 EPMuxElement()

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.

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.

epSpecificConfigLength A 10-bit data element to indicate the size of ErrorProtectionSpecificConfig()

epSpecificConfigLengthParity An 11-bit data element for epHeaderLength, calculated as described in subclause 1.8.4.3 with "1) Basic set of FEC codes".
Note: This means shortened Golay(23,12) is used

ErrorProtectionSpecificConfig() A data function covering configuration information for the EP tool which is applied to AudioMuxElement() as defined in subclause 1.8.2.1.

ErrorProtectionSpecificConfigParity() A data function covering the parity bits for **ErrorProtectionSpecificConfig()**, calculated as described in subclause 1.8.4.3, Table 1.46.

EPAudioMuxElement() A data function covering **e**rror 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 1.3 shows an example for the order of the instances assuming numSubFrames is one (1).

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

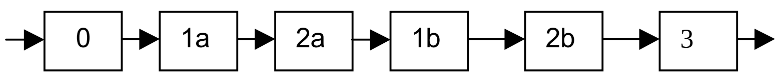


Figure 1.3 — Instance order in EPAudioMuxElement()

1.7.3.2.2 AudioMuxElement()

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.

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

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 data element indicating the other data information.

1.7.3.2.3 StreamMuxConfig()

AudioSpecificConfig() is specified in subclause 1.6.2.1. In this case it constitutes a standalone element in itself (i.e. it does not extend the class BaseDescriptor as in the case of subclause 1.6).

audioMuxVersion A data element to signal the used multiplex syntax.
Note: In addition to (audioMuxVersion == 0), (audioMuxVersion == 1) supports the transmission of a taraBufferFullness and the transmission of the lengths of individual AudioSpecificConfig() data functions.

audioMuxVersionA A data element to signal the used syntax version. Possible values: 0 (default), 1 (reserved for future extensions).

taraBufferFullness A helper variable indicating the state of the bit reservoir in the course of encoding the LATM status information. It is transmitted as the number of available bits in the tara bit reservoir divided by 32 and truncated to an integer value. The maximum value that can be signaled using any setting of bytesForValue signals that the particular program and layer is of variable rate. This might be the value of hexadecimal FF (bytesForValue == 0), FFFF (bytesForValue == 1), FFFFFF (bytesForValue == 2) or FFFFFFFF (bytesForValue == 3). In these cases, buffer fullness is not applicable. The state of the bit reservoir is derived according to what is stated in subpart 4, subclause 4.5.3.2 (Bit reservoir). The LATM status information considered by the taraBufferFullness comprises any data of the AudioMuxElement() except of PayloadMux().

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

numSubFrames A data element indicating how many PayloadMux() frames are multiplexed (numSubFrames+1). If more than one PayloadMux() frame is multiplexed, all PayloadMux() share a common StreamMuxConfig().The minimum value is 0 indicating 1 subframe.

numProgram A data element indicating how many programs are multiplexed (numProgram+1). The minimum value is 0 indicating 1 program.

numLayer A data element indicating how many scalable layers are multiplexed (numLayer+1). The minimum value is 0 indicating 1 layer.

useSameConfig A data element indicating that no AudioSpecificConfig() is transmitted but that the AudioSpecificConfig() most recently transmitted shall be applied.

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

ascLen[prog][lay] A helper variable indicating the length in bits of the subsequent AudioSpecificConfig() data function including possible fill bits.

fillBits Fill bits.

frameLengthType A data element 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 indices 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-indices, 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-indices, 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-indices, namely HVXCframeLengthTableIndex and MuxSlotLengthCoded.

latmBufferFullness[streamID[prog][lay]] data element indicating the state of the bit reservoir in the course of encoding the first access unit of a particular program and layer in an AudioMuxElement(). It is transmitted as the number of available bits in the bit reservoir divided by the NCC divided by 32 and truncated to an integer value. A value of hexadecimal FF signals that the particular program and layer is of variable rate. In this case, buffer fullness is not applicable. The state of the bit reservoir is derived according to what is stated in subpart 4, subclause 4.5.3.2 (Bit reservoir).
In the case of (audioMuxVersion == 0), bits spend for data other than any payload (e.g. multiplex status information or other data) are considered in the first occuring latmBufferFullness in an AudioMuxElement(). For AAC, the limitations given by the minimum decoder input buffer apply (see subpart 4, subclause 4.5.3.1). In the case of (allStreamsSameTimeFraming==1), and if only one program and one layer is present, this leads to an LATM configuration similar to ADTS.
In the case of (audioMUxVersion == 1), bits spend for data other than any payload are considered by taraBufferFullness.

coreFrameOffset identifies the first CELP frame of the current super-frame. It is defined only in case of scalable configurations with CELP core and AAC enhancement layer(s) and transmitted with the first AAC enhancement layer. The value 0 identifies the first CELP frame following StreamMuxConfig() as the first CELP frame of the current super-frame. A value > 0 signals the number of CELP frames that the first CELP frame of the current super-frame is transmitted earlier.

frameLength A data element indicating the frame length of the payload with frameLengthType of 1. The payload length in bits is specified as 8 * (frameLength + 20).

CELPframeLengthTableIndex A data element indicating one of two indices for pointing out the frame length for a CELP or ER_CELP object. (Table 1.34 and Table 1.35)

HVXCframeLengthTableIndex A data element indicating one of two indices for pointing out the frame length for a HVXC or ER_HVXC object. (Table 1.33)

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 helper variable indicating the length in bits of the other data.

crcCheckPresent A data element indicating the presence of CRC check bits for the StreamMuxConfig() data functions.

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

crcCheckSum CRC error detection data. This CRC uses the generation polynomial CRC8, as defined in subclause 1.8.4.5 and covers the entire StreamMuxConfig() upto but excluding the crcCheckPresent bit.

1.7.3.2.4 LatmGetValue()

bytesForValue A data element indicating the number of occurencies of the data element valueTmp.

valueTmp A data element used to calculate the helper variable value.

value A helper variable representing a value returned by the data function LatmGetValue().

1.7.3.2.5 PayloadLengthInfo()

tmp A data element 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 data element indicating one of two indices for pointing out the payload length for CELP, HVXC, ER_CELP, and ER_HVXC objects.

numChunk A data element indicating the number of payload chunks (numChunk+1). 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 0 indicating 1 chunk.

streamIndx A data element 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.

1.7.3.2.6 PayloadMux()

payload The actual audio payload by means of either an access unit (allStreamsSameTimeFraming == 1) or a part of a concatenation of subsequent access units (allStreamsSameTimeFraming == 0).

1.7.3.3 Tables

Table 1.33 — Frame length of HVXC [bits]

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

Table 1.34 — Frame Length of CELP Layer 0 [bits]

	Fixed-Rate frameLengthType[] =4	1-of-4 Rates (Silence Compression) frameLengthType[]=5				1-of-2 Rates (FRC) frameLengthType[]=3	
		MuxSlotLengthCoded				MuxSlotLengthCoded	
CELPframeLenghTable Index		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 1.35 — Frame Length of CELP Layer 1-5 [bits]

CELPframeLenghTableIndex	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				

1.8 Error protection

1.8.1 Overview of the tools

For error resilient audio object types, the error protection (EP) tool may be applied. The usage of this tool is signalled by the **epConfig** field. The input of the EP tool decoder consists of error protected access units. In case usage of the EP tool decoder is signalled by epConfig, the following restrictions apply:

- There exists one elementary stream per scalability layer, or just one elementary stream in case of non-scalable configurations.

- The output of the EP decoder is a set of several EP classes. The concatenation of EP classes at the output of the EP decoder is identical to epConfig = 0 data.

The definition of an EP class depends on **epConfig** and **directMapping**. For epConfig = 2, EP classes are not strictly defined. Their exact content is to be defined at application level, although the above mentioned restrictions have to be fulfilled. For epConfig = 3, the mapping between EP classes and instances of error sensitivity categories (ESCs) is normatively defined. In this case, the mapping is signalled by **directMapping**. In case directMapping = 1, each EP class maps exactly to one instance of an error sensitivity class. The EP decoder output then is identical to the case in which epConfig = 1. Figure 1.4 summarises the usage of EP classes, depending on the value of epConfig.

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

- 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 streams and variable-length frame streams

- providing a UEP configuration control with low overhead

The basic idea of UEP is to divide the frame into sub-frames according to the bit error sensitivities (these sub-frames are referred to be as classes in the following subclauses), and to protect these sub-frames with appropriate strength of FEC and/or CRC. If this would not be done, the decoded audio quality is determined by how the most error sensitive part is corrupted, and thus the strongest FEC/CRC has to be applied to the whole frame, requiring much more redundancy.

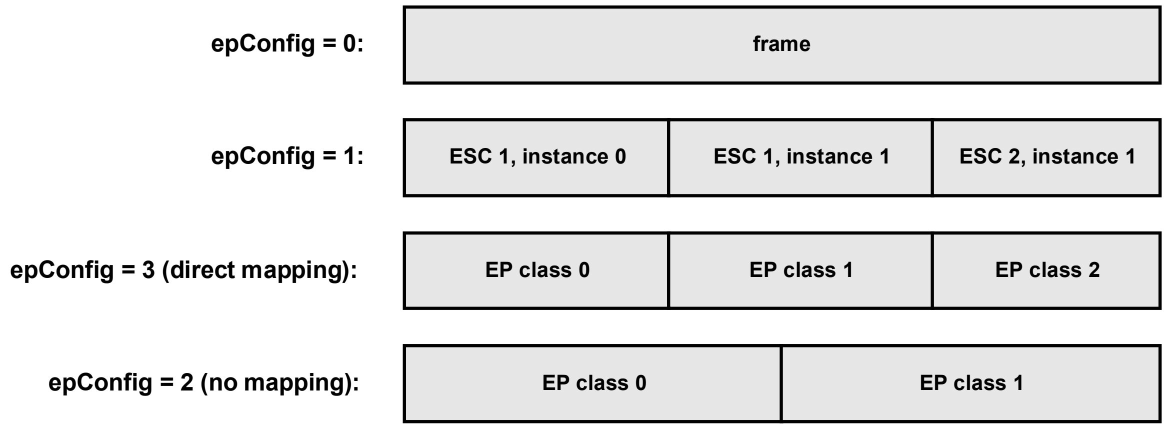


Figure 1.4 — EP classes for different epConfig values

In order to apply UEP to audio frames, the following information is required:

1) Number of classes

2) Number of bits each class contains

3) The CRC code to be applied for each class, which can be presented as a number of CRC bits

4) The FEC code to be applied for each class

This information is called as "frame configuration parameters" in the following sections. The same information is used to decode the UEP encoded frames; thus they have to be transmitted. To transmit them effectively, the frame structures of MPEG-4 audio algorithms have been taken into account for this EP tool.

The MPEG-4 audio frame structure can be categorized into three different approaches from the viewpoint of UEP application:

1) All the frame configurations are constant while the transmission (as CELP).

2) The frame configurations are restricted to be one of the several patterns (as Twin-VQ).

3) Most of the parameters are constant during the transmission, but some parameters can be different frame by frame (as AAC).

To utilize these characteristics, the EP tool uses two paths to transmit the frame configuration parameters. One is the out-of-band signaling, which is the same way as the transmission of codec configuration parameters. The parameters that are shared by the frames are transmitted through this path. In case there are several patterns of configuration, all these patterns are transmitted with indices. The other is the in-band transmission, which is made by defining the EP-frame structure with a header. Only the parameters that are not transmitted out-of-band are transmitted through this path. With this parameter transmission technique, the amount of in-band information, which is a part of the redundancy caused by the EP tool, is minimized.

With these parameters, each class is FEC/CRC encoded and decoded. To enhance the performance of this error protection, an interleaving technique is adopted. The objective of interleaving is to randomize burst errors within the frames, and this is not desirable for the class that is not protected. This is because there are other error resilience tools whose objective is to localize the effect of the errors, and randomization of errors with interleaving would have a harmful influence on such part of bitstream payload.

The outline of the EP encoder and EP decoder is figured out in Figure 1.5 and Figure 1.6.

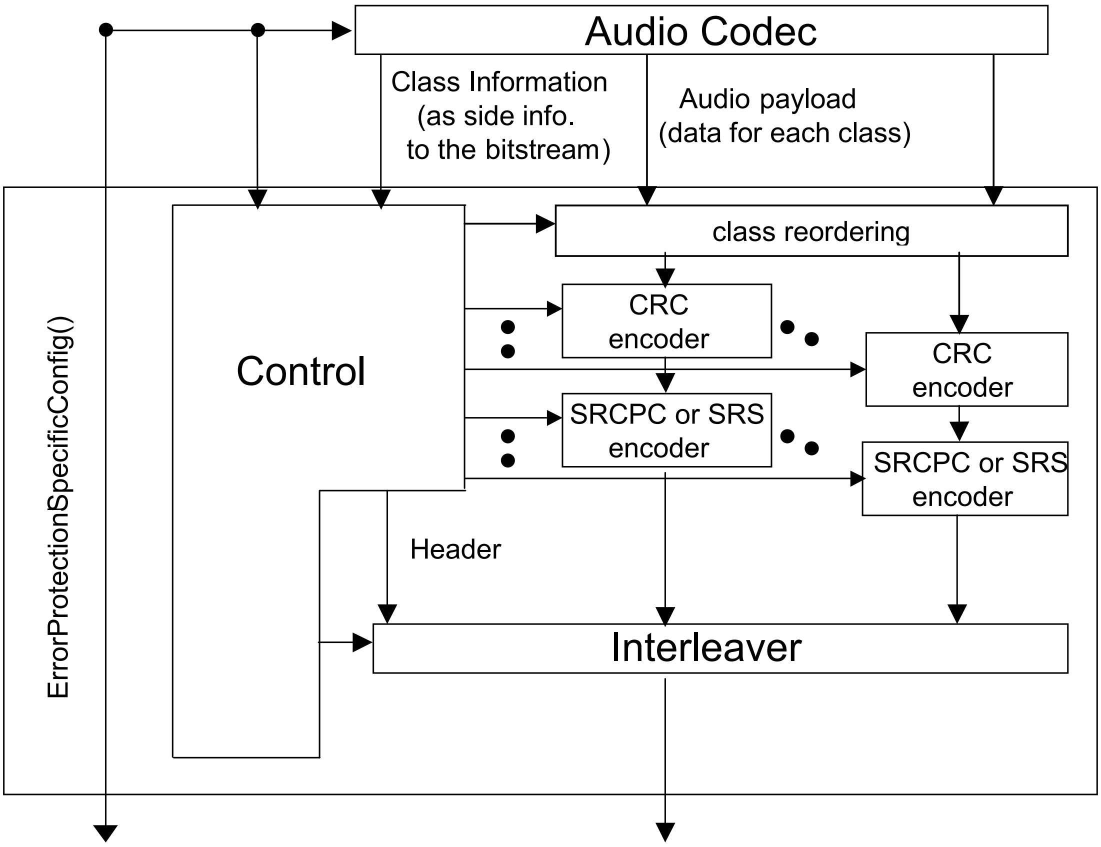


Figure 1.5 — Outline of EP encoder

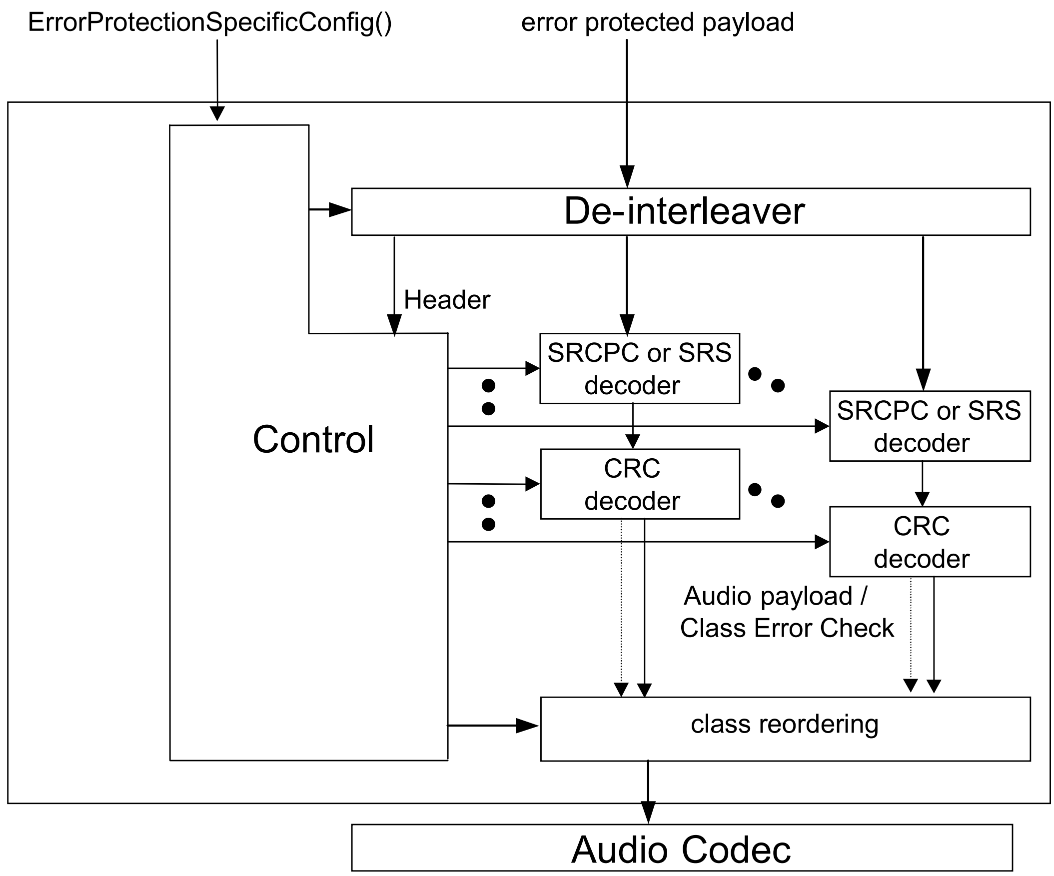


Figure 1.6 — Outline of EP decoder

1.8.2 Syntax

1.8.2.1 Error protection specific configuration

This part defines the syntax of the specific configuration for error protection.

Table 1.36 — Syntax of ErrorProtectionSpecificConfig()

Syntax	No. of bits	Mnemonic
ErrorProtectionSpecificConfig()		
{		
number_of_predefined_set;	**8**	**uimsbf**
interleave_type;	**2**	**uimsbf**
bit_stuffing;	**3**	**uimsbf**
number_of_concatenated_frame;	**3**	**uimsbf**
for (i = 0; i < number_of_predefined_set; i++) {		
number_of_class[i];	**6**	**uimsbf**
for (j = 0; j < number_of_class[i]; j++) {		
length_escape[i][j];	**1**	**uimsbf**
rate_escape[i][j];	**1**	**uimsbf**
crclen_escape[i][j];	**1**	**uimsbf**
if (number_of_concatenated_frame != 1) {		
concatenate_flag[i][j];	**1**	**uimsbf**

```
            }
            fec_type[i][j];                                   2        uimsbf
            if( fec_type[i][j] == 0) {
                termination_switch[i][j];                     1        uimsbf
            }
            if (interleave_type == 2) {
                interleave_switch[i][j];                      2        uimsbf
            }
            class_optional;                                   1        uimsbf
            if ( length_escape[i][j] == 1 ) {    /* ESC */
                number_of_bits_for_length[i][j];              4        uimsbf
            }
            else {
                class_length[i][j];                           16       uimsbf
            }
            if ( rate_escape[i][j] != 1 ) {   /* not ESC */
                if(fec_type[i][j]){
                    class_rate[i][j]                          7        uimsbf
                }else{
                    class_rate[i][j]                          5        uimsbf
                }
            }
            if ( crclen_escape[i][j] != 1 ) {     /* not ESC */
                class_crclen[i][j];                           5        uimsbf
            }
        }
        class_reordered_output;                               1        uimsbf
        if ( class_reordered_output == 1 ) {
            for ( j = 0; j < number_of_class[i]; j++ ) {
                class_output_order[i][j];                     6        uimsbf
            }
        }
    }
    header_protection;                                        1        uimsbf
    if ( header_protection == 1 ) {
        header_rate;                                          5        uimsbf
        header_crclen;                                        5        uimsbf
    }
}
```

1.8.2.2 Error protection bitstream payloads

This part defines the syntax of the error protected audio bitstream payload. This kind of syntax can be selected by setting epConfig=2 or epConfig=3. It is common for all audio object types. If MPEG-4 Systems is used, one ep_frame() is directly mapped to one access unit.

Table 1.37 — Syntax of ep_frame ()

Syntax	No. of bits	Mnemonic
ep_frame()		
{		
if (interleave_type == 0){		
ep_header();		
ep_encoded_classes();		
stuffing_bits;	**Nstuff**	**bslbf**
}		
if (interleave_type == 1){		
interleaved_frame_mode1;	**1 -**	**bslbf**
}		
if (interleave_type == 2){		
interleaved_frame_mode2;	**1 -**	**bslbf**
}		
}		

Nstuff: number of stuffing bits, identical to num_stuffing_bits, see subclause 1.8.3.1

Table 1.38 —Syntax of ep_header ()

Syntax	No. of bits	Mnemonic
ep_header()		
{		
choice_of_pred;	N_{pred}	**uimsbf**
choice_of_pred_parity;	N_{pred_parity}	**bslbf**
class_attrib();		
class_attrib_parity;	N_{attrib_parity}	**bslbf**
}		

Npred: ceil(log2 (number_of_predefined_set)).
Npred_parity: See subclause 1.8.4.3
Nattrib_parity: See subclause 1.8.4.3

Table 1.39 — Syntax of class_attrib ()

Syntax	No. of bits	Mnemonic
class_attrib()		
{		
for(j=0; j< number_of_class[choice_of_pred]; j++){		
if(class_reordered_output == 1){		
k = class_output_order[choice_of_pred][j];		
} else {		
k = j;		
}		
if (length_escape[choice_of_pred][k] == 1){		
class_bit_count[k];	**Nbitcount**	**uimsbf**
}		
if (rate_escape[choice_of_pred][k] == 1){		
class_code_rate[k];	**3**	**uimsbf**
}		
if (crclen_escape[choice_of_pred][k] == 1){		
class_crc_count[k];	**3**	**uimsbf**
}		
}		
if (bit_stuffing == 1){		
num_stuffing_bits;	**3**	**uimsbf**
}		
}		

Table 1.40 — Syntax of ep_encoded_classes()

Syntax	No. of bits	Mnemonic
ep_encoded_classes()		
{		
for(j=0; j< number_of_class[choice_of_pred]; j++){		
if(class_reordered_output == 1){		
k = class_output_order[choice_of_pred][j];		
} else {		
k = j;		
}		
ep_encoded_class[k];		**bslbf**
}		
}		

1.8.3 General information

1.8.3.1 Definitions

ErrorProtectionSpecificConfig (): Error protection specific configuration that is out-of-band information.

number_of_predefined_set The number of pre-defined set.

interleave_type This variable defines the interleave type. (interleave_type == 0) means no interleaving, (interleave_type == 1) means intra-frame interleaving and (interleave_type == 2) enables interleaving fine tuning for each class. For details see subclause 1.8.4.8. (interleave_type==3) is reserved.

bit_stuffing Signals whether the bit stuffing to ensure the byte alignment is used with the in-band information or not:
1 indicates the bit stuffing is used.
0 indicates the bit stuffing is not used. This implies that the configuration provided with the out-of-band information ensure the EP-frame is byte-aligned.

number_of_concatenated_frame The number of concatenated source coder frames for the constitution of one error protected frame.

Table 1.41 — concatenated frames depending on number_of_concatenated_frame

Codeword	000	001	010	011	100	101	110	111
number of concatenated frame	reserved	1	2	3	4	5	6	7

number_of_class[i] The number of classes for i-th pre-defined set.

length_escape[i][j] If 0, the length of j-th class in i-th pre-defined set is fixed value. If 1, the length is variable. Note that in case "until the end", this value should be 1, and the **number_of_bits_for_length[i][j]** value should be 0.

rate_escape[i][j] If 0, the SRCPC code rate of j-th class in i-th pre-defined set is fixed value. If 1, the code rate is signaled in-band.

crclen_escape[i][j] If 0, the CRC length of j-th class in i-th pre-defined set is fixed value. If 1, the CRC length is signaled in-band.

concatenate_flag[i][j] This parameter defines whether j-th class of i-th pre-defined set is concatenated or not. 0 indicates "not concatenated" and 1 indicates "concatenated". (See subclause 1.8.4.4)

fec_type[i][j] This parameter defines whether SRCPC code ("0") or RS code ("1" or "2") are used to protect the j-th class of i-th pre-defined set. Note that the class length which is signaled to be protected by RS code shall be byte aligned, in either case that the length is signaled in the out-of-band information or that the length is signaled as in-band information. If this field is set to "2", it indicates that this class is RS encoded in conjunction with next class as one RS code. Note that more than two succeeding classes have the value "2" for this field, it means these classes are concatenated and RS encoded as one RS code. If this field is "1", it indicates that this class is not concatenated with next class. This means this class is the last class to be concatenated before RS encoding, or this class is RS encoded independently.

termination_switch[i][j] This parameter defines whether j-th class of i-th pre-defined set is terminated or not when it is SRCPC encoded. See subclause 1.8.4.6.2.

interleave_switch[i][j] This parameter defines how to interleave j-th class of i-th pre-defined set.
0 – not interleaved
1 – interleaved without intraclass-interleaving: the interleaving width is same as the number of bits within the current class if (fec_type == 0), but same as the number of bytes within the current class if (fec_type==1 || fec_type == 2)
2 – interleaved with intraclass-interleaving: interleaving width is 28 if (fec_type == 0); but this value is reserved if (fec_type == 1 || fec_type == 2)
3 – concatenated
(see subclause 1.8.4.8.2.2)

class_optional This flag signals, whether the class is mandatory (class_optional == 0) or optional (class_optional == 1). This flag can be used to reduce the redundancy within ErrorProtectionSpecificConfig. Usually it would be necessary to define 2^N predefinition sets, where N equals the number of optional classes (see subclause 1.8.4.2).

number_of_bits_for_length[i][j] This field exists only when the **length_escape[i][j]** is 1. This value shows the number of bits for the class length in-band signaling. This value should be set considering possible maximum length of the class. The value 0 indicates the "until the end" functionality (see subclause 1.8.4.1).

class_length[i][j] This field exists only when the **length_escape[i][j]** is 0. This value shows the length of the j-th class in i-th pre-defined set, which is the fixed value while the transmission.

class_rate[i][j] This field exists only when the **rate_escape[i][j]** is 0. In case **fec_type[i][j]** is 0, this value shows the SRCPC code rate of the j-th class in i-th pre-defined set, which is the fixed value while the transmission. The value from 0 to 24 corresponds to the code rate from 8/8 to 8/32, respectively. In case **fec_type[i][j]** is 1 or 2, this value shows the number of erroneous bytes which can be corrected by RS code (see subclause 1.8.4.7). All the classes which is signaled to be concatenated with **fec_type[i][j]** shall have the same value of **class_rate[i][j]**.

class_crclen[i][j] This field exists only when the **crclen_escape[i][j]** is 0. This value shows the CRC length of the j-th class in i-th pre-defined set, which is the fixed value while the transmission. The value should be 0 – 18, which represents CRC length 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 24 or 32. (See subclause1.8.4.5)

class_reordered_output If this value is "1", the classes output from ep decoder is re-ordered. If "0", no such processing is made. See subclause 1.8.4.9.

class_output_order[i][j] This field only exists when class_reordered_output is set to "1", to signal the order of the class after re-ordering. The j-th class of i-th pre-defined set is output as (**class_output_order[i][j]**)-th class from ep decoder. See subclause 1.8.4.9.

header_protection This value indicates the header error protection mode. 0 indicates the use of basic set of FEC, and 1 indicates the use of extended header error protection, as defined in subclause 1.8.4.3. The extended header error protection is applied only if the length of the header exceeds 16 bits.

header_rate, header_crclen These values have the same semantics with **class_rate[i][j]** and **class_crclen[i][j]** respectively, while these error protection is utilized for the protection of header part.

ep_frame() error protected frame.

ep_header() EP frame header information.

ep_encoded_classes() The EP encoded audio information.

interleaved_frame_mode1 The information bits after interleaving with interleaving mode 1. See subclause 1.8.4.1 and subclause 1.8.4.8.

interleaved_frame_mode2 The information bits after interleaving with interleaving mode 2. See subclause 1.8.4.1 and subclause 1.8.4.8.

stuffing_bits The stuffing bits for the EP frame octet alignment. The number of bits Nstuff is signaled in class_attrib(), and should be in the range of 0...7.

choice_of_pred The choice of pre-defined set. See subclause 1.8.4.2.

choice_of_pred_parity The parity bits for **choice_of_pred**. See subclause 1.8.4.2.

class_attrib_parity The parity bits for class_attrib(). See subclause 1.8.4.2.

class_attrib() Attribution information for each class

class_bit_count[j] The number of information bits included in the class. This field only exists in case the length_escape in out-of-band information is 1 (escape). The number of bits of this parameter Nbitcount is also signaled in the out-of-band information.

class_code_rate[j] The coding rate for the audio data belonging to the class, as defined in the table below. This field only exists in case the rate_escape in out-of-band information is 1 (escape).

Table 1.42 — The coding rate for the audio data belonging to the class

Codeword	000	001	010	011	100	101	110	111
Puncture Rate	8/8	8/11	8/12	8/14	8/16	8/20	8/24	8/32
Puncture Pattern	FF, 00 00, 00	FF, A8 00, 00	FF, AA 00, 00	FF, EE 00, 00	FF, FF 00, 00	FF, FF AA, 00	FF, FF FF, 00	FF, FF FF, FF

class_crc_count[j] The number of CRC bits for the audio data belonging to the class, as defined in the table below. This field only exists in case the crclen_escape in out-of-band information is 1 (escape).

Table 1.43 — The number of CRC bits for the audio data belonging to the class

Codeword	000	001	010	011	100	101	110	111
CRC bits	0	6	8	10	12	14	16	32

num_stuffing_bits the number of stuffing bits for the EP frame octet alignment. This field only exists in case the **bit_stuffing** in out-of-band information is 1.

ep_encoded_class[j] CRC/SRCPC encoded audio data of j-th class.Note that if **class_bit_count[j]** == 0, audio data of j-th class is not encoded by CRC/SRCPC/SRS.

1.8.4 Tool description

1.8.4.1 Out-of-band information

The content of out-of band information is represented by means of ErrorProtectionSpecificConfig(). Some configuration examples are provided in Annex 1.B.

The length of the last class processed by the EP decoder (prior to any subsequent re-ordering as described in subclause 1.8.4.9) does not have to be transmitted explicitly, but its length might be signaled as “until the end”. In MPEG-4 Systems, the systems layer guarantees the audio frame boundary by mapping one audio frame to one access unit. Therefore, the length of the “until the end” class can be calculated from the length of other classes and the total EP-encoded audio frame length.

The flag class_optional might be used to reduce the redundancy within ErrorProtectionSpecificConfig(). However, the EP tool still works with the same number of pre-defined sets. If there are N classes with (class_optional == 1), this pre-defined set is extended to 2^N pre-defined sets. Unwrapping of the predefinition sets is described within the following subclause.

1.8.4.2 Derivation of pre-defined sets

This subclause describes the post processing, whose input is ErrorProtectionSpecificConfig() with “class_optional” switch and whose output are pre-defined sets used for the ep_frame() parameters.

General procedure:

- Each pre-defined set expands $2^{NCO[i]}$ pre-defined sets, where NCO[i] is the number of classes with (class_optional == 1) in the i-th original pre-defined set. Hereafter, any class with (class_optional == 1) is referred to as optClass.
- These expanded pre-defined sets start from “all the optClasses exist” to “all the optClasses do not exist”.

Algorithm:

```
transPred = 0;
for ( i = 0; i < nPred; i++ ) {          /* for all predefinition sets */
   for ( j = 0; j < 2^NCO[i]; j++ ) {       /* unwraping */
      for ( k = 0; k < NCO[i]; k++ ) {      /* for all optional classes */
         if ( j & ( 0x01 << k ) ) {
            optClassExists[k] = 0;
         }
         else {
            optClassExists[k] = 1;
         }
      }
      DefineTransPred(transPred, i, optClassExists);
      transPred ++;
   }
}
```

where,

optClassExists[k] signals whether k-th optClass of the pre-defined set exists (1) or not (0) in the defining new pre-defined set.

DefineTransPred (transPred, i, optClassExists)   defines transPred-th new pre-defined set used for the transmission. This new pre-defined set is a copy of the i-th original pre-defined set, except it does not have optClasses whose optClassExists equals to 0.

Example

ErrorProtectionSpecificConfig() defines pre-defined sets as follows:

Table 1.44 — The example of pre-defined set

Pred #0		Pred #1	
Class A	class_optional = 1	Class F	class_optional = 1
Class B	class_optional = 0	Class G	class_optional = 0
Class C	class_optional = 1		
Class D	class_optional = 0		
Class E	class_optional = 1		

After the pre-processing described above, the pre-defined sets used for ep_frame() becomes as follows:

Table 1.45 — The example of pre-defined sets after the pre-processing

Pred #0	Pred #1	Pred #2	Pred #3	Pred #4	Pred #5	Pred #6	Pred #7	Pred #8	Pred #9
Class A	Class B	Class A	Class B	Class A	Class B	Class A	Class B	Class F	Class G
Class B	Class C	Class B	Class D	Class B	Class C	Class B	Class D	Class G	
Class C	Class D	Class D	Class E	Class C	Class D	Class D			
Class D	Class E	Class E		Class D					
Class E									

1.8.4.3 In-band information

The EP frame information, which is not included in the out-of-band information, is the in-band information. The parameters belonging to this information are transmitted as an EP frame header. The parameters are:

- The choice of pre-defined set
- The number of stuffing bits for byte alignment
- The class information which is not included in the out-of-band information

The EP decoder cannot decode the audio frame information without these parameters, and thus they have to be error protected stronger than or equal to the other parts. On this error protection, the choice of pre-defined set has to be treated differently from the other parts. This is because the length of the class information can be changed according to which pre-defined set is chosen. For this reason, this parameter is FEC encoded independently from the other parts. At decoder side, the choice of pre-defined set is decoded first, and then the length of the remaining header part is calculated with this information, and decodes that.

The FEC applied for these parts are as follows:

Basic set of FEC codes:

Table 1.46 — Basic set of FEC codes for in-band information

number of bits to be protected	FEC code	total number of bits	Length of codeword
1-2	majority (repeat 3 times)	3-6	3
3-4	BCH(7,4)	6-7	6-7
5-7	BCH(15,7)	13-15	13-15
8-12	Golay(23,12)	19-23	19-23
13-16	BCH(31,16)	28-31	28-31
17-	RCPC 8/16 + 4-bit CRC	50 -	-

Notes:

- total number of bits: number of bits after FEC encoding
- interleaving width: width of the interleaving matrix, see also subclause 1.8.4.8
- Npred_parity (or Nattrib_parity) = total number of bits – number of bits to be protected
- SRCPC is terminated
- number of bits to be protected is Npred (or the total number of bits for class_attrib())

Extended FEC:

If a header length exceeds 16 bits, this header is protected using a CRC and a terminated SRCPC. The SRCPC code rate and the number of CRC bits are signaled. The encoding and decoding method for this is the same as described below within the CRC/SRCPC description.

The generator polynomials for each FEC are as follows:

BCH(7,4): x^3+x+1

BCH(15,7): $x^8+x^7+x^6+x^4+1$

Golay(23,12): $x^{11}+x^9+x^7+x^6+x^5+x+1$

BCH(31,16): $x^{15}+x^{11}+x^{10}+x^9+x^8+x^7+x^5+x^3+x^2+x+1$

With these polynomials, the FEC (n, k) for l-bit information encoding is made as follows:

Calculate the polynomial R(x) that satisfies

$M(x)\, x^{n-l} = Q(x)G(x) + R(x)$

M(x): Information bits. Highest order corresponds to the first bit to be transmitted

G(x): The generation polynomial from the above definition

This polynomial R(x) represents parity to choice_of_pred or class_attrib(), and set to choice_of_pred_parity or class_attrib_parity respectively. The highest order corresponds to the first bit. The decoder can perform error correction using these parity bits , while it is optional operation.

1.8.4.4 Concatenation functionality

EP tool has a functionality to concatenate several source coder frames to build up a new frame for the EP tool. In this concatenation, the groups of bits belonging to the same class in the different source coder frames are concatenated in adjacent, class by class basis. The concatenated groups belonging to the same class is either treated as a single new one class or independent class in the same manner as before the concatenation.

The number of frames to be concatenated is signaled as number_of_concatenated_frame in ErrorProtectionSpecificConfig(), and the choice whether the concatenated groups belonging to the same class is treated as single new one class or independent class is signaled by concatenate_flag[i][j] (1 indicate “single new one class”, and 0 indicates “independent class”). This process is illustrated in Figure 1.7.

The same pre-defined set shall be used for all concatenated frames. No escape mechanism shall be used for any class parameter.

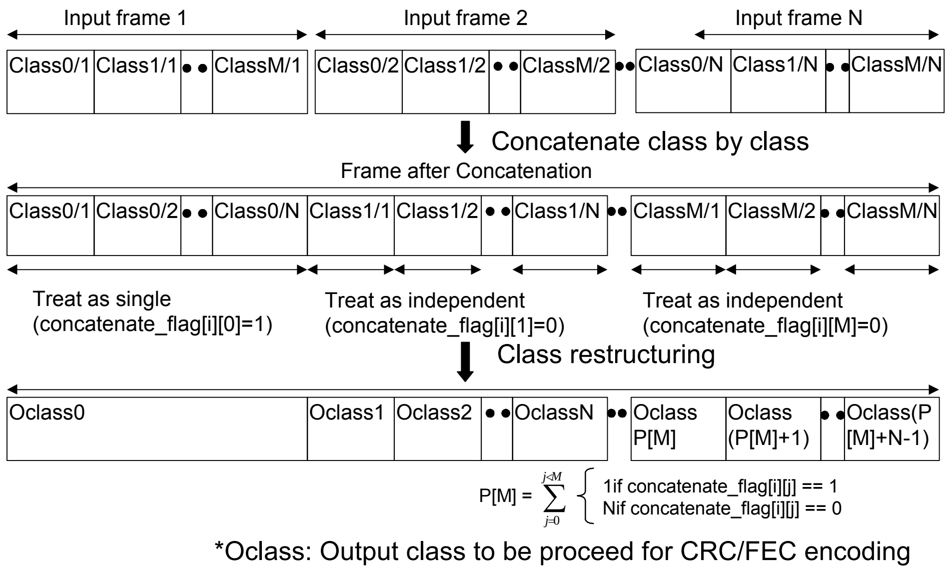


Figure 1.7 — Concatenation procedure

1.8.4.5 CRC

The CRC provides error detection capability. The information bits of each class is CRC encoded as a first process. In this tool, the following set of the CRC is defined:

1-bit CRC ***CRC1***: $x+1$

2-bit CRC ***CRC2***: x^2+x+1

3-bit CRC ***CRC3***: x^3+x+1

4-bit CRC ***CRC4***: $x^4+x^3+x^2+1$

5-bit CRC ***CRC5***: $x^5+x^4+x^2+1$

6-bit CRC ***CRC6*** : $x^6+x^5+x^3+x^2+x+1$

7-bit CRC ***CRC7*** : $x^7+x^6+x^2+1$

8-bit CRC ***CRC8*** : x^8+x^2+x+1

9-bit CRC ***CRC9*** : $x^9+x^8+x^5+x^2+x+1$

10-bit CRC ***CRC10*** : $x^{10}+x^9+x^5+x^4+x+1$

11-bit CRC ***CRC11*** : $x^{11}+x^{10}+x^4+x^3+x+1$

12-bit CRC ***CRC12*** : $x^{12}+x^{11}+x^3+x^2+x+1$

13-bit CRC ***CRC13*** : $x^{13}+x^{12}+x^7+x^6+x^5+x^4+x^2+1$

14-bit CRC ***CRC14*** : $x^{14}+x^{13}+x^5+x^3+x^2+1$

15-bit CRC ***CRC15*** : $x^{15}+x^{14}+x^{11}+x^{10}+x^7+x^6+x^2+1$

16-bit CRC ***CRC16*** : $x^{16}+x^{12}+x^5+1$

24-bit CRC ***CRC24*** : $x^{24}+x^{23}+x^6+x^5+x+1$

32-bit CRC ***CRC32*** : $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

With these polynomials, the CRC encoding is made as follows:

Calculate the polynomial R(x) that satisfies

$M(x)x^k = Q(x)G(x) + R(x)$

M(x): Information bits. Highest order corresponds to the first bit to be transmitted

G(x): The generation polynomial from the above definition

k: The number of CRC bits.

With this polynomial *R(x),* the CRC encoded bits *W(x)* is represented as:

$W(x) = M(x)x^k + R(x)$

Note that the value k should be chosen so that the number of CRC encoded bits does not exceed 2k-1.

The CRC bits are written in a reversed manner, i. e. each bit is inverted. Using these CRC bits, the decoder can perform error detection. When an error is detected through CRC, error concealment may be applied to reduce the quality degradation caused by the error. The error concealment method depends on MPEG-4 audio algorithms, an example is given in subclause 1.B.3.

1.8.4.6 Systematic rate-compatible punctured convolutional (SRCPC) codes

Following to the CRC encoding, FEC encoding is made with the SRCPC codes. This subclause describes the SRCPC encoding process.

The channel encoder is based on a systematic recursive convolutional (SRC) encoder with rate R=1/4. The CRC encoded classes are concatenated and input into this encoder. Then, with the puncturing procedure described in the subclause later, we obtain a Rate Compatible Punctured Convolutional (RCPC) code whose code rate varies for each class according to the error sensitivity.

1.8.4.6.1 SRC code generation

The SRC code is generated from a rational generator matrix by using a feedback loop. A shift register realization of the encoder is shown in Figure 1.8.

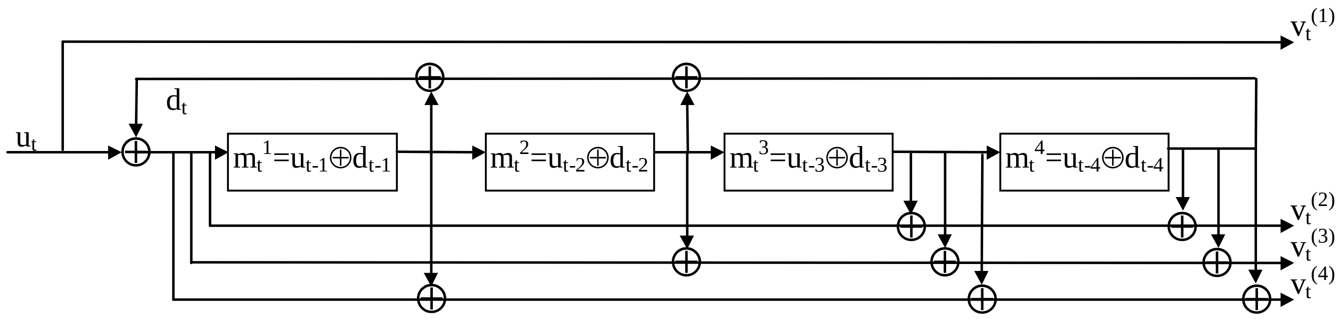


Figure 1.8 — Shift register realization for systematic recursive convolutional encoder

To obtain the output vectors v_t at each time instant t, one has to know the content of the shift registers m_t^1, m_t^2, m_t^3, m_t^4 (corresponds to the state) and the input bit u_t at time t.

We obtain the output $v_t^{(2)}$, $v_t^{(3)}$ and $v_t^{(4)}$

$$v_t^{(2)} = m_t^4 \oplus m_t^3 \oplus (u_t \oplus d_t)$$

$$v_t^{(3)} = m_t^4 \oplus m_t^3 \oplus m_t^2 \oplus (u_t \oplus d_t)$$

$$v_t^{(4)} = m_t^4 \oplus m_t^3 \oplus m_t^1 \oplus (u_t \oplus d_t)$$

with

$$d_t = m_t^4 \oplus m_t^2 \oplus m_t^1,\ m_t^4 = u_{t-4} \oplus d_{t-4},\ m_t^3 = u_{t-3} \oplus d_{t-3},\ m_t^2 = u_{t-2} \oplus d_{t-2},\ m_t^1 = u_{t-1} \oplus d_{t-1}$$

Finally we obtain for the output vector $\underline{\mathbf{v}}_t = (v_t^{(1)}, v_t^{(2)}, v_t^{(3)}, v_t^{(4)})$ at time t depending on the input bit u_t and the current state $\underline{\mathbf{m}}_t = (m_t^1, m_t^2, m_t^3, m_t^4)$:

$$\boxed{\begin{aligned}
V_t^{(1)} &= u_t \\
V_t^{(2)} &= m_t^4 \oplus m_t^3 \oplus (u_t \oplus d_t) = m_t^3 \oplus m_t^2 \oplus m_t^1 \oplus u_t \\
V_t^{(3)} &= m_t^4 \oplus m_t^3 \oplus m_t^2 \oplus (u_t \oplus d_t) = m_t^3 \oplus m_t^1 \oplus u_t \\
V_t^{(4)} &= m_t^4 \oplus m_t^3 \oplus m_t^1 \oplus (u_t \oplus d_t) = m_t^3 \oplus m_t^2 \oplus u_t
\end{aligned}}$$

with $\underline{\mathbf{m}}_1 = (m_1^1, m_1^2, m_1^3, m_1^4) = (0, 0, 0, 0) = \underline{\mathbf{0}}$

The initial state is always $\underline{\mathbf{0}}$, i.e. each memory cell contains a 0 before the input of the first information bit u_t.

1.8.4.6.2 Termination of SRC code

In case the SRC coded class is indicated as terminated with termination_switch[i] in ErrorProtectionSpecificConfig(), or SRC code is used for the protection of in-band information, the SRC encoder shall add the tail bits at the end of this class, and start the succeeding SRC encoding with initial state (all the encoder shift register shall reset to be 0).

The tail bits following the information sequence $\boldsymbol{u}$ for returning to state $\underline{\mathbf{m}}_n = \underline{\mathbf{0}}$ (termination) depends on the last state $\underline{\mathbf{m}}_{n-3}$ (state after the input of the last information bit u_{n-4}.). The termination sequence for each state described by $\underline{\mathbf{m}}_{n-3}$ is given in Table 1.47. The receiver may use these tail bits (TB) for additional error detection.

The appendix $(u_{n-3}, u_{n-2}, u_{n-1}\ u_n)$ to the information sequence can be calculated with the following condition:

$$\text{for all } t \text{ with } n-3 \le t \le n:\ u_t \oplus d_t = 0$$

Hence we obtain for the tail bit vector $\underline{\mathbf{u}}'=(u_{n-3}, u_{n-2}, u_{n-1}, u_n)$ depending on the state $\underline{\mathbf{m}}_{n-3} =(m_{n-3}^1, m_{n-3}^2, m_{n-3}^3, m_{n-3}^4)$

$$\boxed{\begin{aligned} u_{n-3} &= d_{n-3} = m_{n-3}^4 \oplus m_{n-3}^2 \oplus m_{n-3}^1 \\ u_{n-2} &= d_{n-2} = m_{n-2}^4 \oplus m_{n-2}^2 \oplus m_{n-2}^1 = m_{n-3}^3 \oplus m_{n-3}^1 \oplus 0 = m_{n-3}^3 \oplus m_{n-3}^1 \\ u_{n-1} &= d_{n-1} = m_{n-1}^4 \oplus m_{n-1}^3 \oplus m_{n-1}^2 = m_{n-3}^2 \oplus 0 \oplus 0 = m_{n-3}^2 \\ u_n &= d_n = m_{n-3}^1 \oplus 0 \oplus 0 = m_{n-3}^1 \end{aligned}}$$

Table 1.47 — Tail bits for systematic recursive convolutional code

state $\underline{m}_{n-3}$	m_{n-3}^4	m_{n-3}^3	m_{n-3}^2	m_{n-3}^1	u_{n-3}	u_{n-2}	u_{n-1}	u_n
0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	1	0	1
2	0	0	1	0	1	0	1	0
3	0	0	1	1	0	1	1	1
4	0	1	0	0	0	1	0	0
5	0	1	0	1	1	0	0	1
6	0	1	1	0	1	1	1	0
7	0	1	1	1	0	0	1	1
8	1	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	1
10	1	0	1	0	0	0	1	0
11	1	0	1	1	1	1	1	1
12	1	1	0	0	1	1	0	0
13	1	1	0	1	0	0	0	1
14	1	1	1	0	0	1	1	0
15	1	1	1	1	1	0	1	1

1.8.4.6.3 Puncturing of SRC for SRCPC code

Puncturing of the output of the SRC encoder allows different rates for transmission. The puncturing tables are listed in Table 1.48.

Table 1.48 — Puncturing tables (all values in hexadecimal representation)

Rate r	8/8	8/9	8/10	8/11	8/12	8/13	8/14	8/15	8/16	8/17	8/18	8/19	8/20
$P_r(0)$	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
$P_r(1)$	00	80	88	A8	AA	EA	EE	FE	FF	FF	FF	FF	FF
$P_r(2)$	00	00	00	00	00	00	00	00	00	80	88	A8	AA
$P_r(3)$	00	00	00	00	00	00	00	00	00	00	00	00	00

Rate r	8/21	8/22	8/23	8/24	8/25	8/26	8/27	8/28	8/29	8/30	8/31	8/32
$P_r(0)$	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
$P_r(1)$	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
$P_r(2)$	EA	EE	FE	FF	FF	FF	FF	FF	FF	FF	FF	FF
$P_r(3)$	00	00	00	00	80	88	A8	AA	EA	EE	FE	FF

The puncturing pattern, which is applied with the period of 8, depends on the class_rate (see Table 1.48). Each bit of Pr(i) indicates whether the corresponding vt(i) from the SRC encoder is punctured (i.e. not considered). Each bit of Pr(i) is used from MSB to LSB, and 0/1 indicates punctured/not-punctured respectively. The puncturing pattern changes class-wise, but only at points where a period of 8 is completed. After the decision which bits from vt(i) are considered, they are output in the order from vt(0) to vt(3).

1.8.4.6.4 Decoding process of SRCPC code

At the decoder, the error correction should be performed using this SRCPC code, while it is the optional operation and the decoder may extract the original information by just ignoring parity bits.

Decoding of SRCPC can be achieved using Viterbi algorithm for the punctured convolutional coding.

1.8.4.7 Shortened Reed-Solomon codes

Shortened RS codes SRS(255-l, 255-2k-l) defined over GF(2^8) can be used to protect one single class or several concatenated classes. Concatenated classes are subsequently treated as one single class. Here, k is the number of correctable errors in one SRS codeword. The value of l reflects the shortening.

Before the SRS encoding, the EP class is sub-divided into parts such that their lengths are less than or equal to 255-2k. The lengths of the parts are calculated as follows:

l_i = 255-2k, for $i<N$

l_i = L mod (255-2k), for $i = N$

$N = ceil\ (L\ /\ (255\text{-}2k))$

L: The length of the EP class in octets

N: The number of parts

l_i: The length of the i-th part (0<i<N+1)

If the length of the N-th part l_N is smaller than 255-2k bytes, as many bits with the value 0 are added as needed to reach the length of 255-2k bytes before SRS en- or decoding, and again removed afterwards.

At decoder side, if SRS decoding is performed, the same number of ‘0’s have to be added before the SRS decoding procedure, and removed again after SRS decoding.

The SRS code defined in the Galois Field GF(2^8) is generated from a generator polynomial $g(x) = (x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{2k})$, where α denotes a root of the primitive polynomial $m(x)=x^8+x^4+x^3+x^2+1$. The binary representative of α^i is shown in Table 1.49, where the MSB of the octet is transmitted first.

Table 1.49 — Binary representation for α^i ($0 \leq i \leq 254$) over GF(2^8)

a^i	binary rep.	a^i	binary rep.	a^i	binary rep.	a^i	binary rep.
0	00000000	a^{63}	10100001	a^{127}	11001100	a^{191}	01000001
a^{0}	00000001	a^{64}	01011111	a^{128}	10000101	a^{192}	10000010
a^{1}	00000010	a^{65}	10111110	a^{129}	00010111	a^{193}	00011001
a^{2}	00000100	a^{66}	01100001	a^{130}	00101110	a^{194}	00110010
a^{3}	00001000	a^{67}	11000010	a^{131}	01011100	a^{195}	01100100
a^{4}	00010000	a^{68}	10011001	a^{132}	10111000	a^{196}	11001000
a^{5}	00100000	a^{69}	00101111	a^{133}	01101101	a^{197}	10001101
a^{6}	01000000	a^{70}	01011110	a^{134}	11011010	a^{198}	00000111
a^{7}	10000000	a^{71}	10111100	a^{135}	10101001	a^{199}	00001110
a^{8}	00011101	a^{72}	01100101	a^{136}	01001111	a^{200}	00011100
a^{9}	00111010	a^{73}	11001010	a^{137}	10011110	a^{201}	00111000
a^{10}	01110100	a^{74}	10001001	a^{138}	00100001	a^{202}	01110000
a^{11}	11101000	a^{75}	00001111	a^{139}	01000010	a^{203}	11100000
a^{12}	11001101	a^{76}	00011110	a^{140}	10000100	a^{204}	11011101
a^{13}	10000111	a^{77}	00111100	a^{141}	00010101	a^{205}	10100111
a^{14}	00010011	a^{78}	01111000	a^{142}	00101010	a^{206}	01010011
a^{15}	00100110	a^{79}	11110000	a^{143}	01010100	a^{207}	10100110
a^{16}	01001100	a^{80}	11111101	a^{144}	10101000	a^{208}	01010001
a^{17}	10011000	a^{81}	11100111	a^{145}	01001101	a^{209}	10100010
a^{18}	00101101	a^{82}	11010011	a^{146}	10011010	a^{210}	01011001
a^{19}	01011010	a^{83}	10111011	a^{147}	00101001	a^{211}	10110010
a^{20}	10110100	a^{84}	01101011	a^{148}	01010010	a^{212}	01111001
a^{21}	01110101	a^{85}	11010110	a^{149}	10100100	a^{213}	11110010
a^{22}	11101010	a^{86}	10110001	a^{150}	01010101	a^{214}	11111001
a^{23}	11001001	a^{87}	01111111	a^{151}	10101010	a^{215}	11101111
a^{24}	10001111	a^{88}	11111110	a^{152}	01001001	a^{216}	11000011
a^{25}	00000011	a^{89}	11100001	a^{153}	10010010	a^{217}	10011011
a^{26}	00000110	a^{90}	11011111	a^{154}	00111001	a^{218}	00101011
a^{27}	00001100	a^{91}	10100011	a^{155}	01110010	a^{219}	01010110
a^{28}	00011000	a^{92}	01011011	a^{156}	11100100	a^{220}	10101100
a^{29}	00110000	a^{93}	10110110	a^{157}	11010101	a^{221}	01000101
a^{30}	01100000	a^{94}	01110001	a^{158}	10110111	a^{222}	10001010
a^{31}	11000000	a^{95}	11100010	a^{159}	01110011	a^{223}	00001001
a^{32}	10011101	a^{96}	11011001	a^{160}	11100110	a^{224}	00010010
a^{33}	00100111	a^{97}	10101111	a^{161}	11010001	a^{225}	00100100
a^{34}	01001110	a^{98}	01000011	a^{162}	10111111	a^{226}	01001000
a^{35}	10011100	a^{99}	10000110	a^{163}	01100011	a^{227}	10010000
a^{36}	00100101	a^{100}	00010001	a^{164}	11000110	a^{228}	00111101
a^{37}	01001010	a^{101}	00100010	a^{165}	10010001	a^{229}	01111010
a^{38}	10010100	a^{102}	01000100	a^{166}	00111111	a^{230}	11110100
a^{39}	00110101	a^{103}	10001000	a^{167}	01111110	a^{231}	11110101
a^{40}	01101010	a^{104}	00001101	a^{168}	11111100	a^{232}	11110111
a^{41}	11010100	a^{105}	00011010	a^{169}	11100101	a^{233}	11110011
a^{42}	10110101	a^{106}	00110100	a^{170}	11010111	a^{234}	11111011
a^{43}	01110111	a^{107}	01101000	a^{171}	10110011	a^{235}	11101011
a^{44}	11101110	a^{108}	11010000	a^{172}	01111011	a^{236}	11001011
a^{45}	11000001	a^{109}	10111101	a^{173}	11110110	a^{237}	10001011
a^{46}	10011111	a^{110}	01100111	a^{174}	11110001	a^{238}	00001011
a^{47}	00100011	a^{111}	11001110	a^{175}	11111111	a^{239}	00010110
a^{48}	01000110	a^{112}	10000001	a^{176}	11100011	a^{240}	00101100
a^{49}	10001100	a^{113}	00011111	a^{177}	11011011	a^{241}	01011000
a^{50}	00000101	a^{114}	00111110	a^{178}	10101011	a^{242}	10110000
a^{51}	00001010	a^{115}	01111100	a^{179}	01001011	a^{243}	01111101

a^{52}	00010100	a^{116}	11111000	a^{180}	10010110	a^{244}	11111010
a^{53}	00101000	a^{117}	11101101	a^{181}	00110001	a^{245}	11101001
a^{54}	01010000	a^{118}	11000111	a^{182}	01100010	a^{246}	11001111
a^{55}	10100000	a^{119}	10010011	a^{183}	11000100	a^{247}	10000011
a^{56}	01011101	a^{120}	00111011	a^{184}	10010101	a^{248}	00011011
a^{57}	10111010	a^{121}	01110110	a^{185}	00110111	a^{249}	00110110
a^{58}	01101001	a^{122}	11101100	a^{186}	01101110	a^{250}	01101100
a^{59}	11010010	a^{123}	11000101	a^{187}	11011100	a^{251}	11011000
a^{60}	10111001	a^{124}	10010111	a^{188}	10100101	a^{252}	10101101
a^{61}	01101111	a^{125}	00110011	a^{189}	01010111	a^{253}	01000111
a^{62}	11011110	a^{126}	01100110	a^{190}	10101110	a^{254}	10001110

For each of the parts, the SRS parity digits with a total length of $2k$ octets are calculated using $g(x)$ as follows:

$$p(x) = x^{2k} \cdot u(x) \bmod g(x)$$

$u(x)$: polynomial representative of a part. Lowest order corresponds to the first octet.

$p(x)$: polynomial representative of the parity digits. Lowest order corresponds to the first octet.

For storage and transmission, the parity digits are appended at the end of the EP class. This process is illustrated in Figure 1.9.

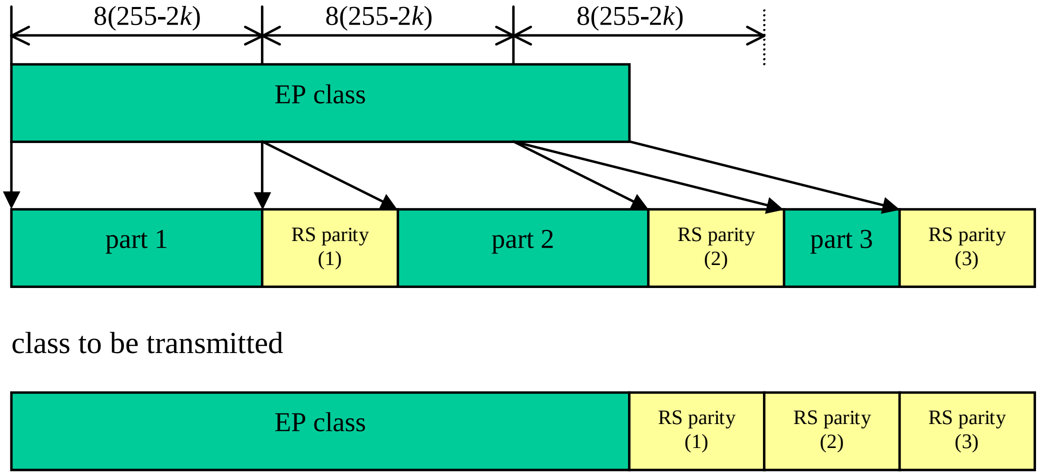


Figure 1.9 — RS encoding of EP frame

1.8.4.8 Recursive interleaving

The interleaving is applied in multi-stage manner. Figure 1.10 shows the interleaving method.

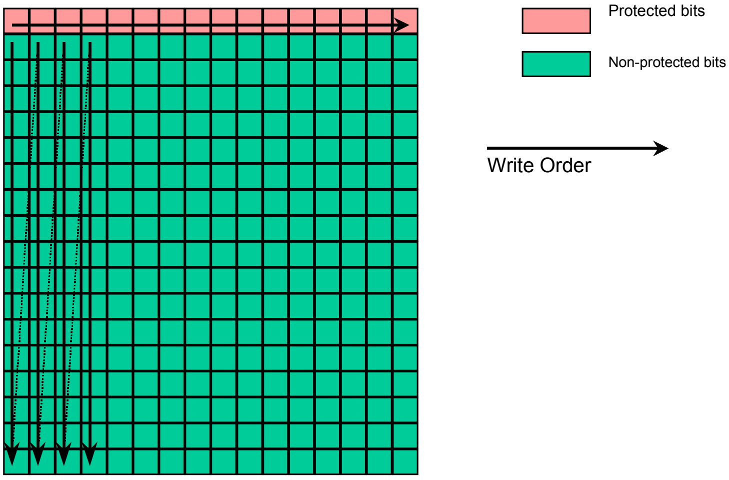


Figure 1.10 — One stage of interleaving

In the multistage interleaving, the output of this one stage of interleaving is treated as a non-protected part in the next stage. Figure 1.11 shows the example of 2 stage interleaving.

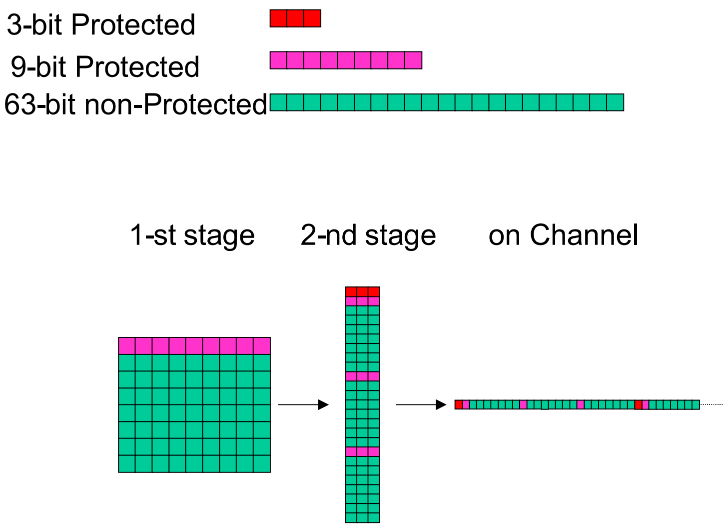


Figure 1.11 — Example of multi-stage interleaving

By choosing the width W of the interleave-matrix to be the same as the FEC code length (or the value 28 in case of SRCPC codes), the interleaving size can be optimized for all the FEC codes.

In actual case, the total number of bits for the interleaving may not allow to use such rectangular. In such case, the matrix as shown in Figure 1.12 is used.

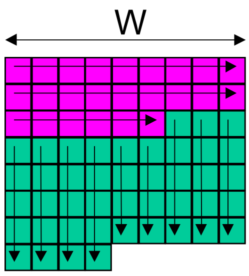


Figure 1.12 — Interleave matrix in non-rectangular case

1.8.4.8.1 Definition of recursive interleaver

Two information streams are input to this interleaver, X_i and Y_i.

X_i, $0 <= i < I_x$

Y_j, $0 <= j < I_y$,

where I_x and I_y is the number of bits for each input streams X_i and Y_j, respectively. X_i is set to the interleaving matrix from the top left to the bottom right, into the horizontal direction. Then Y_i is set into the rest place in vertical direction.

With the width of interleaver W, the size of interleaving matrix is shown as Figure 1.13. Where,

$D = (I_x + I_y) / W$

$d = I_x + I_y - D * W$

Where '/' indicates division by truncation.

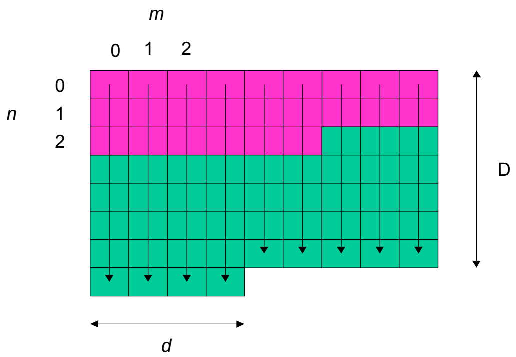


Figure 1.13 — The size of interleaving matrix

The output bitstream payload Z_k ($0 < k <= I_x+I_y$) is read from this matrix from top left to bottom right, column by column in horizontal direction. Thus the bit placed m-th column, n-th row (m and n starts from 0) corresponds to Z_k where:

$k = m * D + \min(m, d) + n$

In the matrix, X_i is set to

$m = i \bmod W, \quad n = i / W,$

Thus Z_k which is set by the X_i becomes:

$Z_k = X_i$, where $k = (i \bmod W) * D + \min(i \bmod W, d) + i / W$

The bits which are set with X_i in the interleaving matrix are shown as Figure 1.14 where:

$D' = l_x / W$

$d' = l_x - D' * W$

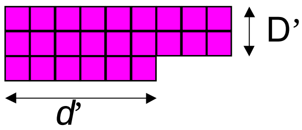


Figure 1.14 — The bits which are set with Xi in the interleaving matrix

Thus, in the m-th row, Y_j is set from the n-th row where n = $D' + (m < d' ? 1 : 0)$ to the bottom. Thus Z_k set by Y_j is represented as follows:

```
Set j to 0;
for m = 0 to D-1 {
    for k = m * D + min(m, d) + D' + (m < d' ? 1 : 0) to (m+1) * D + min(m+1, d) - 1 {
        Z_k = Y_j;
        j ++;
    }
}
```

1.8.4.8.2 Modes of interleaving

Two modes of interleaving, mode 1 and mode 2, according to interleave_type 1 and 2, are defined in the following subclauses. Table 1.50 and Table 1.51 give an overview of the available configurations.

Table 1.50 — Width of the interleaving matrix

interleave_type	fec_type == 0 (SRCPC)	fec_type ==1/2 (SRS)
0	no interleaving	
1	28 bit	length of class
2	depends on interleave_switch (see Table 1.51)	
3	reserved	

Table 1.51 — Width of the interleaving matrix for interleave_type 2

interleave_switch	fec_type == 0 (SRCPC)	fec_type ==1/2 (SRS)
0	no interleaving	
1	length of class	length of class
2	28 bit	not permitted
3	concatenation	

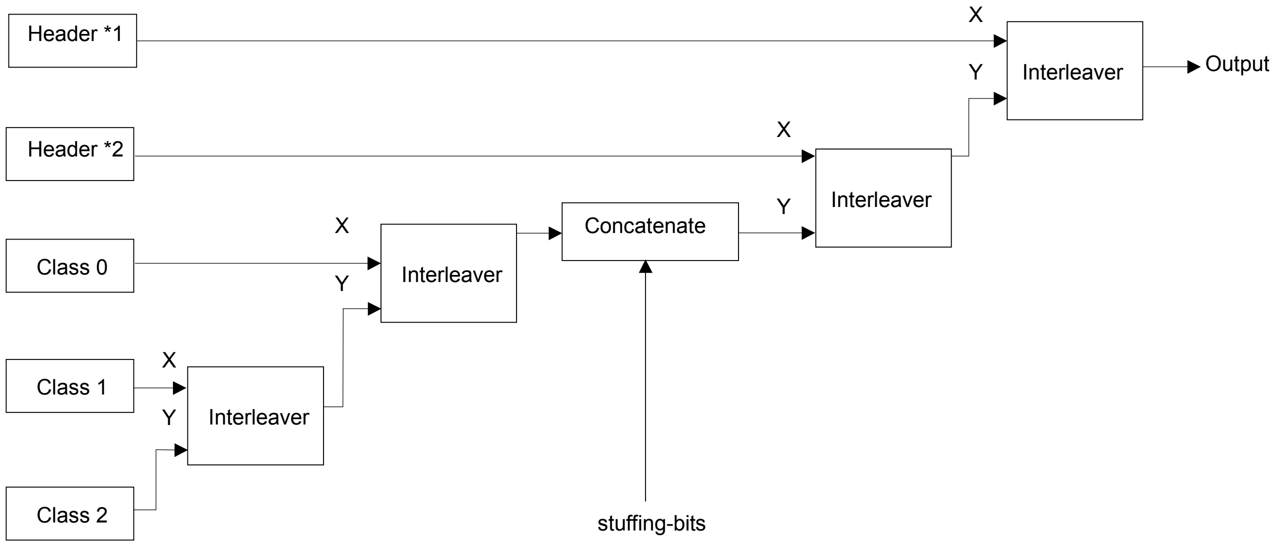
In the case of fec_type=0 (SRCPC), the interlaving is performed bitwise. In the case of fec_type == 1 or fec_type == 2 (SRS), the interleaving is performed bytewise.

1.8.4.8.2.1 Interleaving operation in mode 1

Multi-stage interleaving is processed for **ep_encoded_class** from the last class to first class, then the stuffing-bits are appended at the end of the interleaved classes. The interleaving process continues with class attribution part of ep_header() (which is class_attrib() + **class_attrib_parity**), and the pre-defined part of ep_header() (which is **choice_of_pred** + **choice_of_pred_parity),** as illustrated in Figure 1.15.

The width of the interleaving matrix is chosen according to the FEC in use. In the case of SRCPC coding (fec_type == 0), the width of the interleaving matrix is 28 bit. In the case of SRS coding (fec_type == 1 or 2), the width of the interleaving matrix is equal to the length of the class in bytes. Figure 1.16 shows the interleaving scheme principle for the latter case. The bits in the class are written into the interleaving matrix byte by byte for each column.

The width of the interleaving matrix for the header parts is either equal to the length of the codeword (in bits) provided by the block code according to Table 1.48, or 28 bits if SRCPC is used.



*1 First part of Header : class_attrib() followed by class_attrib_parity
*2 Second part of Header : choice_of_pred followed by choice_of_pred_parity

Figure 1.15 — Interleaving process of mode1 specification

1.8.4.8.2.2 Interleaving operation in mode 2

In mode 2, a flag indicates whether the class is processed with interleaver, and how it is interleaved. This flag interleave_switch is signaled within the out-of-band information. The value 0 indicates the class is not processed by the interleaver. The value 1 indicates the class is interleaved by the recursive interleaver, and the length of the class is used as the width of the interleaver (either the length in bits in case of SRCPC or the length in bytes in case of SRS). The value 2 indicates the class is interleaved by the recursive interleaver, and the width is set to be equal to 28 (permitted only in case of SRCPC). The value 3 indicates the class is concatenated but not interleaved by the recursive interleaver. The interleaving operation for the ep_header is same as mode 1.

Figure 1.16 shows the interleaving scheme principle for fec_type == 1 or 2 (SRS) and interleave_switch == 1. The width is set to be the number of bytes in the class. The bits in the class are written into the interleaving matrix byte by byte for each column.

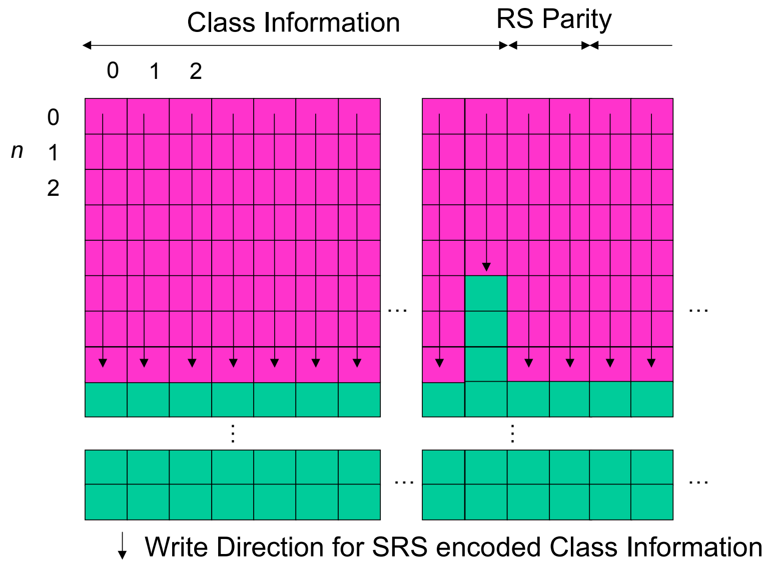


Figure 1.16 — Interleaving matrix in RS encoded class case

The interleaving process to obtain interleaved_frame_mode2 is as follows (N: number of classes):

```
    clear buffer BUF_NO /* Buffer for non-interleaved part. */
    clear buffer BUF_Y  /* Buffer for Y input in the next stage */
    for( j = 0; j < N; j++ )
       if ( class_reordered_output == 1 ) {
          k = class_output_order[choice_of_pred][j];
       } else {
          k = j;
       }
       if ( interleave_switch[choice_of_pred][k] == 3) {
       add ep_encoded_class[k] to BUF_Y;
       }
       if ( interleave_switch[choice_of_pred][k] == 0 ) {
       add ep_encoded_class[k] to BUF_NO;
       }
    }
    for ( j = N-1; j >= 0; j-- ) {
       if ( class_reordered_output == 1 ) {
          k = class_output_order[choice_of_pred][j];
       } else {
          k = j;
       }
       if ( ( interleave_switch[choice_of_pred][k] != 0 )
             && ( interleave_switch[choice_of_pred][k] != 3 ) ){
          if ( interleave_switch[choice_of_pred][k] == 1 ) {
             set the size of the interleave window to be the length of ep_encoded_class[k];
          } else if ( interleave_switch[choice_of_pred][k] == 2 ) {
             set the size of the interleave window to be 28;
          }
          input ep_encoded_class[k] into the recursive interleaver as X input;
          input BUF_Y into the recursive interleaver as Y input;
          set the output of the interleaver into BUF_Y;
       }
    }
    add BUF_NO to BUF_Y;
    if( bit_stuffing ) {
       add Nstuff stuffing-bits to BUF_Y;
    }
    input class_attrib() followed by class_attrib_parity into the recursive interleaver as X
input;
    input BUF_Y into the recursive interleaver as Y input;
```

```
    set the output of the interleaver into BUF_Y;
    input choice_of_pred followed by choice_of_pred_parity into the recursive interleaver as X
input;
    input BUF_Y into the recursive interleaver as Y input;
    set the output of the interleaver into BUF_Y;
    set BUF_Y into interleaved_frame_mode2;
```

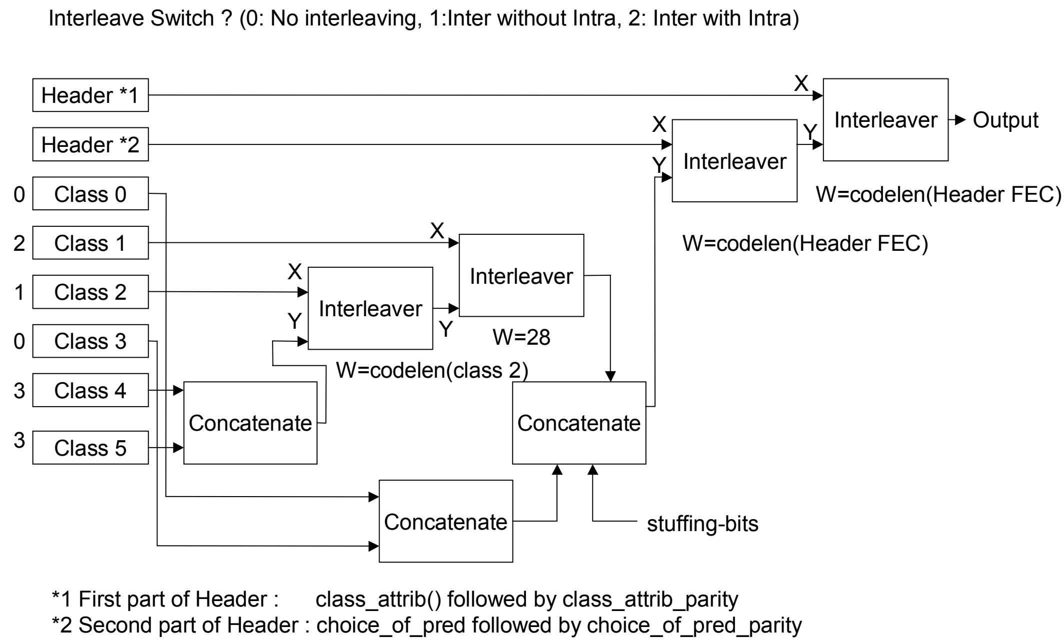


Figure 1.17 — Interleave process with class-wise control of interleaving

1.8.4.9 Class reordered output

The EP tool allows to reorder the classes such that it does not have to stick at the order provided / required by the audio codec. Figure 1.18 gives an example of the reordering process on decoder side. The class order after this reordering is signaled as **class_output_order[i][j]** in the out-of-band information. The EP decoder reorders the classes in the EP frame using i-th pre-defined set, so that the j-th class of EP frame is output as (**class_output_order[i][j]**)-th class to the audio decoder.

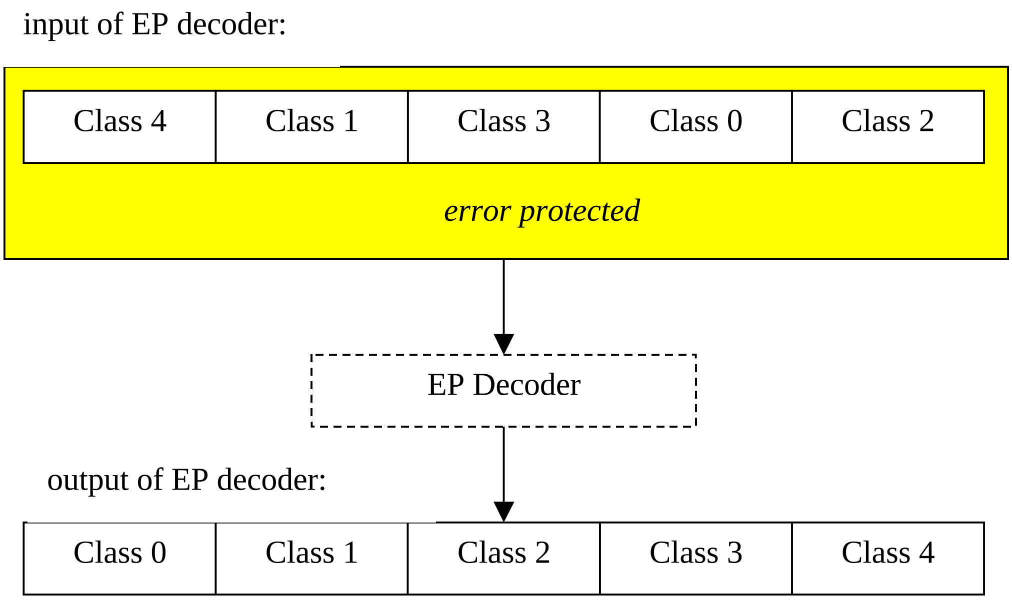


Figure 1.18 — Example for class reordered output with class_output_order 4, 1, 3, 0, 2

Annex 1.A
(informative)

Audio Interchange Formats

1.A.1 Introduction

The full capabilities and flexibility of MPEG-4 Audio, like the composition of audio scenes out of multiple audio objects, synthetic audio, and text to speech conversion, is available only if MPEG-4 Audio is used together with MPEG-4 Systems (ISO/IEC-14496-1). The interchange formats, defined in here in Annex A, only support a small subset of the capabilities of MPEG-4 Audio, by defining formats for the storage and transmission of a single mono or stereo or multi-channel audio object, very similar to the formats defined in MPEG-1 and MPEG-2.
As already stated in the introduction to subpart 1, the normative elements in MPEG-4 Audio end with the definition of the payloads (roughly equivalent to a bitstream frame in MPEG-1 and MPEG-2), and the coder configuration structures (resembling the MPEG-1/2 header information). However, there is no normative definition in MPEG-4 Audio how these elements are multiplexed, as this is required only for a limited number of applications. Nevertheless, this informative annex describes such a multiplex. However, MPEG-4 decoders are not obliged to comply to these interface formats.

1.A.2 AAC Interchange formats

1.A.3 Syntax

1.A.3.1 MPEG-2 AAC Audio_Data_Interchange_Format, ADIF

Table 1.A.1 — Syntax of adif_sequence

Syntax	No. of bits	Mnemonic

adif_sequence()
{
    adif_header();
    byte_alignment();
    raw_data_stream();
}		

Table 1.A.2 — Syntax of adif_header()

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

Table 1.A.3 — Syntax of raw_data_stream()

Syntax	No. of bits	Mnemonic
raw_data_stream()		
{		
while (data_available()) {		
raw_data_block();		
}		
}		

1.A.3.2 Audio_Data_Transport_Stream frame, ADTS

Table 1.A.4 — Syntax of adts_sequence()

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

Table 1.A.5 — Syntax of adts_frame()

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

1.A.3.2.1 Fixed Header of ADTS

Table 1.A.6 — Syntax of adts_fixed_header()

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

1.A.3.2.2 Variable Header of ADTS

Table 1.A.7 — Syntax of adts_variable_header()

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

1.A.3.2.3 Error detection

Table 1.A.8 — Syntax of adts_error_check

Syntax	No. of bits	Mnemonic
adts_error_check()		
{		
if (protection_absent == '0')		
crc_check;	**16**	**Rpchof**
}		

Table 1.A.9 — Syntax of adts_header_error_check

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

Table 1.A.10 — Syntax of adts_raw_data_block_error_check()

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

1.A.4 Semantic

1.A.4.1 Overview

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

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

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

1.A.4.2 Audio Data Interchange Format (ADIF)

The semantic of the Audio Data Interchange Format (ADIF) is specified in ISO/IEC 13818-7. The following additional definitions apply in the context of MPEG-4:

raw_data_stream() sequence of raw_data_block()'s.

program_config_element() Contains information about the configuration for one program. See subpart 4 for the definition.

raw_data_block() Defined in subpart 4.

1.A.4.3 Audio Data Transport Stream (ADTS)

The semantic of the Audio Data Transport Stream (ATDS) is specified in ISO/IEC 13818-7. The following changes apply when used in MPEG4:

raw_data_block() Defined in subpart 4.

ID MPEG identifier, set to '1' if the audio data in the ADTS stream is MPEG-2 AAC (see ISO/IEC 13818-7) and set to '0' if the audio data is MPEG-4. See also ISO/IEC 11172-3, subclause 2.4.2.3.

profile_ObjectType The interpretation of this data element depends on the value of the ID bit. If ID is equal to '1' this field holds the same information as the profile field in the ADTS stream defined in ISO/IEC 13818-7. If ID is equal to '0' this element denotes the MPEG-4 Audio Object Type (profile_ObjectType+1) according to the table defined in subclause 1.5.2.1.

Table 1.A.11 — MPEG-2 Audio profiles and MPEG-4 Audio object types

profile_ObjectType	MPEG-2 profile (ID == 1)	MPEG-4 object type (ID == 0)
0	Main profile	AAC Main
1	Low Complexity profile (LC)	AAC LC
2	Scalable Sampling Rate profile (SSR)	AAC SSR
3	(reserved)	AAC LTP

sampling_frequency_index Indicates the sampling frequency used according to the table defined in subclause 1.6.3.4. The escape value is not permitted.

channel_configuration Indicates the channel configuration used. In the case of (channel_configuration > 0), the channel configuration is given in Table 1.17. In the case of (channel_configuration == 0), the channel configuration is not specified in the header, but as follows:

MPEG-2/4 ADTS: A single program_config_element() following as first syntactic element in the first raw_data_block() after the header specifies the channel configuration. Note that the program_config_element() might not be present in each frame. An MPEG-4 ADTS decoder should not generate any output until it received a program_config_element(), while an MPEG-2 ADTS decoder may assume an implicit channel configuration.

MPEG-2 ADTS: Beside the usage of a program_config_element(), the channel configuration may be assumed to be given implicitly (see ISO/IEC13818-7) or may be known in the application.

Annex 1.B
(informative)

Error protection tool

Text file format of out-of-band information, and its example for AAC, Twin-VQ, CELP, and HVXC are presented. In addition, the example of error concealment is presented.

1.B.1 Example of out-of-band information

1.B.1.1 Example for AAC

based on the error sensitivity category assignement described within the normative part, the following error protection setup could be used, while sensitivity categories are directly mapped to classes. This example shows just a simple setup using one channel and no extension_payload().

class	length	interleaving	SRCPC puncture rate	CRC length
0	6 bit in-band field	intra-frame	8/24	6
1	12 bit in-band field	intra-frame	8/24	6
2	9 bit in-band field	inter-frame	8/8	6
3	9 bit in-band field	none	8/8	4
4	until the end	inter-frame	8/8	none

1.B.1.2 Example for Twin-VQ

This subclause describes examples of the bit assignment of UEP to the Scalable Audio Profile (TwinVQ object).

Here two encoding modes, PPC (Periodic Peak Component) - enable mode and disable mode, are described. Normally, encoder can adaptively select the PPC switch, but we force the switch always ON or always OFF in this experiment. If PPC is ON, 43 bits are assigned to quantize periodic peak components and these bits should be protected as side information.

For each mode we show bit assignment of four different bitrates, 16 kbit/s mono, 32 kbit/s stereo, 8 kbit/s + 8 kbit/s scalable mono and 16 kbit/s + 16 kbit/s stereo for each mode.

In all cases, error correction and detection tools are applied to only 10 % of bits for the side information. Remaining bits for the index of MDCT coefficients have no protection at all. As a result of these bit allocations, increase of bitrate comparing with the original source rate is around 10 % in case of PPC switch is ON, and less than 10% in case of the switch is OFF.

(A) PPC(Periodic Peak Component) enable version

- 16 kbit/s mono
  Class 1: 121 bit(fixed), SRCPC code rate 8/12, 8 bit CRC
  Class 2: 839 bit(fixed), SRCPC code rate 8/8, no CRC

- 32 kbit/s stereo
  Class 1: 238 bit(fixed), SRCPC code rate 8/12, 10 bit CRC
  Class 2: 1682 bit(fixed), SRCPC code rate 8/8, no CRC

- 8 kbit/s + 8 kbit/s scalable mono
  Class 1: 121 bit(fixed), SRCPC code rate 8/12, 8 bit CRC
  Class 2: 359 bit(fixed), SRCPC code rate 8/8, no CRC

Class 3: 72 bit(fixed), SRCPC code rate 8/12, 8 bit CRC
Class 4: 408 bit(fixed), SRCPC code rate 8/8, no CRC

- 16 kbit/s + 16 kbit/s scalable stereo
  Class 1: 238 bit(fixed), SRCPC code rate 8/12, 10 bit CRC
  Class 2: 722 bit(fixed), SRCPC code rate 8/8, no CRC
  Class 3: 146 bit(fixed), SRCPC code rate 8/12, 10 bit CRC
  Class 4: 814 bit(fixed), SRCPC code rate 8/8, no CRC

(B)PPC disable version

- 16 kbit/s mono
  Class 1: 78 bit(fixed), SRCPC code rate 8/12, 8 bit CRC
  Class 2: 882 bit(fixed), SRCPC code rate 8/8, no CRC

- 32 kbit/s stereo
  Class 1: 152 bit(fixed), SRCPC code rate 8/12, 10 bit CRC
  Class 2: 1768 bit(fixed), SRCPC code rate 8/8, no CRC

- 8 kbit/s + 8 kbit/s scalable mono
  Class 1: 78 bit(fixed), SRCPC code rate 8/12, 8 bit CRC
  Class 2: 402 bit(fixed), SRCPC code rate 8/8, no CRC
  Class 3: 72 bit(fixed), SRCPC code rate 8/12, 8 bit CRC
  Class 4: 408 bit(fixed), SRCPC code rate 8/8, no CRC

- 16 kbit/s + 16 kbit/s scalable stereo
  Class 1: 152 bit (fixed), SRCPC code rate 8/12, 10 bit CRC
  Class 2: 808 bit (fixed), SRCPC code rate 8/8, no CRC
  Class 3: 146 bit (fixed), SRCPC code rate 8/12, 10 bit CRC
  Class 4: 814 bit (fixed), SRCPC code rate 8/8, no CRC

1.B.1.3 Example for CELP

The following tables provide an overview of the number of bit that are assigned to each error sensitivity category, dependent on the configuration.

1.B.1.3.1 MPE-Narrowband Mode

Table 1.B.1 — Overview of bit assignment for MPE-narrowband

MPE-Mode	subframes	Bit/Frame	bitrate	ECR0	ECR1	ECR2	ECR3	ECR4
0	4	154	3850	6	13	20	37	78
1	4	170	4250	6	13	20	41	90
2	4	186	4650	6	13	20	45	102
3	3	147	4900	5	11	16	36	79
4	3	156	5200	5	11	16	39	85
5	3	165	5500	5	11	16	42	91
6	2	114	5700	4	9	12	29	60
7	2	120	6000	4	9	12	31	64
8	2	126	6300	4	9	12	33	68
9	2	132	6600	4	9	12	35	72
10	2	138	6900	4	9	12	37	76
11	2	142	7100	4	9	12	39	78
12	2	146	7300	4	9	12	41	80
13	4	154	7700	6	13	20	41	74
14	4	166	8300	6	13	20	45	82
15	4	174	8700	6	13	20	49	86
16	4	182	9100	6	13	20	53	90

17	4	190	9500	6	13	20	57	94
18	4	198	9900	6	13	20	61	98
19	4	206	10300	6	13	20	65	102
20	4	210	10500	6	13	20	69	102
21	4	214	10700	6	13	20	73	102
22	2	110	11000	4	9	12	33	52
23	2	114	11400	4	9	12	35	54
24	2	118	11800	4	9	12	37	56
25	2	120	12000	4	9	12	39	56
26	2	122	12200	4	9	12	41	56
27	4	186	6200	6	13	20	49	98
28	reserved							
29	reserved							
30	reserved							
31	reserved							

1.B.1.3.2 MPE-Wideband Mode

Table 1.B.2 — Overview of bit assignment for MPE-wideband

MPE-Mode	subframes	Bit/Frame	bitrate	ECR0	ECR1	ECR2	ECR3	ECR4
0	4	218	10900	17	20	27	45	109
1	4	230	11500	17	20	27	49	117
2	4	242	12100	17	20	27	53	125
3	4	254	12700	17	20	27	57	133
4	4	266	13300	17	20	27	61	141
5	4	278	13900	17	20	27	65	149
6	4	286	14300	17	20	27	69	153
7	reserved							
8	8	294	14700	17	32	43	61	141
9	8	318	15900	17	32	43	69	157
10	8	342	17100	17	32	43	77	173
11	8	358	17900	17	32	43	85	181
12	8	374	18700	17	32	43	93	189
13	8	390	19500	17	32	43	101	197
14	8	406	20300	17	32	43	109	205
15	8	422	21100	17	32	43	117	213
16	2	136	13600	17	14	19	29	57
17	2	142	14200	17	14	19	31	61
18	2	148	14800	17	14	19	33	65
19	2	154	15400	17	14	19	35	69
20	2	160	16000	17	14	19	37	73
21	2	166	16600	17	14	19	39	77
22	2	170	17000	17	14	19	41	79
23	reserved							
24	4	174	17400	17	20	27	37	73
25	4	186	18600	17	20	27	41	81
26	4	198	19800	17	20	27	45	89
27	4	206	20600	17	20	27	49	93
28	4	214	21400	17	20	27	53	97
29	4	222	22200	17	20	27	57	101
30	4	230	23000	17	20	27	61	105
31	4	238	23800	17	20	27	65	109

1.B.1.3.3 RPE-Wideband Mode

Table 1.B.3 — Characteristic parameters for wideband CELP with RPE

RPE Mode	subframes	Bit/frame	bitrate	ERC0	ERC1	ERC2	ERC3	ERC4
0	6	216	14400	40	24	34	25	93
1	4	160	16000	32	18	26	21	63
2	8	280	18667	48	30	42	29	131
3	10	338	22533	56	36	50	33	163

1.B.1.4 Example for HVXC

- 2 kbit/s source coder
  Class 1: 22 bit (fixed), SRCPC code rate 8/16, 6 bit CRC
  Class 2: 4 bit (fixed), SRCPC code rate 8/8,1 bit CRC
  Class 3: 4 bit (fixed), SRCPC code rate 8/8,1 bit CRC
  Class 4: 10 bit (fixed), SRCPC code rate 8/8, no CRC

- 4 kbit/s source coder
  Class 1: 33 bit (fixed), SRCPC code rate 8/16, 6 bit CRC
  Class 2: 22 bit (fixed), SRCPC code rate 8/8, 6 bit CRC
  Class 3: 4 bit (fixed), SRCPC code rate 8/8, 1 bit CRC
  Class 4: 4 bit (fixed), SRCPC code rate 8/8, 1 bit CRC
  Class 5: 17 bit (fixed), SRCPC code rate 8/8, no CRC

1.B.1.5 Example for ER BSAC

This subclause describes examples of the bit assignment of Unequal Error Protection (UEP) to the ER BSAC Object Type.

The lower error sensitivity category (ESC) described in subpart 4, subclause 4.5.2.6.3 indicates the class with the higher error sensitivity, whereas the higher ESC indicates the class with the lower sensitivity. Based on the error sesitivity category of the BSAC syntax, the following error protection setup example could be used, This example shows just a simple setup where sensitivity categories are directly mapped to classes.

Class	category	length	interleaving	SRCPC puncture rate	CRC length
0	0	9 bit in-band field	intra-frame	8/24	6
1	others	11 bit in-band field	no	8/8	none

In this example, error correction and detection tools are applied to only the general side information. Remaining bits for the index of MDCT coefficients have no protection at all. As a result of these bit allocations, increase of bitrate comparing with the original source rate is around 10 %.

And, the predefined set for UEP can be set up in addition to the UPE class setup as follows :

1.B.2 Example of error concealment

The error concealment tool is an optional decoder tool to reduce the quality degradation of decoded signals when the decoder input bitstream payload is affected by errors, such as bitstream payload transmission error. This is especially valid in applying the MPEG-4 Audio tools to radio applications. The error detection and decision method to replace a frame is not defined in this subclause and decided based on each application.

1.B.2.1 General

Detailed examples of the out-of-band information for the individual codec types are provided in ISO/IEC14496-5 in conjunction with the ep-tool software. Further examples are given in ISO/IEC 14496-4 by means of conformance test sequences with epConfig=2 and epConfig=3.

1.B.2.2 Example for CELP

1.B.2.2.1 Overview of the error concealment tool

The error concealment tool is used with the MPEG-4 CELP decoder described in ISO/IEC 14496-3. This tool reduces unpleasant noise when the MPEG-4 CELP decodes speech from erroneous input frame data. It also enables the MPEG-4 CELP to decode speech even if the input frame data is lost.

The tool has two operating modes|: a Bit Error (BE) mode and a Frame Erasure (FE) mode. The mode is switched based on the availability of frame data at the decoder. When frame data is available (the BE mode), decoding is done using the frame data received in the past and the usable subpart of the current frame data. When frame data is not available (the FE mode), the decoder generates the speech using only the past frame data.

This tool operates according to a flag (the BF_flag) that indicates whether the frame data is complete (BF_flag=0), or is damaged by corruption and/or lost (BF_flag=1). The flag is usually given by the channel coder or the transmission system.

This tool operates in Coding Mode II (subclause 3.1.2.1 of ISO/IEC 14496-3:2001), which has narrow band, wideband and band width scalable modes that use Multi-Pulse Excitation at sampling rates of 8 and 16 kHz.

1.B.2.2.2 Definitions

BE: Bit Error
BWS: BandWidth Scalable
FE: Frame Erasure
LP: Linear Prediction
LSP: Line Spectral Pair
MPE: Multi-Pulse Excitation
NB: Narrow Band
RMS: Root Mean Square (the frame energy)
WB: WideBand

1.B.2.2.3 Helping variables

frame_size: the number of samples in a frame
g_ac: the adaptive codebook gain
g_ec: the MPE gain
lpc_order: the order of LP
signal_mode: the speech mode
signal_mode_pre: the speech mode of the previous frame

1.B.2.2.4 Specifications of the error concealment tool

The error concealment tool operates based on the transition model with six states depicted in Figure 1.B.1. The state indicates the quality of the transmission channel. The bigger the state number is, the worse the channel quality is. Each state has a different concealment operation. The initial state in decoding is State 0 and the state is transited based on the *BF_flag*. Each concealment operation is described in the following subclauses.

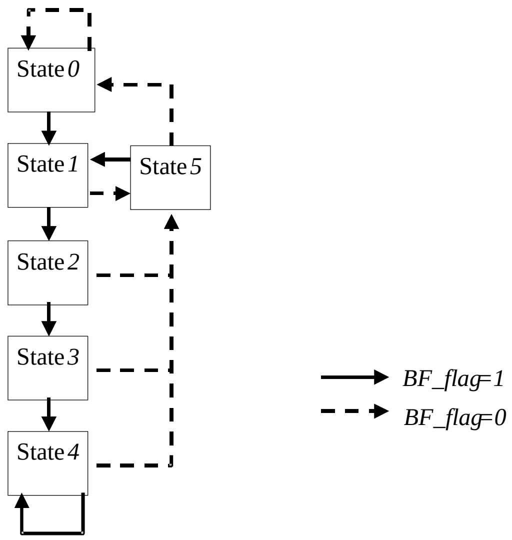


Figure 1.B.1 — State transition model for controlling the error concealment

1.B.2.2.4.1 Operations in States *0* and *5*

The decoding process is identical to that of the MPEG-4 CELP decoder with the following exceptions regarding the adaptive codebook and the excitation codebook gains:

In State *0* following State *5*, and in State *5*:

(1) for the first *80* samples in and after the frame where the *BF_flag* is changed from *1* to *0*, the gains *g_ac* and *g_ec* are calculated from the gains *g_ac'* and *g_ec'* decoded from the current frame data as follows:

```
if (g_ac > 1.0) {
        g_ec = g_ec'/ g_ac';
        g_ac=1.0;
}
```

(2) for the *160* samples that follow the first *80* samples, the gains are calculated as:

```
if(g_ac > 1.0){
        g_ec = g _ec'*(g_ac'+1.0) /g_ac'/2.0 ;
        g_ac = (g_ac'+1.0)/2.0;
},
```

where these operations continue in, at most, four subframes.

1.B.2.2.4.2 Operations in States *1*, *2*, *3* and *4*

The decoding process is identical to that of the MPEG-4 CELP decoder with the exceptions described in the following subclauses.

1.B.2.2.4.2.1 Speech mode

1.B.2.2.4.2.1.1 FE mode

The speech mode (*signal_mode*) is decoded from the previous frame data.

1.B.2.2.4.2.1.2 BE mode

The speech mode (*singal_mode*) is decoded from the current frame data for *signal _mode_pre=0* or *1*. Otherwise, the mode is decoded from the previous frame data.

1.B.2.2.4.2.2 Multi-Pulse Excitation (MPE)

1.B.2.2.4.2.2.1 FE mode

The MPE is decoded from the randomly generated frame data.

1.B.2.2.4.2.2.2 BE mode

The MPE is decoded from the frame data received in the current frame.

1.B.2.2.4.2.3 RMS

For State *1* through *K*, the RMS in the last subframe of the previous frame is used after being attenuated in the first subframe of the current frame. In the following subframes, the RMS in the previous subframe is used after being attenuated. The attenuation level P_{att} depends on the state as follows:

$$P_{att} = \begin{cases} 0.4\ dB & for\ \text{State}\ 1,\ldots,K \\ 1.2\ dB & for\ \text{State}\ K+1,\ldots \end{cases},$$

where K is the smaller number of 4 and K_0, and K_0 is the maximum integer satisfying $frame\_size \times K_0 \le 320$.

1.B.2.2.4.2.4 LSP

The LSPs decoded in the previous frame are used. In the BWS mode, the LSP codevectors should be buffered for the interframe prediction described in subclause 3.5.6.3.3 of ISO/IEC 14496-3:2001. However, when the frame data is corrupted or lost, the correct codevector can not be obtained. Therefore, the buffered codevector (*blsp*[*0*][]) is estimated from the LSP (*qlsp_pre*[]) in the previous frame, the predicted LSP (*vec_hat*[]) and the prediction coefficient (*cb*[*0*][]) in the current frame as follows:

```
for (i = 0; i < lpc_order; i++) {
      blsp[0] [i] = (qlsp_pre[i] - vec_hat[i])/cb[0] [i];
}
```

1.B.2.2.4.2.5 Delay of the adaptive codebook

1.B.2.2.4.2.5.1 FE mode

All the delays of the adaptive codebook are decoded from the delay index received in the last subframe of the previous frame.

1.B.2.2.4.2.5.2 BE mode

(1) When *signal_mode_pre=0*, the delays are decoded from the current frame data.

(2) When *signal_mode_pre*=1, and if the maximum difference between the delay indices of the adjacent subframes in the frame are less than *10*, the delays are decoded from the delay indices in the current frame. In each subframe where the difference in the delay indices between the current and the previous subframes is equal to or greater than *10*, the delay is decoded from the index of the previous subframe.

(3) When *signal_mode_pre= 2* or *3*, the delay is decoded from the index in the last subframe of the previous frame.

1.B.2.2.4.2.6 Gains

1.B.2.2.4.2.6.1 Index operation

1.B.2.2.4.2.6.1.1 FE mode

All the gains have the same value, which is decoded from the gain index received in the last subframe of the previous frame.

1.B.2.2.4.2.6.1.2 BE mode

The gains are decoded from current frame data.

1.B.2.2.4.2.6.2 Adjustment operation

1.B.2.2.4.2.6.2.1 FE mode

(1) When *signal_mode_pre=0*, the gains *g_ac'* and *g_ec'* are decoded from the current frame data. The gains *g_ac* and *g_ec* are then obtained by multiplying *g_ac'* by *0.5* and *g_ec'* by *X,* respectively. *X* satisfies the following equation and is calculated in each subframe:

(*g_ac'***g_ac'*)A+(*g_ec'***g_ec'*)B=((0.5**g_ac')* *(0.5**g_ac')*)A+((*X***g_ec') * (X***g_ec'))B*

where

$A = norm_{ac} \times norm_{ac}$

$B = norm_{ec} \times norm_{ec}$.

$norm_{ac}$ and $norm_{ec}$ are the respective RMS values of the adaptive and the excitation codevectors.

(2) When *signal_mode_pre=1*, the gains are decoded from the current frame data.

(3) When *signal_mode_pre=2* or *3*, gains for the first *320* samples in or after the frame where the *BF_flag* is changed from *0* to *1,* are calculated as:

$g\_ac = 0.95 \times 10^{(-0.4/20)}$

$g\_ec = X \times 10^{(-0.4/20)}$.

After the first *320* samples,

$g\_ac = 0.95 \times 10^{(-1.2/20)}$

$g\_ec = X \times 10^{(-1.2/20)}$,

where $X = 0.05 \times \frac{norm_{ac}}{norm_{ec}}$.

1.B.2.2.4.2.6.2.2 BE mode

(1) When *signal_mode_pre=0* or *1*, the gains are calculated using the gains *g_ac'* and *g_ec'* decoded from the current frame data and the gains *g_ac_pre* and *g_ec_pre* of the previous subframe so that the calculated gains fall in a normal range and generate no unpleasant noise as follows:

```
if (g_ac > 1.2589) {
    g_ec = g_ec' * 1.2589/g_ac';
    g_ac = 1.2589;
}
if (g_ec > 1.2589*g_ec_pre) {
    g_a c= g_ac'* 1.2589*g_ec_pre/g_ec' ;
g_ec  = g_ec_pre*1.2589;
}
if (signal_mode = 1 &  g_ac_pre < 1.2589 & g_ac < g_ac_pre*0.7943) {
    g_ac = g_ac_pre*0.7943;
    g_ec = g_ec_pre*0.7943;
}.
```

(2) When *signal_mode_pre=2* or *3*, the operation is identical to that for *signal_mode_pre=2* or *3* in the FE mode.

1.B.2.3 Error concealment for the silence compression tool

In frames where the bitstream payload received at the decoder is corrupted or lost due to transmission bit errors, error concealment is performed. When the received TX_flag is *1*, the decoding process is identical to that of the error concealment for the MPEG-4 CELP. For TX_flag=*0, 2* or *3*, the decoding process for the TX_flag=*0* is used.

1.B.3 Example of EP tool setting and error concealment for HVXC

This subclause describes one example of the implementation of EP (Error Protection) tool and error concealment method for HVXC. Some of perceptually important bits are protected by FEC (forward error correction) scheme and some are checked by CRC to judge whether or not erroneous bits are included. When CRC error occures, error concealment is executed to reduce perceptible degradation.

It should be noted that error correction method and EP tool setting, error concealment algorithm described below are one example, and they should be modified depending on the actual channel conditions.

1.B.3.1 Definitions

--- 2/4 kbit/s common parameters ---

LSP1	LSP index 1	(5 bit)
LSP2	LSP index 2	(7 bit)
LSP3	LSP index 3	(5 bit)
LSP4	LSP index 4	(1 bit)
VUV	voiced/unvoiced flag	(2 bit)
Pitch	pitch parameter	(7 bit)
SE_shape1	spectrum index 0	(4 bit)
SE_shape2	spectrum index 1	(4 bit)
SE_gain	spectrum gain index	(5 bit)
VX_shape1[0]	stochastic codebook index 0	(6 bit)
VX_shape1[1]	stochastic codebook index 1	(6 bit)
VX_gain1[0]	gain codebook index 0	(4 bit)
VX_gain1[1]	gain codebook index 1	(4 bit)

--- only 4 kbit/s parameters ---

LSP5	LSP index 5	(8 bit)
SE_shape3	4k spectrum index 0	(7 bit)
SE_shape4	4k spectrum index 1	(10 bit)

SE_shape5	4k spectrum index 2	(9 bit)
SE_shape6	4k spectrum index 3	(6 bit)
VX_shape2[0]	4k stochastic codebook index 0	(5 bit)
VX_shape2[1]	4k stochastic codebook index 1	(5 bit)
VX_shape2[2]	4k stochastic codebook index 2	(5 bit)
VX_shape2[3]	4k stochastic codebook index 3	(5 bit)
VX_gain2[0]	4k gain codebook index 0	(3 bit)
VX_gain2[1]	4k gain codebook index 1	(3 bit)
VX_gain2[2]	4k gain codebook index 2	(3 bit)
VX_gain2[3]	4k gain codebook index 3	(3 bit)

1.B.3.2 Channel coding

1.B.3.2.1 Protected bit selection

According to the sensitivity of bits, encoded bits are classified to several classes. The number of bits for each class is shown in the Table 1.B.4, Table 1.B.5 (2 kbit/s), Table 1.B.6 and Table 1.B.7 (4 kbit/s). As an example, bitrate setting of 3.5 kbit/s(for 2 kbit/s) and 6.2 kbit/s (for 4 kbit/s) are shown. In these cases, two source coder frames are processed as one set. Suffix "p" means parameters of the "previous" frame, and "c" means those of the "current" frame.

For 3.5 kbit/s mode, 6 classes are used. CRC check is applied for class I, II, III, IV, and V bits. Class VI bits are not checked by CRC.

The Table 1.B.4 below shows protected/unprotected(class I…VI) bit assignment in the case where both previous and current frames are voiced.

Table 1.B.4 — Number of protected/unprotected bits at 3.5 kbit/s (voiced frame)

para-meters	voiced frame						
	class I bits	**class II bits**	**class III bits**	**class IV bits**	**class V bits**	**class VI bits**	**total**
LSP1p/c	5/5	-	-	-	-	-	**10**
LSP2p/c	2/2	-	-	-	-	5/5	**14**
LSP3p/c	1/1	-	-	-	-	4/4	**10**
LSP4p/c	1/1	-	-	-	-	-	**2**
VUVp/c	2/2	-	-	-	-	**-**	**4**
Pitchp/c	6/6	-	-	-	-	1/1	**14**
SE_gainp/c	5/5	-	-	-	-	-	**10**
SE_shape1p	-	4	-	-	-	-	**4**
SE_shape1c	-	-	-	4	-	-	**4**
SE_shape2p	-	-	4	-	-	-	**4**
SE_shape2c	-	-	-	-	4	-	**4**
total	**44**	**4**	**4**	**4**	**4**	**20**	**80**

The Table 1.B.5 shows protected/unprotected(class I…VI) bit assignment in the case where both previous and current frames are unvoiced.

Table 1.B.5 — Number of protected/unprotected bits at 3.5 kbit/s (unvoiced frame)

para-meters	unvoiced frame						
	class I bits	class II bits	class III bits	class IV bits	Class V bits	class VI bits	total
LSP1p/c	5/5	-	-	-	-	-	**10**
LSP2p/c	4/4	-	-	-	-	3/3	**14**
LSP3p/c	2/2	-	-	-	-	3/3	**10**
LSP4p/c	1/1	-	-	-	-	-	**2**
VUVp/c	2/2	-	-	-	-	-	**4**
VX_gain1[0]p/c	4/4	-	-	-	-	-	**8**
VX_gain1[1]p/c	4/4	-	-	-	-	-	**8**
VX_shape1[0]p/	-	-	-	-	-	6/6	**12**
VX_shape1[1]p/	-	-	-	-	-	6/6	**12**
total	**44**	**0**	**0**	**0**	**0**	**36**	**80**

When the previous frame is unvoiced and the current frame is voiced, or when the previous frame is voiced and the current frame is unvoiced, the same protected/unprotected bit assignment rules as shown above are used.

For 6.2 kbit/s mode, 7 classes are used. CRC check is applied for class I, II, III, IV, V, and VI bits. Class VII bits are not checked by CRC.

The Table 1.B.6 shows protected/unprotected(class I…VII) bit assignment in the case where both previous and current frames are voiced.

Table 1.B.6 — Number of protected/unprotected bits at 6.2 kbit/s (voiced sound)

Para-meters	voiced sound							
	class I bits	class II bits	class III bits	class IV bits	class V bits	class VI bits	class VII bits	total
LSP1p/c	5/5	-	-	-	-	-	-	**10**
LSP2p/c	4/4	-	-	-	-	-	3/3	**14**
LSP3p/c	1/1	-	-	-	-	-	4/4	**10**
LSP4p/c	1/1	-	-	-	-	-	-	**2**
LSP5p/c	1/1	-	-	-	-	-	7/7	**16**
VUVp/c	2/2	-	-	-	-	-	-	**4**
Pitchp/c	6/6	-	-	-	-	-	1/1	**14**
SE_gainp/c	5/5	-	-	-	-	-	-	**10**
SE_shape1p	-	-	4	-	-	-	-	**4**
SE_shape1c	-	-	-	-	4	-	-	**4**
SE_shape2p	-	-	-	4	-	-	-	**4**
SE_shape2c	-	-	-	-	-	4	-	**4**
SE_shape3p/c	5/5	-	-	-	-	-	2/2	**14**
SE_shape4p/c	1/1	9/9	-	-	-	-	-	**20**
SE_shape5p/c	1/1	8/8	-	-	-	-	-	**18**
SE_shape6p/c	1/1	5/5	-	-	-	-	-	**12**
Total	**66**	**44**	**4**	**4**	**4**	**4**	**34**	**160**

The Table 1.B.7 below shows protected/unprotected(class I…VII) bit assignment in the case where both previous and current frames are unvoiced.

Table 1.B.7 — Number of protected/unprotected bits at 6.2 kbit/s (unvoiced sound)

Para-meters	**unvoiced sound**							
	class I bits	**class II bits**	**class III bits**	**class IV bits**	**class V bits**	**class VI bits**	**class VII bits**	**total**
LSP1p/c	5/5	-	-	-	-	-	-	**10**
LSP2p/c	4/4	-	-	-	-	-	3/3	**14**
LSP3p/c	1/1	-	-	-	-	-	4/4	**10**
LSP4p/c	1/1	-	-	-	-	-	-	**2**
LSP5p/c	1/1	-	-	-	-	-	7/7	**16**
VUVp/c	2/2	-	-	-	-	-	-	**4**
VX_gain1[0]p/c	4/4	-	-	-	-	-	-	**8**
VX_gain1[1]p/c	4/4	-	-	-	-	-	-	**8**
VX_shape1[0]p/	-	-	-	-	-	-	6/6	**12**
VX_shape1[1]p/	-	-	-	-	-	-	6/6	**12**
VX_gain2[0]p/c	3/3	-	-	-	-	-	-	**6**
VX_gain2[1]p/c	3/3	-	-	-	-	-	-	**6**
VX_gain2[2]p/c	3/3	-	-	-	-	-	-	**6**
VX_gain2[3]p/c	2/2	-	-	-	-	-	1/1	**6**
VX_shape2[0]p/	-	-	-	-	-	-	5/5	**10**
VX_shape2[1]p/	-	-	-	-	-	-	5/5	**10**
VX_shape2[2]p/	-	-	-	-	-	-	5/5	**10**
VX_shape2[3]p/	-	-	-	-	-	-	5/5	**10**
total	**66**	**0**	**0**	**0**	**0**	**0**	**94**	**160**

When the previous frame is unvoiced and the current frame is voiced, or when the previous frame is voiced and the current frame is unvoiced, the same protected/unprotected bit assignment rules as shown above are used.

The bit order for UEP input is shown from Table 1.B.8 through Table 1.B.11 (for 2 kbit/s), and from Table 1.B.12 through Table 1.B.15 (for 4 kbit/s). These tables show the bit order for each of the combinations of V/UV condition of 2 frames. For example, if previous frame is voiced and current frame is voiced at 2 kbit/s mode, Table 1.B.8 is used. The bit order is arranged according to the error sensitivity. The column "Bit" denotes the bit index of the parameter. "0" means LSB.

Table 1.B.8 — Bit Order for 2 kbit/s (voiced frame -- voiced frame)

voiced frame – voiced frame								
No.	**Item**	**Bit**	**No.**	**Item**	**Bit**	**No.**	**Item**	**Bit**
Class I Bit			**28**	SE_gainc	1	**54**	SE_shape1c	1
0	VUVp	1	**29**	SE_gainc	0	**55**	SE_shape1c	0
1	VUVp	0	**30**	LSP1c	4	**Class V Bit**		
2	LSP4p	0	**31**	LSP1c	3	**56**	SE_shape2c	3
3	SE_gainp	4	**32**	LSP1c	2	**57**	SE_shape2c	2
4	SE_gainp	3	**33**	LSP1c	1	**58**	SE_shape2c	1
5	SE_gainp	2	**34**	LSP1c	0	**59**	SE_shape2c	0
6	SE_gainp	1	**35**	Pitchc	6	**Class VI Bit**		
7	SE_gainp	0	**36**	Pitchc	5	**60**	LSP2p	4

8	LSP1p	4	**37**	Pitchc	4	**61**	LSP2p	3
9	LSP1p	3	**38**	Pitchc	3	**62**	LSP2p	2
10	LSP1p	2	**39**	Pitchc	2	**63**	LSP2p	1
11	LSP1p	1	**40**	Pitchc	1	**64**	LSP2p	0
12	LSP1p	0	**41**	LSP2c	6	**65**	LSP3p	3
13	Pitchp	6	**42**	LSP3c	4	**66**	LSP3p	2
14	Pitchp	5	**43**	LSP2c	5	**67**	LSP3p	1
15	Pitchp	4	**Class II Bit**			**68**	LSP3p	0
16	Pitchp	3	**44**	SE_shape1p	3	**69**	Pitchp	0
17	Pitchp	2	**45**	SE_shape1p	2	**70**	LSP2c	4
18	Pitchp	1	**46**	SE_shape1p	1	**71**	LSP2c	3
19	LSP2p	6	**47**	SE_shape1p	0	**72**	LSP2c	2
20	LSP3p	4	**Class III Bit**			**73**	LSP2c	1
21	LSP2p	5	**48**	SE_shape2p	3	**74**	LSP2c	0
22	VUVc	1	**49**	SE_shape2p	2	**75**	LSP3c	3
23	VUVc	0	**50**	SE_shape2p	1	**76**	LSP3c	2
24	LSP4c	0	**51**	SE_shape2p	0	**77**	LSP3c	1
25	SE_gainc	4	**Class IV Bit**			**78**	LSP3c	0
26	SE_gainc	3	**52**	SE_shape1c	3	**79**	Pitchc	0
27	SE_gainc	2	**53**	SE_shape1c	2			

Table 1.B.9 — Bit Order for 2 kbit/s (voiced frame – unvoiced frame)

voiced frame – unvoiced frame								
No.	**Item**	**Bit**	**No.**	**Item**	**Bit**	**No.**	**Item**	**Bit**
Class I Bit			**28**	VX_gain1[0]c	0	**54**	LSP2c	0
0	VUVp	1	**29**	VX_gain1[1]c	3	**55**	LSP3c	2
1	VUVp	0	**30**	VX_gain1[1]c	2	**Class V Bit**		
2	LSP4p	0	**31**	VX_gain1[1]c	1	**56**	LSP3c	1
3	SE_gainp	4	**32**	VX_gain1[1]c	0	**57**	LSP3c	0
4	SE_gainp	3	**33**	LSP1c	4	**58**	VX_shape1[0]c	5
5	SE_gainp	2	**34**	LSP1c	3	**59**	VX_shape1[0]c	4
6	SE_gainp	1	**35**	LSP1c	2	**Class VI Bit**		
7	SE_gainp	0	**36**	LSP1c	1	**60**	LSP2p	4
8	LSP1p	4	**37**	LSP1c	0	**61**	LSP2p	3
9	LSP1p	3	**38**	LSP2c	6	**62**	LSP2p	2
10	LSP1p	2	**39**	LSP2c	5	**63**	LSP2p	1
11	LSP1p	1	**40**	LSP2c	4	**64**	LSP2p	0
12	LSP1p	0	**41**	LSP2c	3	**65**	LSP3p	3
13	Pitchp	6	**42**	LSP3c	4	**66**	LSP3p	2
14	Pitchp	5	**43**	LSP3c	3	**67**	LSP3p	1
15	Pitchp	4	**Class II Bit**			**68**	LSP3p	0
16	Pitchp	3	**44**	SE_shape1p	3	**69**	Pitchp	0
17	Pitchp	2	**45**	SE_shape1p	2	**70**	VX_shape1[0]c	3
18	Pitchp	1	**46**	SE_shape1p	1	**71**	VX_shape1[0]c	2
19	LSP2p	6	**47**	SE_shape1p	0	**72**	VX_shape1[0]c	1
20	LSP3p	4	**Class III Bit**			**73**	VX_shape1[0]c	0
21	LSP2p	5	**48**	SE_shape2p	3	**74**	VX_shape1[1]c	5
22	VUVc	1	**49**	SE_shape2p	2	**75**	VX_shape1[1]c	4

23	VUVc	0	**50**	SE_shape2p	1	**76**	VX_shape1[1]c	3
24	LSP4c	0	**51**	SE_shape2p	0	**77**	VX_shape1[1]c	2
25	VX_gain1[0]c	3	**Class IV Bit**			**78**	VX_shape1[1]c	1
26	VX_gain1[0]c	2	**52**	LSP2c	2	**79**	VX_shape1[1]c	0
27	VX_gain1[0]c	1	**53**	LSP2c	1			

Table 1.B.10 — Bit Order for 2 kbit/s (unvoiced frame – voiced frame)

unvoiced frame – unvoiced frame								
No.	**Item**	**Bit**	**No.**	**Item**	**Bit**	**No.**	**Item**	**Bit**
Class I Bit			**28**	SE_gainc	1	**54**	SE_shape1c	1
0	VUVp	1	**29**	SE_gainc	0	**55**	SE_shape1c	0
1	VUVp	0	**30**	LSP1c	4	**Class V Bit**		
2	LSP4p	0	**31**	LSP1c	3	**56**	SE_shape2c	3
3	VX_gain1[0]p	3	**32**	LSP1c	2	**57**	SE_shape2c	2
4	VX_gain1[0]p	2	**33**	LSP1c	1	**58**	SE_shape2c	1
5	VX_gain1[0]p	1	**34**	LSP1c	0	**59**	SE_shape2c	0
6	VX_gain1[0]p	0	**35**	Pitchc	6	**Class VI Bit**		
7	VX_gain1[1]p	3	**36**	Pitchc	5	**60**	VX_shape1[0]p	3
8	VX_gain1[1]p	2	**37**	Pitchc	4	**61**	VX_shape1[0]p	2
9	VX_gain1[1]p	1	**38**	Pitchc	3	**62**	VX_shape1[0]p	1
10	VX_gain1[1]p	0	**39**	Pitchc	2	**63**	VX_shape1[0]p	0
11	LSP1p	4	**40**	Pitchc	1	**64**	VX_shape1[1]p	5
12	LSP1p	3	**41**	LSP2c	6	**65**	VX_shape1[1]p	4
13	LSP1p	2	**42**	LSP3c	4	**66**	VX_shape1[1]p	3
14	LSP1p	1	**43**	LSP2c	5	**67**	VX_shape1[1]p	2
15	LSP1p	0	**Class II Bit**			**68**	VX_shape1[1]p	1
16	LSP2p	6	**44**	LSP2p	2	**69**	VX_shape1[1]p	0
17	LSP2p	5	**45**	LSP2p	1	**70**	LSP2c	4
18	LSP2p	4	**46**	LSP2p	0	**71**	LSP2c	3
19	LSP2p	3	**47**	LSP3p	2	**72**	LSP2c	2
20	LSP3p	4	**Class III Bit**			**73**	LSP2c	1
21	LSP3p	3	**48**	LSP3p	1	**74**	LSP2c	0
22	VUVc	1	**49**	LSP3p	0	**75**	LSP3c	3
23	VUVc	0	**50**	VX_shape1[0]p	5	**76**	LSP3c	2
24	LSP4c	0	**51**	VX_shape1[0]p	4	**77**	LSP3c	1
25	SE_gainc	4	**Class IV Bit**			**78**	LSP3c	0
26	SE_gainc	3	**52**	SE_shape1c	3	**79**	Pitchc	0
27	SE_gainc	2	**53**	SE_shape1c	2			

Table 1.B.11 — Bit Order for 2 kbit/s (unvoiced frame – unvoiced frame)

unvoiced frame – unvoiced frame								
No.	**Item**	**Bit**	**No.**	**Item**	**Bit**	**No.**	**Item**	**Bit**
Class I Bit			**28**	VX_gain1[0]c	0	**54**	LSP2c	0
0	VUVp	1	**29**	VX_gain1[1]c	3	**55**	LSP3c	2
1	VUVp	0	**30**	VX_gain1[1]c	2	**Class V Bit**		
2	LSP4p	0	**31**	VX_gain1[1]c	1	**56**	LSP3c	1

3	VX_gain1[0]p	3	**32**	VX_gain1[1]c	0	**57**	LSP3c	0
4	VX_gain1[0]p	2	**33**	LSP1c	4	**58**	VX_shape1[0]c	5
5	VX_gain1[0]p	1	**34**	LSP1c	3	**59**	VX_shape1[0]c	4
6	VX_gain1[0]p	0	**35**	LSP1c	2	**Class VI Bit**		
7	VX_gain1[1]p	3	**36**	LSP1c	1	**60**	VX_shape1[0]p	3
8	VX_gain1[1]p	2	**37**	LSP1c	0	**61**	VX_shape1[0]p	2
9	VX_gain1[1]p	1	**38**	LSP2c	6	**62**	VX_shape1[0]p	1
10	VX_gain1[1]p	0	**39**	LSP2c	5	**63**	VX_shape1[0]p	0
11	LSP1p	4	**40**	LSP2c	4	**64**	VX_shape1[1]p	5
12	LSP1p	3	**41**	LSP2c	3	**65**	VX_shape1[1]p	4
13	LSP1p	2	**42**	LSP3c	4	**66**	VX_shape1[1]p	3
14	LSP1p	1	**43**	LSP3c	3	**67**	VX_shape1[1]p	2
15	LSP1p	0	**Class II Bit**			**68**	VX_shape1[1]p	1
16	LSP2p	6	**44**	LSP2p	2	**69**	VX_shape1[1]p	0
17	LSP2p	5	**45**	LSP2p	1	**70**	VX_shape1[0]c	3
18	LSP2p	4	**46**	LSP2p	0	**71**	VX_shape1[0]c	2
19	LSP2p	3	**47**	LSP3p	2	**72**	VX_shape1[0]c	1
20	LSP3p	4	**Class III Bit**			**73**	VX_shape1[0]c	0
21	LSP3p	3	**48**	LSP3p	1	**74**	VX_shape1[1]c	5
22	VUVc	1	**49**	LSP3p	0	**75**	VX_shape1[1]c	4
23	VUVc	0	**50**	VX_shape1[0]p	5	**76**	VX_shape1[1]c	3
24	LSP4c	0	**51**	VX_shape1[0]p	4	**77**	VX_shape1[1]c	2
25	VX_gain1[0]c	3	**Class IV Bit**			**78**	VX_shape1[1]c	1
26	VX_gain1[0]c	2	**52**	LSP2c	2	**79**	VX_shape1[1]c	0
27	VX_gain1[0]c	1	**53**	LSP2c	1			

Table 1.B.12 — Bit Order for 4 kbit/s (voiced frame – voiced frame)

voiced frame – voiced frame								
No.	**Item**	**Bit**	**No.**	**Item**	**Bit**	**No.**	**Item**	**Bit**
Class 1bit			**55**	LSP2c	3	**Class III Bit**		
0	VUVp	1	**56**	SE_shape3c	6	**110**	SE_shape1p	3
1	VUVp	0	**57**	SE_shape3c	5	**111**	SE_shape1p	2
2	LSP4p	0	**58**	SE_shape3c	4	**112**	SE_shape1p	1
3	SE_gainp	4	**59**	SE_shape3c	3	**113**	SE_shape1p	0
4	SE_gainp	3	**60**	SE_shape3c	2	**Class IV Bit**		
5	SE_gainp	2	**61**	LSP3c	4	**114**	SE_shape2p	3
6	SE_gainp	1	**62**	LSP5c	7	**115**	SE_shape2p	2
7	SE_gainp	0	**63**	SE_shape4c	9	**116**	SE_shape2p	1
8	LSP1p	4	**64**	SE_shape5c	8	**117**	SE_shape2p	0
9	LSP1p	3	**65**	SE_shape6c	5	**Class V Bit**		
10	LSP1p	2	**Class II Bit**			**118**	SE_shape1c	3
11	LSP1p	1	**66**	SE_shape4p	8	**119**	SE_shape1c	2
12	LSP1p	0	**67**	SE_shape4p	7	**120**	SE_shape1c	1
13	Pitchp	6	**68**	SE_shape4p	6	**121**	SE_shape1c	0
14	Pitchp	5	**69**	SE_shape4p	5	**Class VI Bit**		
15	Pitchp	4	**70**	SE_shape4p	4	**122**	SE_shape2c	3
16	Pitchp	3	**71**	SE_shape4p	3	**123**	SE_shape2c	2

17	Pitchp	2	**72**	SE_shape4p	2	**124**	SE_shape2c	1
18	Pitchp	1	**73**	SE_shape4p	1	**125**	SE_shape2c	0
19	LSP2p	6	**74**	SE_shape4p	0	**Class VII Bit**		
20	LSP2p	5	**75**	SE_shape5p	7	**126**	LSP2p	2
21	LSP2p	4	**76**	SE_shape5p	6	**127**	LSP2p	1
22	LSP2p	3	**77**	SE_shape5p	5	**128**	LSP2p	0
23	SE_shape3p	6	**78**	SE_shape5p	4	**129**	LSP3p	3
24	SE_shape3p	5	**79**	SE_shape5p	3	**130**	LSP3p	2
25	SE_shape3p	4	**80**	SE_shape5p	2	**131**	LSP3p	1
26	SE_shape3p	3	**81**	SE_shape5p	1	**132**	LSP3p	0
27	SE_shape3p	2	**82**	SE_shape5p	0	**133**	LSP5p	6
28	LSP3p	4	**83**	SE_shape6p	4	**134**	LSP5p	5
29	LSP5p	7	**84**	SE_shape6p	3	**135**	LSP5p	4
30	SE_shape4p	9	**85**	SE_shape6p	2	**136**	LSP5p	3
31	SE_shape5p	8	**86**	SE_shape6p	1	**137**	LSP5p	2
32	SE_shape6p	5	**87**	SE_shape6p	0	**138**	LSP5p	1
33	VUVc	1	**88**	SE_shape4c	8	**139**	LSP5p	0
34	VUVc	0	**89**	SE_shape4c	7	**140**	Pitchp	0
35	LSP4c	0	**90**	SE_shape4c	6	**141**	SE_shape3p	1
36	SE_gainc	4	**91**	SE_shape4c	5	**142**	SE_shape3p	0
37	SE_gainc	3	**92**	SE_shape4c	4	**143**	LSP2c	2
38	SE_gainc	2	**93**	SE_shape4c	3	**144**	LSP2c	1
39	SE_gainc	1	**94**	SE_shape4c	2	**145**	LSP2c	0
40	SE_gainc	0	**95**	SE_shape4c	1	**146**	LSP3c	3
41	LSP1c	4	**96**	SE_shape4c	0	**147**	LSP3c	2
42	LSP1c	3	**97**	SE_shape5c	7	**148**	LSP3c	1
43	LSP1c	2	**98**	SE_shape5c	6	**149**	LSP3c	0
44	LSP1c	1	**99**	SE_shape5c	5	**150**	LSP5c	6
45	LSP1c	0	**100**	SE_shape5c	4	**151**	LSP5c	5
46	Pitchc	6	**101**	SE_shape5c	3	**152**	LSP5c	4
47	Pitchc	5	**102**	SE_shape5c	2	**153**	LSP5c	3
48	Pitchc	4	**103**	SE_shape5c	1	**154**	LSP5c	2
49	Pitchc	3	**104**	SE_shape5c	0	**155**	LSP5c	1
50	Pitchc	2	**105**	SE_shape6c	4	**156**	LSP5c	0
51	Pitchc	1	**106**	SE_shape6c	3	**157**	Pitchc	0
52	LSP2c	6	**107**	SE_shape6c	2	**158**	SE_shape3c	1
53	LSP2c	5	**108**	SE_shape6c	1	**159**	SE_shape3c	0
54	LSP2c	4	**109**	SE_shape6c	0			

Table 1.B.13 — Bit Order for 4 kbit/s (voiced frame – unvoiced frame)

voiced frame – unvoiced frame								
No.	**Item**	**Bit**	**No.**	**Item**	**Bit**	**No.**	**Item**	**Bit**
Class 1bit			**55**	VX_gain2[0]c	2	**Class III Bit**		
0	VUVp	1	**56**	VX_gain2[0]c	1	**110**	SE_shape1p	3
1	VUVp	0	**57**	VX_gain2[0]c	0	**111**	SE_shape1p	2
2	LSP4p	0	**58**	VX_gain2[1]c	2	**112**	SE_shape1p	1
3	SE_gainp	4	**59**	VX_gain2[1]c	1	**113**	SE_shape1p	0

4	SE_gainp	3	**60**	VX_gain2[1]c	0	**Class IV Bit**		
5	SE_gainp	2	**61**	VX_gain2[2]c	2	**114**	SE_shape2p	3
6	SE_gainp	1	**62**	VX_gain2[2]c	1	**115**	SE_shape2p	2
7	SE_gainp	0	**63**	VX_gain2[2]c	0	**116**	SE_shape2p	1
8	LSP1p	4	**64**	VX_gain2[3]c	2	**117**	SE_shape2p	0
9	LSP1p	3	**65**	VX_gain2[3]c	1	**Class V Bit**		
10	LSP1p	2	**Class II Bit**			**118**	VX_shape1[1]c	4
11	LSP1p	1	**66**	SE_shape4p	8	**119**	VX_shape1[1]c	3
12	LSP1p	0	**67**	SE_shape4p	7	**120**	VX_shape1[1]c	2
13	Pitchp	6	**68**	SE_shape4p	6	**121**	VX_shape1[1]c	1
14	Pitchp	5	**69**	SE_shape4p	5	**Class VI Bit**		
15	Pitchp	4	**70**	SE_shape4p	4	**122**	VX_shape1[1]c	0
16	Pitchp	3	**71**	SE_shape4p	3	**123**	VX_shape2[0]c	4
17	Pitchp	2	**72**	SE_shape4p	2	**124**	VX_shape2[0]c	3
18	Pitchp	1	**73**	SE_shape4p	1	**125**	VX_shape2[0]c	2
19	LSP2p	6	**74**	SE_shape4p	0	**Class VII Bit**		
20	LSP2p	5	**75**	SE_shape5p	7	**126**	LSP2p	2
21	LSP2p	4	**76**	SE_shape5p	6	**127**	LSP2p	1
22	LSP2p	3	**77**	SE_shape5p	5	**128**	LSP2p	0
23	SE_shape3p	6	**78**	SE_shape5p	4	**129**	LSP3p	3
24	SE_shape3p	5	**79**	SE_shape5p	3	**130**	LSP3p	2
25	SE_shape3p	4	**80**	SE_shape5p	2	**131**	LSP3p	1
26	SE_shape3p	3	**81**	SE_shape5p	1	**132**	LSP3p	0
27	SE_shape3p	2	**82**	SE_shape5p	0	**133**	LSP5p	6
28	LSP3p	4	**83**	SE_shape6p	4	**134**	LSP5p	5
29	LSP5p	7	**84**	SE_shape6p	3	**135**	LSP5p	4
30	SE_shape4p	9	**85**	SE_shape6p	2	**136**	LSP5p	3
31	SE_shape5p	8	**86**	SE_shape6p	1	**137**	LSP5p	2
32	SE_shape6p	5	**87**	SE_shape6p	0	**138**	LSP5p	1
33	VUVc	1	**88**	VX_gain2[3]c	0	**139**	LSP5p	0
34	VUVc	0	**89**	LSP2c	2	**140**	Pitchp	0
35	LSP4c	0	**90**	LSP2c	1	**141**	SE_shape3p	1
36	VX_gain1[0]c	3	**91**	LSP2c	0	**142**	SE_shape3p	0
37	VX_gain1[0]c	2	**92**	LSP3c	3	**143**	VX_shape2[0]c	1
38	VX_gain1[0]c	1	**93**	LSP3c	2	**144**	VX_shape2[0]c	0
39	VX_gain1[0]c	0	**94**	LSP3c	1	**145**	VX_shape2[1]c	4
40	VX_gain1[1]c	3	**95**	LSP3c	0	**146**	VX_shape2[1]c	3
41	VX_gain1[1]c	2	**96**	LSP5c	6	**147**	VX_shape2[1]c	2
42	VX_gain1[1]c	1	**97**	LSP5c	5	**148**	VX_shape2[1]c	1
43	VX_gain1[1]c	0	**98**	LSP5c	4	**149**	VX_shape2[1]c	0
44	LSP1c	4	**99**	LSP5c	3	**150**	VX_shape2[2]c	4
45	LSP1c	3	**100**	LSP5c	2	**151**	VX_shape2[2]c	3
46	LSP1c	2	**101**	LSP5c	1	**152**	VX_shape2[2]c	2
47	LSP1c	1	**102**	LSP5c	0	**153**	VX_shape2[2]c	1
48	LSP1c	0	**103**	VX_shape1[0]c	5	**154**	VX_shape2[2]c	0
49	LSP2c	6	**104**	VX_shape1[0]c	4	**155**	VX_shape2[3]c	4
50	LSP2c	5	**105**	VX_shape1[0]c	3	**156**	VX_shape2[3]c	3
51	LSP2c	4	**106**	VX_shape1[0]c	2	**157**	VX_shape2[3]c	2
52	LSP2c	3	**107**	VX_shape1[0]c	1	**158**	VX_shape2[3]c	1

53	LSP3c	4	**108**	VX_shape1[0]c	0	**159**	VX_shape2[3]c	0
54	LSP5c	7	**109**	VX_shape1[1]c	5			

Table 1.B.14 — Bit Order for 4 kbit/s (unvoiced frame – voiced frame)

unvoiced frame – voiced frame								
No.	**Item**	**Bit**	**No.**	**Item**	**Bit**	**No.**	**Item**	**Bit**
Class 1bit			**55**	LSP2c	3	**Class III Bit**		
0	VUVp	1	**56**	SE_shape3c	6	**110**	VX_shape1[1]p	4
1	VUVp	0	**57**	SE_shape3c	5	**111**	VX_shape1[1]p	3
2	LSP4p	0	**58**	SE_shape3c	4	**112**	VX_shape1[1]p	2
3	VX_gain1[0]p	3	**59**	SE_shape3c	3	**113**	VX_shape1[1]p	1
4	VX_gain1[0]p	2	**60**	SE_shape3c	2	**Class IV Bit**		
5	VX_gain1[0]p	1	**61**	LSP3c	4	**114**	VX_shape1[1]p	0
6	VX_gain1[0]p	0	**62**	LSP5c	7	**115**	VX_shape2[0]p	4
7	VX_gain1[1]p	3	**63**	SE_shape4c	9	**116**	VX_shape2[0]p	3
8	VX_gain1[1]p	2	**64**	SE_shape5c	8	**117**	VX_shape2[0]p	2
9	VX_gain1[1]p	1	**65**	SE_shape6c	5	**Class V Bit**		
10	VX_gain1[1]p	0	**Class II Bit**			**118**	SE_shape1c	3
11	LSP1p	4	**66**	VX_gain2[3]p	0	**119**	SE_shape1c	2
12	LSP1p	3	**67**	LSP2p	2	**120**	SE_shape1c	1
13	LSP1p	2	**68**	LSP2p	1	**121**	SE_shape1c	0
14	LSP1p	1	**69**	LSP2p	0	**Class VI Bit**		
15	LSP1p	0	**70**	LSP3p	3	**122**	SE_shape2c	3
16	LSP2p	6	**71**	LSP3p	2	**123**	SE_shape2c	2
17	LSP2p	5	**72**	LSP3p	1	**124**	SE_shape2c	1
18	LSP2p	4	**73**	LSP3p	0	**125**	SE_shape2c	0
19	LSP2p	3	**74**	LSP5p	6	**Class VII Bit**		
20	LSP3p	4	**75**	LSP5p	5	**126**	VX_shape2[0]p	1
21	LSP5p	7	**76**	LSP5p	4	**127**	VX_shape2[0]p	0
22	VX_gain2[0]p	2	**77**	LSP5p	3	**128**	VX_shape2[1]p	4
23	VX_gain2[0]p	1	**78**	LSP5p	2	**129**	VX_shape2[1]p	3
24	VX_gain2[0]p	0	**79**	LSP5p	1	**130**	VX_shape2[1]p	2
25	VX_gain2[1]p	2	**80**	LSP5p	0	**131**	VX_shape2[1]p	1
26	VX_gain2[1]p	1	**81**	VX_shape1[0]p	5	**132**	VX_shape2[1]p	0
27	VX_gain2[1]p	0	**82**	VX_shape1[0]p	4	**133**	VX_shape2[2]p	4
28	VX_gain2[2]p	2	**83**	VX_shape1[0]p	3	**134**	VX_shape2[2]p	3
29	VX_gain2[2]p	1	**84**	VX_shape1[0]p	2	**135**	VX_shape2[2]p	2
30	VX_gain2[2]p	0	**85**	VX_shape1[0]p	1	**136**	VX_shape2[2]p	1
31	VX_gain2[3]p	2	**86**	VX_shape1[0]p	0	**137**	VX_shape2[2]p	0
32	VX_gain2[3]p	1	**87**	VX_shape1[1]p	5	**138**	VX_shape2[3]p	4
33	VUVc	1	**88**	SE_shape4c	8	**139**	VX_shape2[3]p	3
34	VUVc	0	**89**	SE_shape4c	7	**140**	VX_shape2[3]p	2
35	LSP4c	0	**90**	SE_shape4c	6	**141**	VX_shape2[3]p	1
36	SE_gainc	4	**91**	SE_shape4c	5	**142**	VX_shape2[3]p	0
37	SE_gainc	3	**92**	SE_shape4c	4	**143**	LSP2c	2
38	SE_gainc	2	**93**	SE_shape4c	3	**144**	LSP2c	1
39	SE_gainc	1	**94**	SE_shape4c	2	**145**	LSP2c	0

40	SE_gainc	0	**95**	SE_shape4c	1	**146**	LSP3c	3
41	LSP1c	4	**96**	SE_shape4c	0	**147**	LSP3c	2
42	LSP1c	3	**97**	SE_shape5c	7	**148**	LSP3c	1
43	LSP1c	2	**98**	SE_shape5c	6	**149**	LSP3c	0
44	LSP1c	1	**99**	SE_shape5c	5	**150**	LSP5c	6
45	LSP1c	0	**100**	SE_shape5c	4	**151**	LSP5c	5
46	Pitchc	6	**101**	SE_shape5c	3	**152**	LSP5c	4
47	Pitchc	5	**102**	SE_shape5c	2	**153**	LSP5c	3
48	Pitchc	4	**103**	SE_shape5c	1	**154**	LSP5c	2
49	Pitchc	3	**104**	SE_shape5c	0	**155**	LSP5c	1
50	Pitchc	2	**105**	SE_shape6c	4	**156**	LSP5c	0
51	Pitchc	1	**106**	SE_shape6c	3	**157**	Pitchc	0
52	LSP2c	6	**107**	SE_shape6c	2	**158**	SE_shape3c	1
53	LSP2c	5	**108**	SE_shape6c	1	**159**	SE_shape3c	0
54	LSP2c	4	**109**	SE_shape6c	0			

Table 1.B.15 — Bit Order for 4 kbit/s (unvoiced frame – unvoiced frame)

unvoiced frame – unvoiced frame								
No.	**Item**	**Bit**	**No.**	**Item**	**Bit**	**No.**	**Item**	**Bit**
Class 1bit			**55**	VX_gain2[0]c	2	**Class III Bit**		
0	VUVp	1	**56**	VX_gain2[0]c	1	**110**	VX_shape1[1]p	4
1	VUVp	0	**57**	VX_gain2[0]c	0	**111**	VX_shape1[1]p	3
2	LSP4p	0	**58**	VX_gain2[1]c	2	**112**	VX_shape1[1]p	2
3	VX_gain1[0]p	3	**59**	VX_gain2[1]c	1	**113**	VX_shape1[1]p	1
4	VX_gain1[0]p	2	**60**	VX_gain2[1]c	0	**Class IV Bit**		
5	VX_gain1[0]p	1	**61**	VX_gain2[2]c	2	**114**	VX_shape1[1]p	0
6	VX_gain1[0]p	0	**62**	VX_gain2[2]c	1	**115**	VX_shape2[0]p	4
7	VX_gain1[1]p	3	**63**	VX_gain2[2]c	0	**116**	VX_shape2[0]p	3
8	VX_gain1[1]p	2	**64**	VX_gain2[3]c	2	**117**	VX_shape2[0]p	2
9	VX_gain1[1]p	1	**65**	VX_gain2[3]c	1	**Class V Bit**		
10	VX_gain1[1]p	0	**Class II Bit**			**118**	VX_shape1[1]c	4
11	LSP1p	4	**66**	VX_gain2[3]p	0	**119**	VX_shape1[1]c	3
12	LSP1p	3	**67**	LSP2p	2	**120**	VX_shape1[1]c	2
13	LSP1p	2	**68**	LSP2p	1	**121**	VX_shape1[1]c	1
14	LSP1p	1	**69**	LSP2p	0	**Class VI Bit**		
15	LSP1p	0	**70**	LSP3p	3	**122**	VX_shape1[1]c	0
16	LSP2p	6	**71**	LSP3p	2	**123**	VX_shape2[0]c	4
17	LSP2p	5	**72**	LSP3p	1	**124**	VX_shape2[0]c	3
18	LSP2p	4	**73**	LSP3p	0	**125**	VX_shape2[0]c	2
19	LSP2p	3	**74**	LSP5p	6	**Class VII Bit**		
20	LSP3p	4	**75**	LSP5p	5	**126**	VX_shape2[0]p	1
21	LSP5p	7	**76**	LSP5p	4	**127**	VX_shape2[0]p	0
22	VX_gain2[0]p	2	**77**	LSP5p	3	**128**	VX_shape2[1]p	4
23	VX_gain2[0]p	1	**78**	LSP5p	2	**129**	VX_shape2[1]p	3
24	VX_gain2[0]p	0	**79**	LSP5p	1	**130**	VX_shape2[1]p	2
25	VX_gain2[1]p	2	**80**	LSP5p	0	**131**	VX_shape2[1]p	1
26	VX_gain2[1]p	1	**81**	VX_shape1[0]p	5	**132**	VX_shape2[1]p	0

27	VX_gain2[1]p	0	**82**	VX_shape1[0]p	4	**133**	VX_shape2[2]p	4
28	VX_gain2[2]p	2	**83**	VX_shape1[0]p	3	**134**	VX_shape2[2]p	3
29	VX_gain2[2]p	1	**84**	VX_shape1[0]p	2	**135**	VX_shape2[2]p	2
30	VX_gain2[2]p	0	**85**	VX_shape1[0]p	1	**136**	VX_shape2[2]p	1
31	VX_gain2[3]p	2	**86**	VX_shape1[0]p	0	**137**	VX_shape2[2]p	0
32	VX_gain2[3]p	1	**87**	VX_shape1[1]p	5	**138**	VX_shape2[3]p	4
33	VUVc	1	**88**	VX_gain2[3]c	0	**139**	VX_shape2[3]p	3
34	VUVc	0	**89**	LSP2c	2	**140**	VX_shape2[3]p	2
35	LSP4c	0	**90**	LSP2c	1	**141**	VX_shape2[3]p	1
36	VX_gain1[0]c	3	**91**	LSP2c	0	**142**	VX_shape2[3]p	0
37	VX_gain1[0]c	2	**92**	LSP3c	3	**143**	VX_shape2[0]c	1
38	VX_gain1[0]c	1	**93**	LSP3c	2	**144**	VX_shape2[0]c	0
39	VX_gain1[0]c	0	**94**	LSP3c	1	**145**	VX_shape2[1]c	4
40	VX_gain1[1]c	3	**95**	LSP3c	0	**146**	VX_shape2[1]c	3
41	VX_gain1[1]c	2	**96**	LSP5c	6	**147**	VX_shape2[1]c	2
42	VX_gain1[1]c	1	**97**	LSP5c	5	**148**	VX_shape2[1]c	1
43	VX_gain1[1]c	0	**98**	LSP5c	4	**149**	VX_shape2[1]c	0
44	LSP1c	4	**99**	LSP5c	3	**150**	VX_shape2[2]c	4
45	LSP1c	3	**100**	LSP5c	2	**151**	VX_shape2[2]c	3
46	LSP1c	2	**101**	LSP5c	1	**152**	VX_shape2[2]c	2
47	LSP1c	1	**102**	LSP5c	0	**153**	VX_shape2[2]c	1
48	LSP1c	0	**103**	VX_shape1[0]c	5	**154**	VX_shape2[2]c	0
49	LSP2c	6	**104**	VX_shape1[0]c	4	**155**	VX_shape2[3]c	4
50	LSP2c	5	**105**	VX_shape1[0]c	3	**156**	VX_shape2[3]c	3
51	LSP2c	4	**106**	VX_shape1[0]c	2	**157**	VX_shape2[3]c	2
52	LSP2c	3	**107**	VX_shape1[0]c	1	**158**	VX_shape2[3]c	1
53	LSP3c	4	**108**	VX_shape1[0]c	0	**159**	VX_shape2[3]c	0
54	LSP5c	7	**109**	VX_shape1[1]c	5			

1.B.3.3 EP tool setting

1.B.3.3.1 Bit assignment

The Table 1.B.16 below shows an example bit assignment for the use of the EP tool. In this table, bit assignments for both of the 2 kbit/s and 4 kbit/s source coder are described.

Table 1.B.16 — The bit assignment for the use of the EP tool

	2 kbit/s Source Coder	4 kbit/s Source Coder
Class I		
Source coder bits	44	66
CRC parity	6	6
Code Rate	8/16	8/16
Class I total	100	144
Class II		
Source coder bits	4	44
CRC parity	1	6
Code Rate	8/8	8/8
Class II total	5	50
Class III		
Source coder bits	4	4
CRC parity	1	1
Code Rate	8/8	8/8
Class III total	5	5
Class IV		
Source coder bits	4	4
CRC parity	1	1
Code Rate	8/8	8/8
Class IV total	5	5
Class V		
Source coder bits	4	4
CRC parity	1	1
Code Rate	8/8	8/8
Class V total	5	5
Class VI		
Source coder bits	20	4
CRC parity	0	1
Code Rate	8/8	8/8
Class VI total	20	5
Class VII		
Source coder bits		34
CRC parity		0
Code Rate		8/8
Class VII total		34
Total Bit of All Classes	140	248
Bitrate	3.5 kbit/s	6.2 kbit/s

Class I:
CRC covers all the Class I bits, and Class I bits including CRC are protected by convolutional coding.
Class II-V(2 kbit/s), II-VI(4 kbit/s):
At least one CRC bits cover the source coder bits of these classes.
Class VI(2 kbit/s), VII(4 kbit/s) :
The source coder bits are not checked by CRC nor protected by any error correction scheme.

1.B.3.4 Error concealment

When CRC error is detected, error concealment processing (bad frame masking) is carried out. An example of concealment method is described below.

A frame masking state of the current frame is updated based on the decoded CRC result of Class I. The state transition diagram is shown in Figure 1.B.2. The initial state is state = 0. The arrow with a letter “1” denotes the transition for a bad frame, and that with a letter “0” a good frame.

1.B.3.4.1 Parameter replacemet

According to the state value, the following parameter replacement is done. In error free condition, state value becomes 0, and received source coder bits are used without any concealment processing.

1.B.3.4.1.1 LSP parameters

At state=1..6, LSP parameters are replaced with those of previous ones.

When state=7, If LSP4=0 (LSP quantization mode without inter-frame prediction), then LSP parameters are calculated from all LSP indices received in the current frame. If LSP4=1 (LSP quantization mode with inter-frame coding), then LSP parameters are calculated with the following method.

In this mode, LSP parameters from LSP1 index are interpolated with the previous LSPs.

$$LSP_{base}(n) = p \cdot LSP_{prev}(n) + (1-p)LSP_{1st}(n) \qquad \text{for n=1..10} \quad (1)$$

$LSP_{base}(n)$ is LSP parameters of the base layer, $LSP_{prev}(n)$ is the previous LSPs, $LSP_{1st}(n)$ is the decoded LSPs from the current LSP1 index, and p is the factor of interpolation. p is changed according to the number of previous CRC error frames of Class I bits as shown in Table 1.B.17. LSP indices LSP2, LSP3 and LSP5 are not used and $LSP_{base}(n)$ is used as current LSP parameters.

Table 1.B.17 — p factor

frame	p
0	0.7
1	0.6
2	0.5
3	0.4
4	0.3
5	0.2
6	0.1
≥7	0.0

1.B.3.4.1.2 Mute variable

According to the “state” value, a variable “mute” is set to control output level of speech.
The “mute” value below is used.
In state = 7, the average of 1.0 and “mute” value of the previous frame (= 0.5 (1.0 + previous “mute value”)) is used, but when this value is more than 0.8, “mute” value is replaced with 0.8.

Table 1.B.18 — mute value

State	mute
0	1.000
1	0.800
2	0.700
3	0.500
4	0.250
5	0.125
6	0.000
7	Average/0.800

1.B.3.4.1.3 Replacement and gain control of "voiced" parameters

In state=1..6, spectrum parameter SE_shape1, SE_shape2, spectrum gain parameter SE_gain, spectrum parameter for 4 kbit/s codec SE_shape3 .. SE_shape6 are replaced with corresponding parameters of the previous frame. Also, to control volume of output speech, harmonic magnitude parameters of LPC residual signal "$Am[0\ldots127]$" is gain controlled as shown in Eq.(1). In the equation, $Am_{(org)}[i]$ is computed from the received spectrum parameters from the latest error free frame.

$$Am[i] = mute * Am_{(org)}[i] \quad \text{for i=0..127} \qquad (1)$$

If previous frame is unvoiced and current state is state=7, Eq.(1) is replaced with Eq.(2).

$$Am[i] = 0.6 * mute * Am_{(org)}[i] \quad \text{for i=0..127} \quad (2)$$

As described before, SE_shape1 and SE_shape2 are individually protected by 1 bit CRC. In state=0 or 7, when CRC errors of these classes are detected at the same time, the quantized harmonic magnitudes with fixed dimension $Am_{qnt}[1..44]$ are gain suppressed as shown in Eq.(3).

$$Am_{qnt}[i] = s[i] * Am_{qnt(org)}[i] \quad \text{for i=1..44} \qquad (3)$$

$s[i]$ is the factor for the gain suppression.

Table 1.B.19 — factor for gain suppression 's[0..44]'

i	1	2	3	4	5	6	7..44
$s[i]$	0.10	0.25	0.40	0.55	0.70	0.85	1.00

At 4 kbit/s, SE_shape4, SE_shape5, and SE_shape6 are checked by CRC as Class II bits. When CRC error is detected, the spectrum parameter of the enhancement layer is not used.

1.B.3.4.1.4 Replacement and gain control of "unvoiced" parameters

In state=1..6, stochastic codebook gain parameter VX_gain1[0], VX_gain1[1] are replaced with the VX_gain1[1] from the latest error free frame. Also stochastic codebook gain parameter for 4 kbit/s codec VX_gain2[0]..VX_gain2[3] are replaced with the VX_gain2[3] from the latest error free frame.
Stochastic codebook shape parameter VX_shape1[0], VX_shape1[1],and stochastic codebook shape parameter for 4 kbit/s codec are generated from randomly generated index values.
Also, to control volume of output speech, LPC residual signal $res[0\ldots159]$ is gain controlled as shown in Eq.(4). In the equation, $res_{(org)}[i]$ is computed from stochastic codebook parameters.

$$res[i] = mute * res_{(org)}[i] \qquad (0 \le i \le 159) \qquad (4)$$

1.B.3.4.1.5 Frame Masking State Transitions

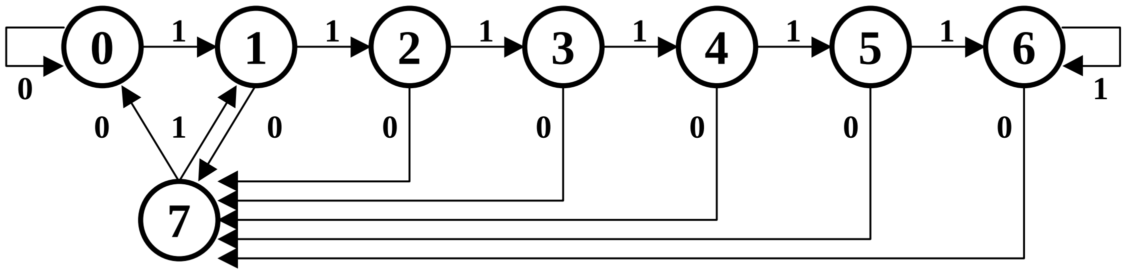


Figure 1.B.2 — Frame Masking State Transitions

Annex 1.C
(informative)

Patent statements

The International Organization for Standardization and the International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this part of ISO/IEC 14496 may involve the use of patents.

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured the ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patents right are registered with ISO and IEC. Information may be obtained from the companies listed in the Table below.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 14496 may be the subject of patent rights other than those identified in this annex. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Alcatel
AT&T
BBC
Bosch
British Telecommunications
Canon
CCETT
Coding Technologies
Columbia Innovation Enterprise
Columbia University
Creative
CSELT
DemoGraFX
DirecTV
Dolby
EPFL
ETRI
France Telecom
Fraunhofer
Fujitsu
GC Technology Corporation
Hitachi
Hyundai
IBM
Institut fuer Rundfunktechnik
JVC
KPN
Matsushita Electric Industrial Co., Ltd.
Microsoft
Mitsubishi
NEC
NHK
Nokia
NTT

NTT Mobile Communication Networks
OKI
Philips
Rockwell
Samsung
Sarnoff
Scientific Atlanta
Sharp
Siemens
Sony
Telenor
Thomson