INTERNATIONAL STANDARD ISO/IEC 14496-3:2001
TECHNICAL CORRIGENDUM 1

Published 2002-12-01

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION
INTERNATIONAL ELECTROTECHNICAL COMMISSION • МЕЖДУНАРОДНАЯ ЭЛЕКТРОТЕХНИЧЕСКАЯ КОМИССИЯ • COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

Information technology — Coding of audio-visual objects —

Part 3:
Audio

TECHNICAL CORRIGENDUM 1

Technologies de l'information — Codage des objets audiovisuels —

Partie 3: Codage audio

RECTIFICATIF TECHNIQUE 1

Technical Corrigendum 1 to ISO/IEC 14496-3:2001 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information.*

ICS 35.040 **Ref. No. ISO/IEC 14496-3:2001/Cor.1:2002(E)**

Published in Switzerland

Information technology — Coding of audio-visual objects —

Part 3:
Audio

TECHNICAL CORRIGENDUM 1

Throughout the text, replace the term

„
bit stream
”

with

„
bitstream
”.

Throughout the text replace

„
scaleable
”

with

„
scalable
”.

Throughout the text replace

„
indexes
”

with

„
indices
”.

In clause (Introduction), replace

”

MPEG-4 has no standard for transport. In all of the MPEG-4 tools for audio and visual coding, the coding standard ends at the point of constructing a sequence of access units that contain the compressed data. The MPEG-4 Systems (ISO/IEC 14496-1:2001) specification describes how to convert the individually coded objects into a bitstream that contains a number of multiplexed sub-streams.

There is no standard mechanism for transport of this stream 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 14496-6:1999) 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 bitstream specification allows transmission functions that are much more sophisticated than are possible with previous MPEG standards.
”

with

”

MPEG-4 has no standard for transport. 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.
”.

In clause (Introduction), replace multiplex, storage and transmission formats table with:

”

	Format	Functionality defined in MPEG-4:	Functionality originally defined in:	Description
Multiplex	FlexMux	ISO/IEC 14496-1 (normative)	-	Flexible 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 14496-1 (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()

”.

Remove subclause 2.2 (Definitions).

Remove subclause 3.2 (Definitions).

Remove subclause 4.3 (GA-specific definitions).

Remove subclause 5.3 (Definitions).

Remove subclause 6.2 (Definitions).

Remove subclause 7.2 (Definitions).

Replace subclause 1.3 (Terms and Definitions) with the following:

“

1. **AAC:** Advanced Audio Coding.
2. **Audio access unit**: An individually accessible portion of audio data within an elementary stream.
3. **Audio composition unit**: An individually accessible portion of the output that an audio decoder produces from audio access units.
4. **Absolute time:** The time at which sound corresponding to a particular event is really created; time in the real-world. Contrast score time.
5. **Actual parameter:** The expression which, upon evaluation, is passed to an opcode as a parameter value.
6. **A-cycle:** See audio cycle.
7. **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.
8. **Alias:** Mirrored spectral component resulting from sampling.
9. **Analysis filterbank:** Filterbank in the encoder that transforms a broadband PCM audio signal into a set of spectral coefficients.
10. **Ancillary data:** Part of the bitstream that might be used for transmission of ancillary data.
11. **API:** Application Programming Interface.
12. **A-rate:** See audio rate.
13. **Asig:** The lexical tag indicating an a-rate variable.
14. **Audio buffer:** A buffer in the system target decoder (see ISO/IEC 13818-1) for storage of compressed audio data.
15. **Audio cycle:** The sequence of processing which computes new values for all a-rate expressions in a particular code block.
16. **Audio rate:** The rate type associated with a variable, expression or statement which may generate new values as often as the sampling rate.
17. **Audio sample:** A short snippet or clip of digitally represented sound. Typically used in wavetable synthesis.
18. **AudioBIFS:** The set of tools specified in ISO/IEC 14496-1 (MPEG-4 Systems) for the composition of audio data in interactive scenes.

19. **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.

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

21. **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 produced according to the newer coding standard.

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

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

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

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

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

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

28. **Bitstream 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.

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

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

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

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

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

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

35. **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.

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

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

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

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

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

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

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

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

44. **Context:** See state space.

45. **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."

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

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

48. **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.

49. **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.

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

51. **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.

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

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

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

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

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

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

58. **DFT:** Discrete Fourier Transform.

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

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

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

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

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

64. **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.

65. **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.

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

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

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

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

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

71. **EP:** Error Protection

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

73. **Event:** One control instruction.

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

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

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

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

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

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

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

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

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

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

84. **Fs:** Sampling frequency.

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

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

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

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

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

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

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

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

93. **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.

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

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

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

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

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

99. **HCR:** Huffman codebook reordering

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

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

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

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

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

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

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

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

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

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

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

111. **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.

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

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

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

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

116. **Instance:** See instrument instantiation.

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

118. **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.

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

120. **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.

121. **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.

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

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

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

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

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

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

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

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

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

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

132. **Lexical element:** See token.

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

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

135. **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.

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

137. **LPC:** Linear Predictive Coding.

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

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

140. **LSP:** Line Spectral Pairs.

141. **LTP:** Long Term Prediction.

142. **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.

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

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

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

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

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

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

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

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

151. **MPE:** Multi Pulse Excitation.

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

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

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

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

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

157. **Multiplexing:** Combining several bitstreams into one.

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

159. **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 subclause 4.5.3.1). In case of other codecs, it tis the total number of channels.

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

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

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

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

164. **OD:** Object Descriptor.

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

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

167. **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.

168. **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.

169. **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.

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

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

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

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

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

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

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

177. **PNS:** Perceptual Noise Substitution.

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

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

180. **PQF:** polyphase quadrature filter

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

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

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

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

185. **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.

186. **Program:** A set of main audio channels, coupling_channel_elements (see subclause 4.5.2.1), lfe_channel_elements (see subclause 4.5.2.1) 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 (see subclause 4.5.1.2). A given single_channel_element (see subclause 4.5.2.1), channel_pair_element (see subclause 4.5.2.1), coupling_channel_element, lfe_channel_element or data channel may accompany one or more programs in any given bitstream.

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

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

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

190. **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.

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

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

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

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

195. **RPE:** Regular Pulse Excitation.

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

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

198. **Sample:** See Audio sample.

199. **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.

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

201. **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.

202. **SAOL orchestra:** See orchestra.

203. **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.

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

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

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

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

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

209. **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.

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

211. **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.

212. **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.

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

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

215. **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.

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

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

218. **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.

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

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

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

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

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

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

225. **SQ:** Scalar Quantization.

226. **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.

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

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

229. **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.

230. **Stuffing (bits); stuffing (bytes):** Code-words that may be inserted at particular locations in the coded bitstream 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.

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

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

233. **Symbol table:** In an MPEG-4 Structured Audio bitstream, 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.

234. **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.

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

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

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

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

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

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

241. **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.

242. **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.

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

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

245. **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.

246. **TNS:** Temporal Noise Shaping

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

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

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

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

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

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

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

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

255. **Variable:** See signal variable.

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

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

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

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

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

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

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

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

264. **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.

265. **VQ:** Vector Quantization.

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

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

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

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

”.

In subclause 1.4.1 (Arithmetic operators), add the following definition:

„

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

”.

In clause 1.4.6 (Mnemonics), definition of bslbf, replace

„
ISO/IEC 11172
"

with

„
ISO/IEC 14496
".

In subclause 1.4.8 (Method of describing bit stream syntax), add

„

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.

".

In subclause 1.4.8 (Method of describing bit stream syntax), replace

„
The bitstream retrieved by the decoder is described in the syntax section of each subpart. Each data item in the bitstream is in bold type. It is described by its name, its length in bits, and a mnemonic for its type and order of transmission.
"

with

„
The bitstream payload retrieved by the decoder is described in the syntax section of each subpart. Each data element 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.),
- 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.

".

At the end of subclause 1.4.8 (Method of describing bit stream syntax), remove the following paragraph:

„
The number of bits for each data element is written in the second column. "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.

”.

In subclause 1.4.8 (Method of describing bit stream syntax), remove

”

Definition of bytealigned function

The function bytealigned ( ) returns 1 if the current position is on a byte boundary, that is the next bit in the bit stream is the first bit in a byte. Otherwise it returns 0.

”.

In subclause 1.5.1.1 (Audio object type definition), replace Table 1.1 (Audio Object Type definition) and the subsequent notes as follows:

”

Tools/ Modules / Audio Object Type	gain control	block switching	window shapes - standard	window shapes – AAC LD	filterbank - standard	filterbank - SSR	TNS	LTP	intensity	coupling	MPEG-2 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 4kbs VR	SA tools	SASBF	MIDI	HILN	TTSI	Remark	Object Type ID
Null																																0
AAC main		X	X		X		X		X	X	X	X	X				X														2)	1
AAC LC		X	X		X		X		X	X		X	X				X															2
AAC SSR	X	X	X			X	X		X	X		X	X				X															3
AAC LTP		X	X		X		X	X	X	X		X	X				X														2)	4
(Reserved)																																5
AAC Scalable		X	X		X		X	X	X			X	X	X	X	X	X														6)	6
TwinVQ		X	X		X		X	X					X					X														7
CELP																							X									8
HVXC																									X							9
(Reserved)																																10
(Reserved)																																11
TTSI																															X	12
Main synthetic																											X	X	X			3)	13
Wavetable synthesis																												X	X			4)	14
General MIDI																													X				15
Algorithmic Synthesis and Audio FX																											X						16
ER AAC LC		X	X		X		X		X			X	X				X			X	X	X										17
(Reserved)																																18
ER AAC LTP		X	X		X		X	X	X			X	X				X			X	X	X									5)	19
ER AAC scalable		X	X		X		X		X			X	X	X	X	X	X			X	X	X									6)	20
ER TwinVQ		X	X		X		X						X					X			X	X										21
ER BSAC		X	X		X		X		X			X	X						X		X	X										22
ER AAC LD				X	X		X	X	X			X	X				X			X	X	X										23
ER CELP																					X	X	X	X								24
ER HVXC																					X	X			X	X						25
ER HILN																					X	X								X		26
ER Parametric																					X	X			X	X				X		27
(Reserved)																																28
(Reserved)																																29
(Reserved)																																30
(Reserved)																																31

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.

“.

In subclause 1.5.1.2.2 (AAC – Main object) after the first sentence, add the following sentence:

„
The restrictions of the AAC Main profile with respect to multiple programs and mixdown elements also apply to the AAC Main object type.
".

In subclause 1.5.2.1 (Profiles), replace

„
The **Main Audio Profile** is a rich superset of all the other Profiles, containing tools for natural and synthetic audio. The Main Audio Profile is a superset of the other three profiles (scalable, speech, synthesis).
"

with

„
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.
".

In subclause 1.5.2.1 (Profiles), add at the end

„
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.
".

In subclause 1.5.2.2 (Complexity Units), replace

„
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

„
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).
".

In subclause 1.5.2.2 (Complexity Units), in Table 1.3, replace

„
Sampling Rate
"

with

„
Sampling Rate Conversion
".

In subclause 1.5.2.2 (Complexity Units), in Table 1.3, in the rightmost column of row "Sampling Rate", add

"

7)

".

In subclause 1.5.2.2 (Complexity Units), below Note 6) , add

"

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.

".

In subclause 1.5.2.2 (Complexity units), add the following explanation and formulas for the calculation of PCUs and RCUs:

"

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 *(#SCE * 1 + #CPE * 2+ #LFE * 1 + #DepCouplingCh * 0.3 + #IndepCouplingCh * 1)

fs:	actual sampling frequency
fs_ref:	reference sampling frequency (sampling freqency for the given PCU_ref)
PCU_ref:	reference PCU given in the standard
#SCE:	number of SCEs
#CPE:	...

RCUs

#CPE<2:

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

#CPE>=2:

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

RCU_ref:	reference RCU given in the standard
#SCE:	number of SCEs

#CPE: ...

”.

In subclause 1.5.2.3 (Levels within Profiles), replace

”

A number of 0 stages of interleaving for the EP-tool indicates that the EP is not used in that particular level. The notation used to specify the number of audio channels indicates the number of full bandwidth channels and the number of low-frequency enhancement channels. For example, “5.1” indicates 5 full bandwidth channels and one low-frequency enhancement channel.

”

with

”

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 1.3a 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 1.3a 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

”.

In subclause 1.5.2.3 (Levels within Profiles), in column 2 (Max. channels/object) of Table 1.4 (Levels for the High Quality Audio Profile) and Table 1.7 (Levels for the Mobile Audio Internetworking Profile), replace all occurrences of

”

5.1

”

with

”

5

”.

In subclause 1.5.2.3 (Levels within Profiles), replace all occurrences of

”

*1: This number does not cover FEC for the EP header, i. e. FEC for the EP header is always permitted. In case of several audio objects the limit is valid independently for each audio object. This value is the maximum redundancy for the Audio object, which has the longest maximum frame length, in each profile & level. For audio object types supporting variable frame lengths and arbitrary bitrates (i. e. any AAC audio object type) this does just extend the required decoder input buffer but does not affect the amount of redundancy actually applied. Nevertheless, the application of any class FEC is not permitted if 0% is specified.

”

with

„

*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.

".

In subclause 1.5.2.3 (Levels within Profiles), add at the end of the level definition for the Scalable Audio Profile:

„

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.

".

In subclause 1.5.2.3 (Levels within Profiles), add at the end of the level definition for the Main Audio Profile:

„

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.

".

In subclause 1.5.2.3 (Levels within Profiles), add at the end of the level definition for the High Quality Audio Profile:

„

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.

".

In subclause 1.5.2.3 (Levels within Profiles), add at the end of the level definition for the Natural Audio 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.

”.

In subclause 1.5.2.3 (Levels within Profiles), add at the end of the level definition for the Mobile Audio Internetworking Profile:

”

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.

”.

Add subclause 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 1.7z.

Table 1.7z – 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 - 0x7F	reserved for ISO use	-
0x80 - 0xFD	user private	-
0xFE	no audio profile specified	-
0xFF	no audio capability required	-
NOTE — Usage of the value 0xFE indicates that the content described by this InitialObjectDescriptor does not comply to any audio profile specified in ISO/IEC 14496-3. Usage of the value 0xFF indicates that none of the audio profile capabilities are required for this content.		

”.

In subclause 1.6.2.1, add the following text at the beginning:

”

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.

”.

In subclause 1.6.2.1, add the following text at the end, after the table:

”

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 lengths shall be derived by difference from the length of AudioSpecificConfig().

”.

Replace subclause 1.6.2.2 (Payloads) with:

”

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 1.9 – Audio Object Types

Audio Object Type	Object Type ID	definition of elementary stream payloads and detailed syntax	Mapping of audio payloads to access units and elementary streams
AAC MAIN	1	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
AAC LC	2	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
AAC SSR	3	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
AAC LTP	4	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
AAC scalable	6	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.3
TwinVQ	7	ISO/IEC 14496-3 subpart 4	
CELP	8	ISO/IEC 14496-3 subpart 3	
HVXC	9	ISO/IEC 14496-3 subpart 2	
TTSI	12	ISO/IEC 14496-3 subpart 6	
Main synthetic	13	ISO/IEC 14496-3 subpart 5	
Wavetable synthesis	14	ISO/IEC 14496-3 subpart 5	
General MIDI	15	ISO/IEC 14496-3 subpart 5	
Algorithmic Synthesis and Audio FX	16	ISO/IEC 14496-3 subpart 5	
ER AAC LC	17	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER AAC LTP	19	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER AAC scalable	20	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER Twin VQ	21	ISO/IEC 14496-3 subpart 4	
ER BSAC	22	ISO/IEC 14496-3 subpart 4	
ER AAC LD	23	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER CELP	24	ISO/IEC 14496-3 subpart 3	
ER HVXC	25	ISO/IEC 14496-3 subpart 2	
ER HILN	26	ISO/IEC 14496-3 subpart 7	
ER Parametric	27	ISO/IEC 14496-3 subpart 2 and 7	

1.6.2.2.2 Mapping of audio payloads to access units and elementary streams

1.6.2.2.2.1.2 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.

1.6.2.2.2.1.3 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.

1.6.2.2.2.1.4 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 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 1.6.3.5). 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).

”.

In subclause 1.7.1 (Overview), replace

”

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

”

with

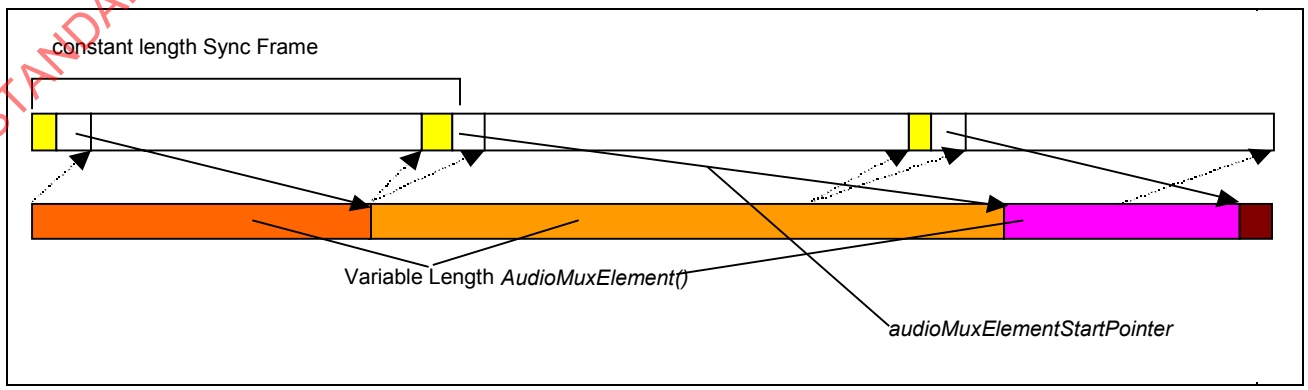
”

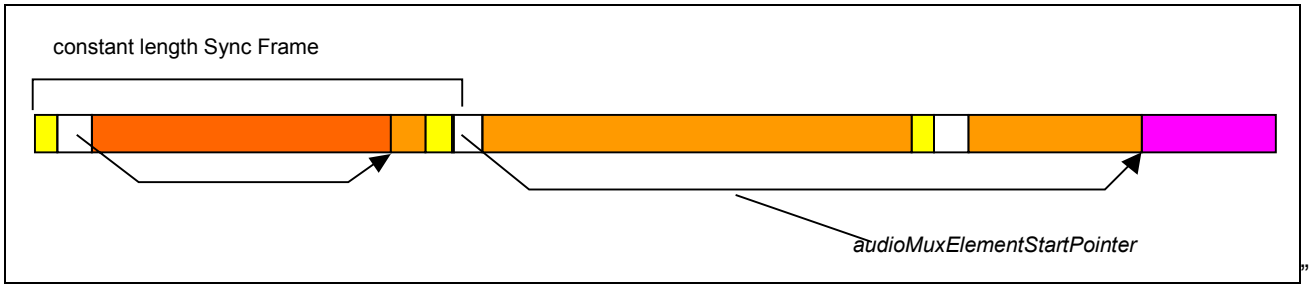
The mechanism defined in this subclause should not be used for transmission of TTSI objects (12), Main Synthetic objects (13), Wavetable Synthesis objects (14), General MIDI objects (15) and Algorithmic Synthesis and Audio FX objects (16). It should further not be used for transmission of any object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

”.

In subclause 1.7.2 (Synchronization Layer), replace Figure 1.2 (Synchronization in Audio PointerStream()):

”



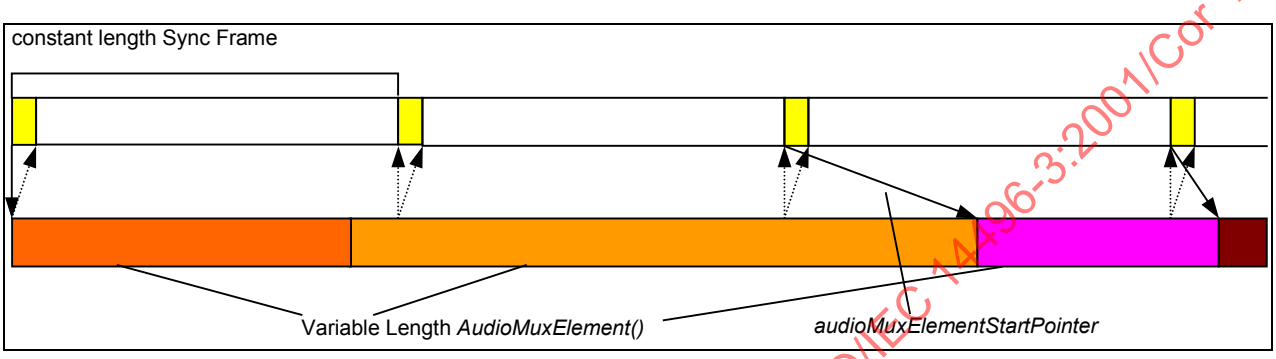


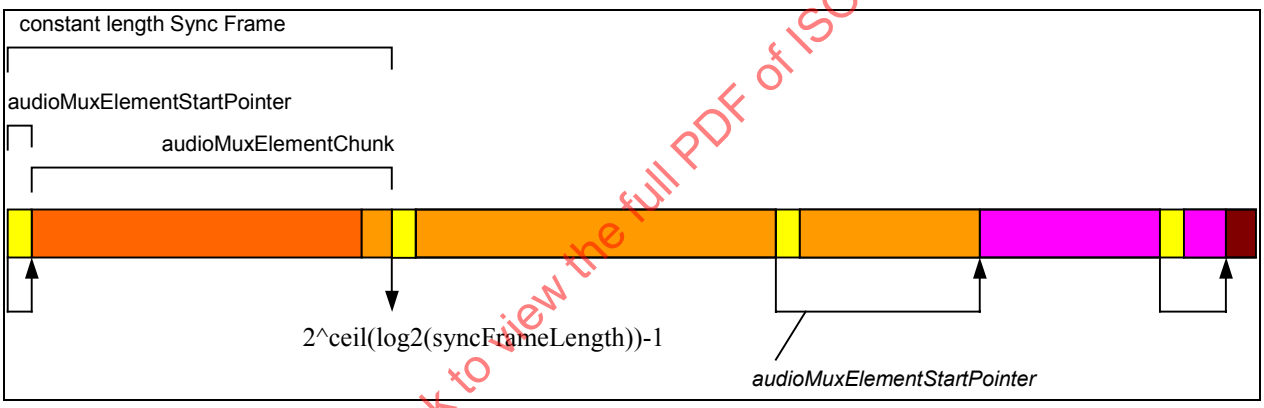
”

”

with

”





”.

In subclause 1.7.2.1 (Syntax), replace Table1.18 (Syntax of AudioPonterStream())

”

Table 1.18 – Syntax of AudioPointerStream()

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

”

with

"

Table 1.18 – Syntax of AudioPointerStream()

Syntax	No. of bits	Mnemonic

AudioPointerStream (syncFrameLength)
{
while (! EndOfStream) {
AudioPointerStreamFrame (syncFrameLength);
}
}		

Table 1.18a – Syntax of AudioPointerStreamFrame()

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

".

In subclause 1.7.2.1 (Syntax)), replace the name of the syntax element:

"
audioMuxLengthBytesLast
"

with

"
audioMuxLengthBytes
".

*In subclause 1.7.2.2 (Semantics), change the semantic description of **audioMuxLengthBytesLast** and **audioMuxLengthBytes** to:*

"

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

".

In subclause 1.7.2.2 (Semantics), replace:

"

audioMuxElementStartPointer A field indicating the start point of the multiplexed element. The number of bits required for this field is calculated as nbits = ceil(log2(the transmission frame length)). The transmission frame length should be provided from the transmission layer. The maximum possible value of this field is reserved to signal that there is no start of an access unit in this sync frame.

"

with

"

audioMuxElementStartPointer A field indicating the starting point of the first AudioMuxElement() within the current AudioPointerStreamFrame(). The number of bits required for this field 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 field is reserved to signal that there is no start of an AudioMuxElement() in this sync frame.

”.

In subclause 1.7.2.2 (Semantics), add:

”

audioMuxElementChunk A part of a concatenation of subsequent AudioMuxElements (see Figure 1.2).

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

”.

In subclause 1.7.3 (Multiplex Layer), replace:

”

If the transmission payload allows only byte-aligned payload, zero-padding bits for byte alignment shall follow the multiplexed element.

”

with

”

If the transmission payload allows only byte-aligned payload, padding bits for byte alignment shall follow the multiplexed element.

”.

In subclause 1.7.3.2 (Semantics), replace:

”

latmBufferFullness state of the bit reservoir. It is transmitted as the number of available bits in the bit reservoir divided by the number of audio channels divided by 32 and truncated to an integer value (mean number of 32 bit words per channel remaining in the encoder buffer after encoding the first audio frame in the AudioMuxElement()). A value of hexadecimal FF signals that the bitstream is a variable rate bitstream. In this case, buffer fullness is not applicable.

”

with

”

latmBufferFullness[streamID[prog][lay]] 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.
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 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 subclause 1.7.3.2 (Semantics), add the following text after the first two tables, before the specification of **epUsePreviousMuxConfig***:*

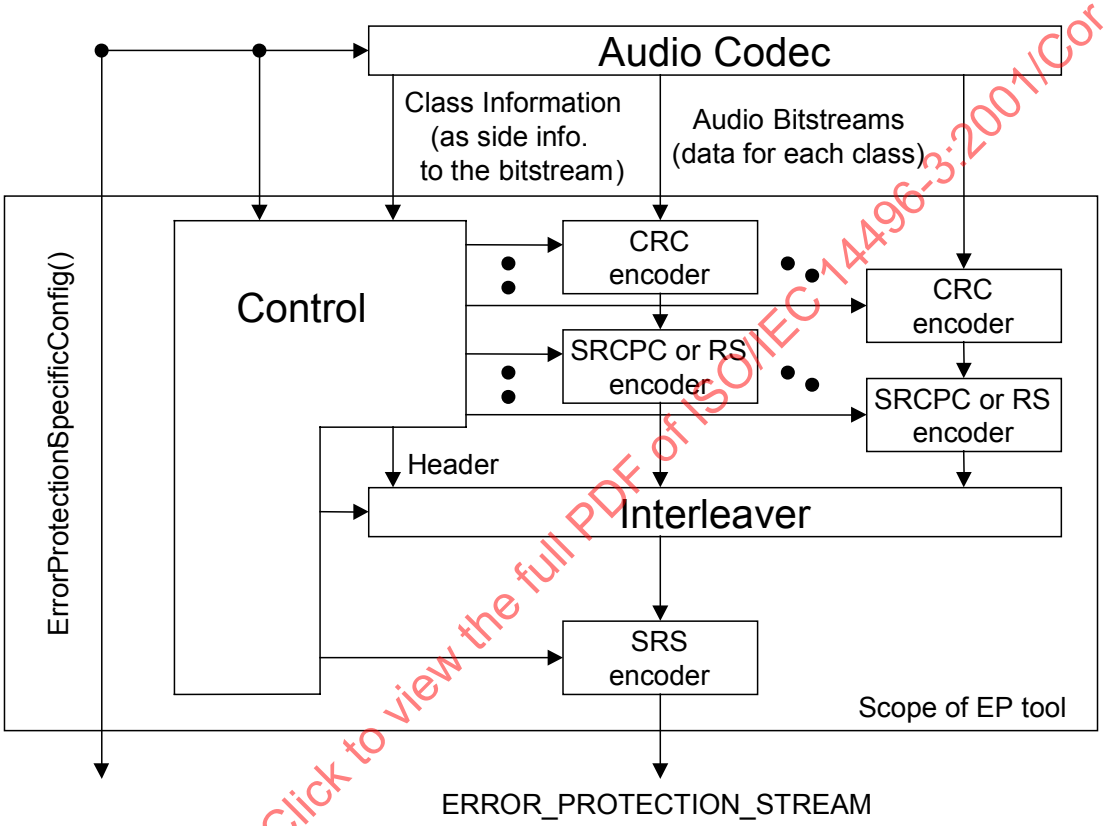
”

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 clause 1.6).

”.

In subclause 1.8.1 (Overview of the tools), replace figure 1.5 (Outline of EP encoder)

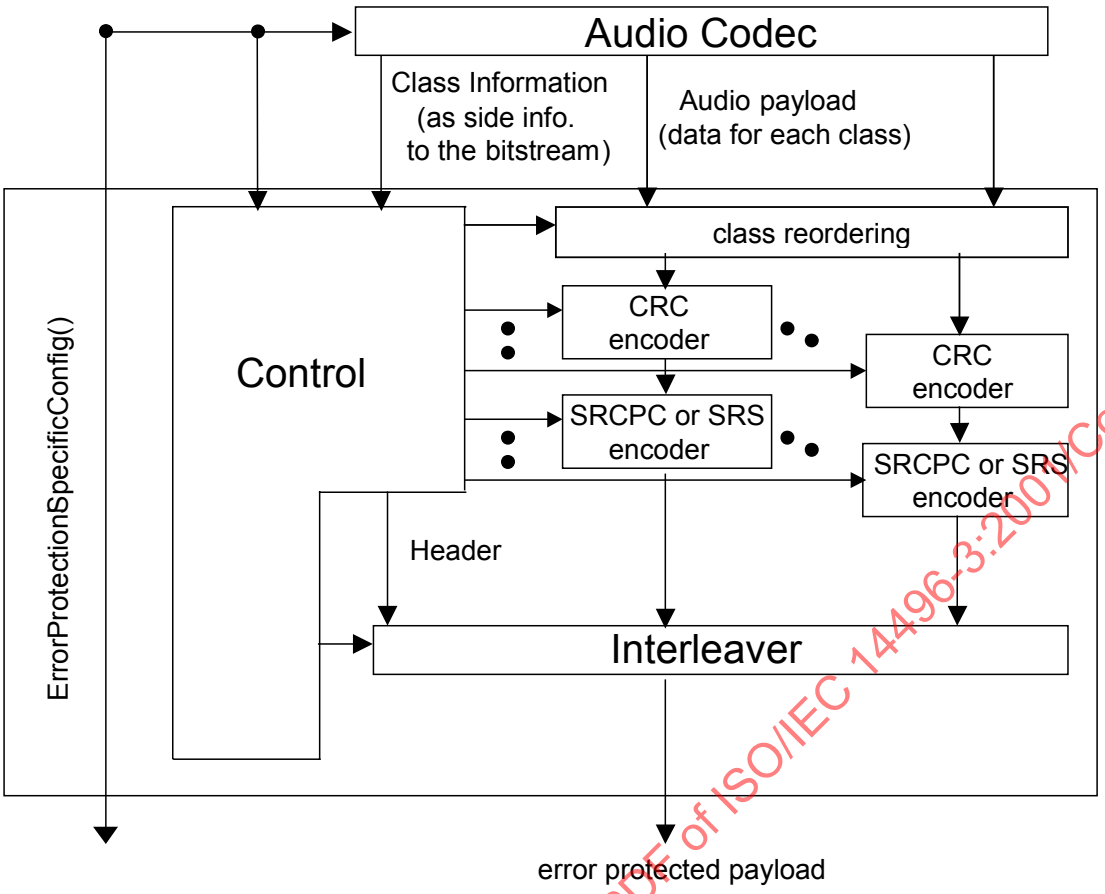
”



“

with

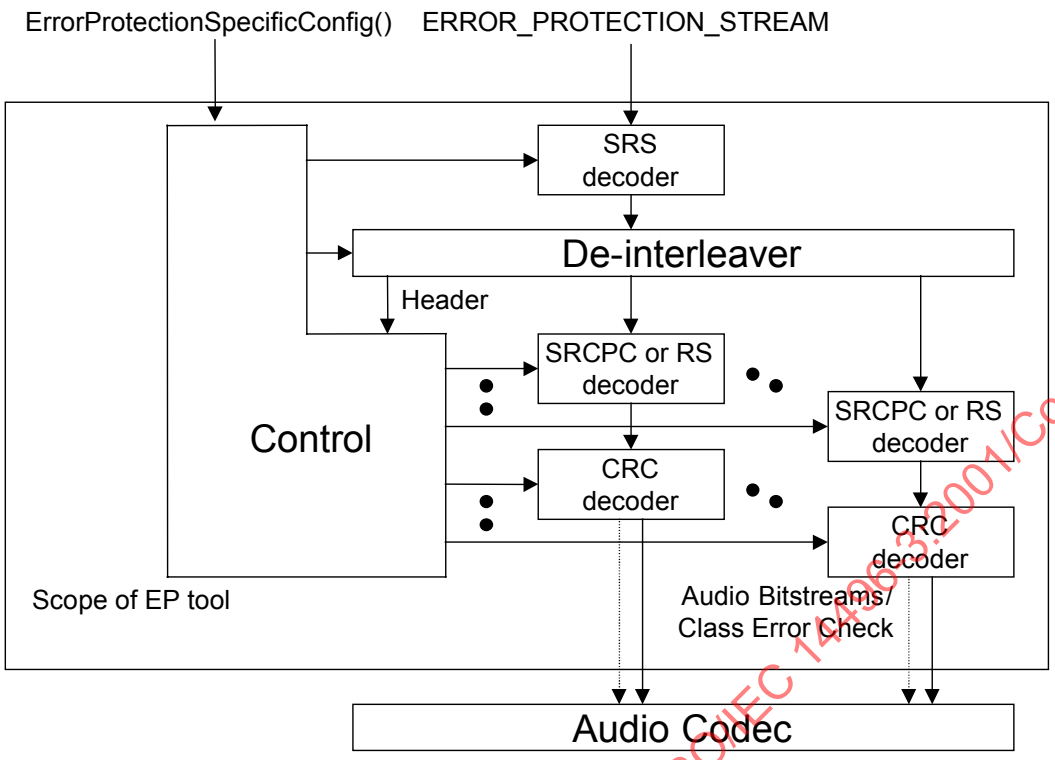
”



”.

In subclause 1.8.1 (Overview of the tools), replace figure 1.6 (Outline of EP decoder)

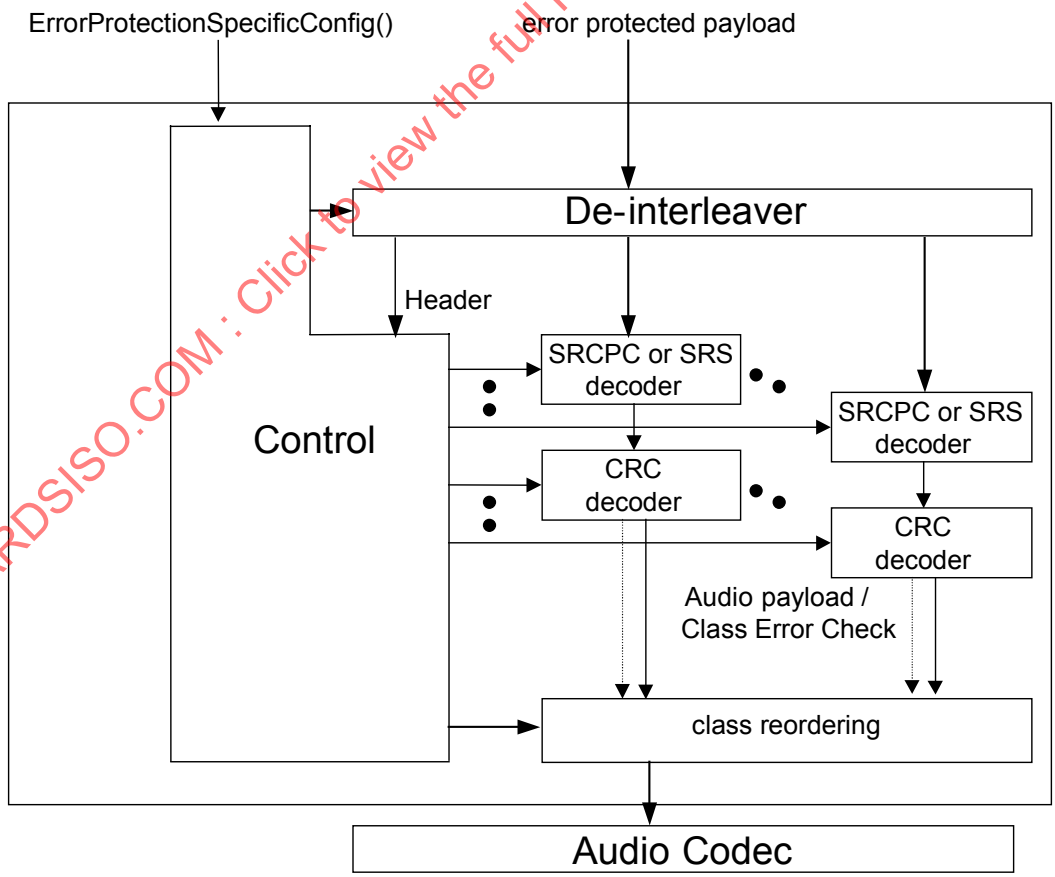
”



”

with

”



“.

In subclause 1.8.2.1 (Error protection specific configuration), replace table 1.27 (Syntax of ErrorProtectionSpecificConfig())

”

Table 1.27 – 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**
}		
rs_fec_capability;	**7**	**uimsbf**
}		

”

with

”

Table 1.27 – 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**
}		
}		

“

In subclause 1.8.2.2 (Error protection bitstream payloads), replace the description below table 1.30 (Syntax of ep_header ()) with:

„

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

".

In subclause 1.8.2.2, replace

„

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

"

with

„

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.

".

In subclause 1.8.2.2 (Error protection bitstream payloads), remove table 1.28 (Syntax of rs_ep_frame()).

In subclause 1.8.2.2 (Error protection bitstream payloads), replace Table 1.29 (Syntax of ep_frame())

"

Syntax	No. of bits	Mnemonic

ep_frame()
{
if (interleave_type == 0){
ep_header();
ep_encoded_classes();
}
if (interleave_type == 1){
interleaved_frame_mode1;
	1 -	**bslbf**

}
if (interleave_type == 2){
interleaved_frame_mode2;
	1 -	**bslbf**

} **stuffing_bits;** }	**Nstuff**	**bslbf**

"

with

“

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

”.

In subclause 1.8.2.2 (Error protection bitstream payloads), replace Table 1.31 (Syntax of class_attrib())

“

Syntax	No. of bits	Mnemonic
class_attrib(class_count, length_escape, rate_escape, crclen_escape, frame_pred)		
{		
for(j=0; j<class_count; j++){		
if (length_escape[frame_pred][j] == 1){		
class_bit_count[j];	**Nbitcount**	**uimsbf**
}		
if (rate_escape[frame_pred][j] == 1){		
class_code_rate[j];	**3**	**uimsbf**
}		
if (crclen_escape[frame_pred][j] == 1){		
class_crc_count[j];	**3**	**uimsbf**
}		
}		
if (bit_stuffing == 1){		
num_stuffing_bits;	**3**	**uimsbf**
}		
}		

class_count: number of class for this frame
frame_pred: selected predefined set for this frame

“

with

“

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**
}		
}		

”.

In subclause 1.8.2.2 (Error protection bitstream payloads), replace in Table 1.32 (Syntax of ep_encoded_classes())

“

Syntax	No. of bits	Mnemonic
ep_encoded_classes(class_count)		
{		
for(j=0; j<class_count; j++){		
ep_encoded_class[j];		**bslbf**
}		
}		

“

with

“

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**
}		
}		

”.

In subclause 1.8.3.1 (Definitions), remove

”

rs_fec_capability	This field indicates the correction capability of SRS code to protect the whole frame (see subclause 1.8.4.7). This capability is given as number of erroneous bytes that can be corrected. The value “0” indicates SRS is not used.
rs_ep_frame()	Reed-Solomon error protected frame that is applied Reed-Solomon code.
rs_parity_bits	The Reed-Solomon parity bits for ep_frame(). See subclause 1.8.4.7.

“.

In subclause 1.8.3.1 (Definitions), replace Table 1.34 (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

”

with

”

fec_type	Codeword	000	001	010	011	100	101	110	111
0	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
1 or 2	number of correctable bytes	0	1	2	4	8	16	32	64

”.

In subclause 1.8.3.1 (Definitions, interleave_switch[l][j]), replace

”

1-interleaved without intraclass-interleaving ... number of bytes wihtin ...

”

with

”

1-interleaved without intraclass-interleaving ... number of bytes within ...

“.

In subsection 1.8.4.1 (Out-of-band information), replace

”

The length of the last class can be specified to last “until the end”.

“

with

”

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 subclause 1.8.4.2 (Derivation of pre-defined sets), replace

„

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 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 don’t exist” to “all the optClasses exists”.

Algorithm:

```
transPred = 0;
for ( i = 0; i < nPred; i++ ) {                  /* for all predefinition sets */
    for ( j = 0; j < pow ( 2, NCO[i] ); j++ ) { /* unwraping */
        for ( k = 0; k < NCO[i]; k++ ) {         /* for all optional classes */
            if ( j & ( 0x01 << k ) ) {
                optClassExists[k] = 1;
            }
            else {
                optClassExists[k] = 0;
            }
        }
        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 i-th original pre-defined set, except it don’t have optClasses whose optClassExists equals to 0.

Example

ErrorProtectionSpecificConfig() defines pre-defined sets as follows:

Table 1.36 – The example of pre-defined set

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

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

Table 1.37 – The example of pre-defined sets after the pre-processing

Pred #0	Pred #1	Pred #2	Pred #3	Pred #4	Pred #5
Class B	Class A	Class B	Class A	Class F	Class E
Class D	Class B	Class C	Class B		Class F
	Class D	Class D	Class C		
			Class D		

“

with

”

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 1.36 – 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 1.37 – 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									

”.

In subclause 1.8.4.3 (In-band information), replace Table 1.38

"

Table 1.38 - Basic set of FEC codes for in-band information

Number of bit 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 -	-

Npred_parity(or Nattrib_parity) = total number of bits – Number of bit to be protected

"

with

"

Table 1.38 – Basic set of FEC codes for in-band information

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

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

".

In subclause 1.8.4.3 (In-band information), replace

"

If a header length exceeds 16 bits, this header is protected in the same way as the class information. 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.

"

with

"

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.

".

In subclause 1.8.4.3 (In-band information), replace

„
The generation polynomials for each FEC is as follows:

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

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

Golay(23,12): $x^{11}+x^9+x^7+x^6+x^5+x^3+x+1$

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

”

with

„
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): $x^{15}+x^{11}+x^{10}+x^9+x^8+x^7+x^5+x^3+x^2+x+1$

”.

In subclause 1.8.4.4 (Concatenation functionality), Figure 1.7 (Concatenation procedure), replace

„
P[m]
”

with

„
P[M]
“.

In subclause 1.8.4.4 (Concatenation functionality), Figure 1.7 (Concatenation procedure), replace

“
$j<m$
“

with

“
$j<M$
”.

In subclause 1.8.4.5 (CRC), replace

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

with

"

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

".

In subclause 1.8.4.5 (CRC), replace

"

Using these CRC bits, the decoder should 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. See the informal annex (example of error concealment).

"

with

"

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.

".

In subclause 1.8.4.6.3 (Puncturing of SRC for SRCPC code), replace

"

The puncturing is made with the period of 8, and each bit of Pr(i) indicates the corresponding vt(i) from the SRC encoder is punctured (not transmitted) or not (transmitted). Each bit of Pr(i) is used from MSB to LSB, and 0/1 indicates not-punctured/punctured respectively. The code rate is a property of the class, thus the choice of the table is made according which class the current bit belongs to. After this decision which bits from vt(i) is transmitted, they are output in the order from vt(0) to vt(3).

"

with

"

The puncturing pattern, which is applied with the period of 8, depends on the class_rate (see Table 1.40). 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).

".

Replace subclause 1.8.4.7 (Shortened Reed-Solomon codes), with the following:

"

Shortened RS codes SRS(255-*l*, 255-2*k*-*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 255-2*k*. The lengths of the parts are calculated as follows:

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

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

N = *ceil* (*L* / (255-2*k*))

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 255-2*k* 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^j is shown in Table 1.41, where the MSB of the octet is transmitted first.

Table 1.41 – Binary representation for α^i $(0 \le i \le 254)$ 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 2*k* 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.

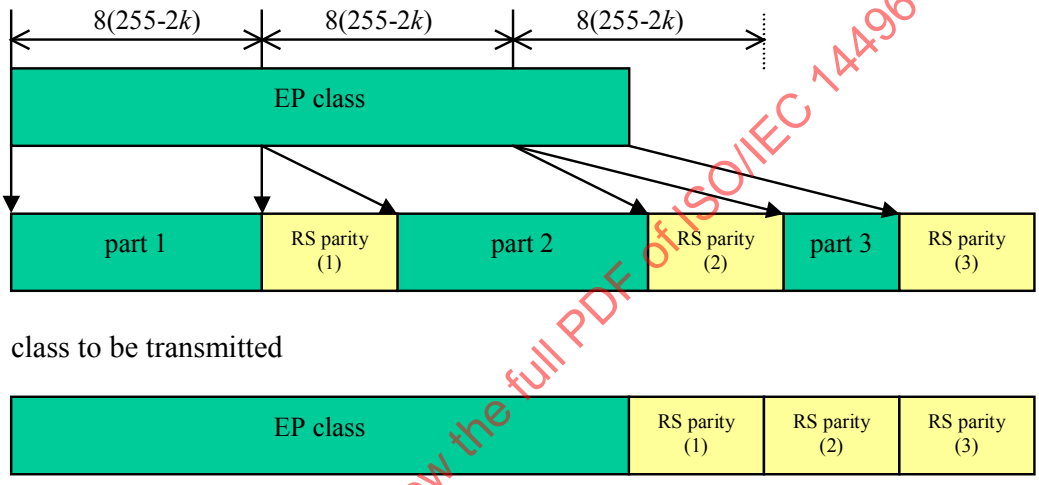


Figure 1.9 – RS encoding of EP frame

".

In subclause 1.8.4.8.2 (Modes of interleaving), replace

"

Two modes of interleaving, mode 1 and mode 2 are defined in the following subclauses.

"

with

"

Two modes of interleaving, mode 1 and mode 2, according to interleave_type 1 and 2, are defined in the following subclauses. Table 1.42 and Table 1.43 give an overview of the available configurations.

Table 1.42 – 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 below)	
3	reserved	

Table 1.43 – 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.

“.

In subclause 1.8.4.8.2.1 (Interleaving operation in mode 1), replace

”

Multi-stage interleaving is processed for **ep_encoded_class** from the last class to first class, and then 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.

”

with

”

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.

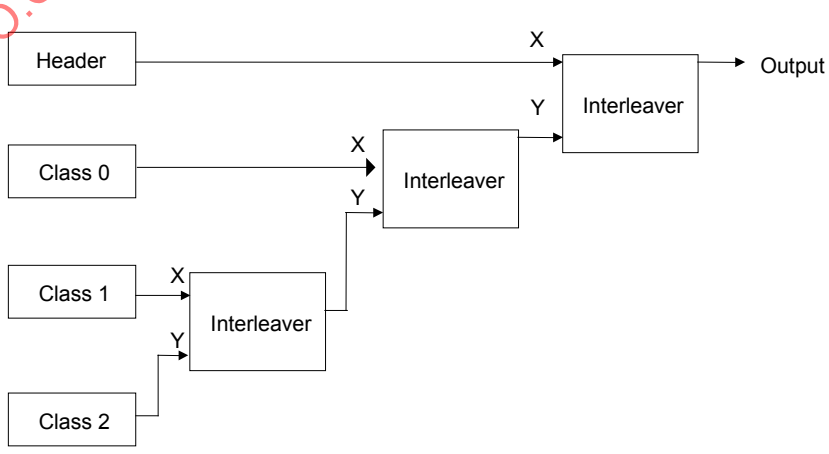
The width of the interleaving matrix is chosen according to the FEC in use. In the case of SRCPC coding (fec_type == 0), the width of the interleaving matrix is 28 bit. In the case of SRS coding (fec_type == 1 or 2), the width of the interleaving matrix is equal to the length of the class in bytes. Figure 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 1.38, or 28 bits if SRCPC is used.

“.

In subclause 1.8.4.8.2.1 (Interleaving operation in mode 1), replace Figure 1.15

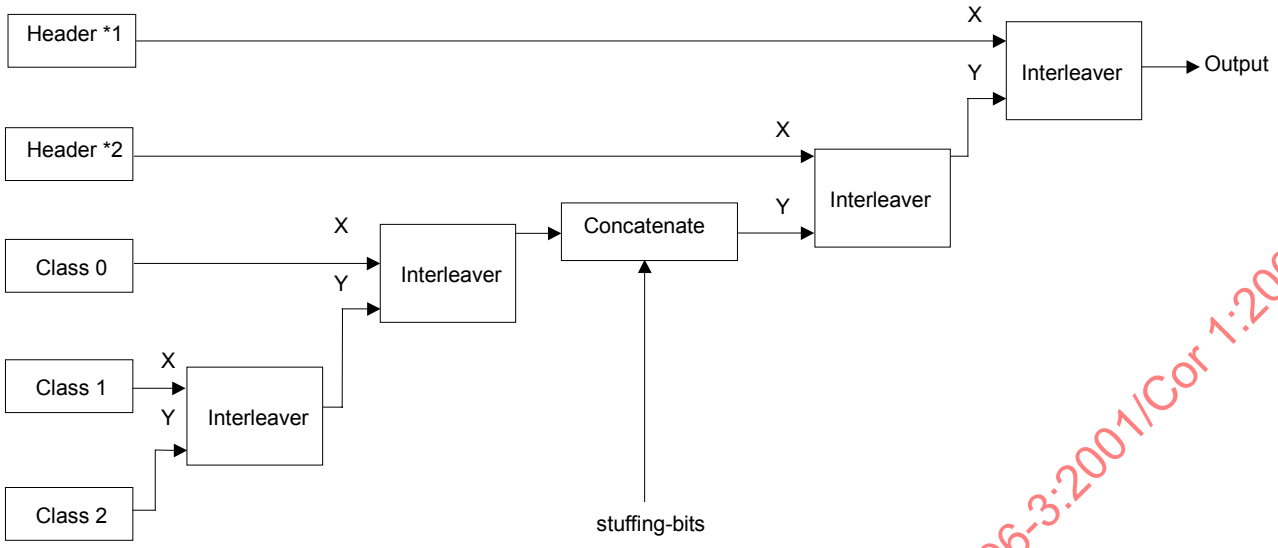
”



“

with

"



".

In subclause 1.8.4.8.2.2 (Interleaving operation in mode 2), replace

"

```
    clear buffer BUF_NO /* Buffer for non-interleaved part. */
    clear buffer BUF_Y  /* Buffer for Y input in the next stage */
    for j = 0 to N-1 {
       if ( interleave_switch[i][j] == 3) {
       concatenate ep_encoded_class[j] at hte end of BUF_N0;
       }
    }
    set BUF_N0 into BUF_Y;
    clear buffer BUF_N0
    for j = N-1 to 0 {
       if ( interleave_switch[i][j] == 0 ) {
          concatenate ep_encoded_class[j] at the end of BUF_NO;
       } else if ( interleave_switch[i][j] != 3){
          if ( interleave_switch[i][j] == 1 ) {
             set the size of the interleave window to be the length of ep_encoded_class[j];
          } else if ( interleave_switch[i][j] == 2 ) {
             set the size of the interleave window to be 28;
          }
          input ep_encoded_class[j] 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;
       }
    }
    concatenate BUF_NO at the end of 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;
```

"

with

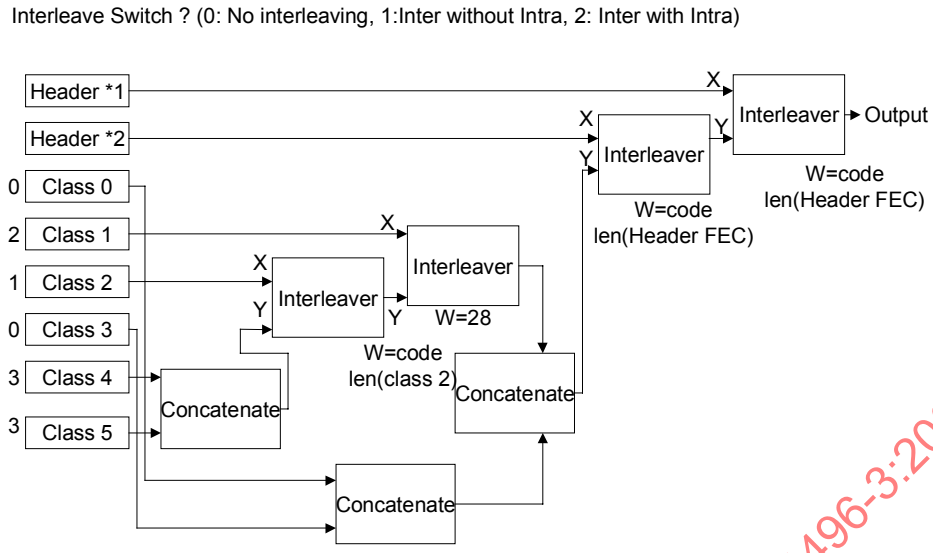
”

```
    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;
```

“.

In subclause 1.8.4.8.2.2 (Interleaving operation in mode 2), replace Figure 1.17

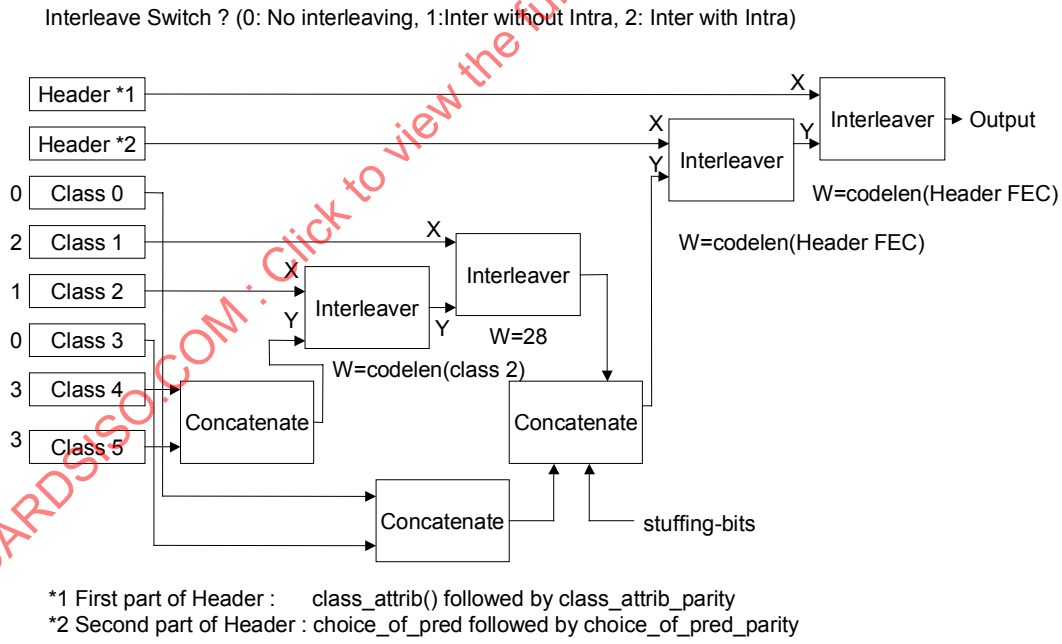
”



“

with

”



“.

In subclause 1.8.4.8.2.2 (Interleaving operation in mode 2), remove

„

The width of interleave matrix is chosen according to the FEC used. In case block codes are used, i.e. In-band information is protected with “Basic set of FEC codes” as shown in Table 1.38, the length of the codeword is used as this width (see subclause 1.8.4.3). In case the RCPC code is used, 28-bit is used as this width.

“.

In subclause 1.8.4.9 (Class reordered output), replace

„

EP tool has a function to re-order the class output to the audio decoder, in order to align the bitstream order as defined for that codec, independently from the EP tool configuration and audio encoder to EP tool interface. Note that this interface is out of the scope of this specification, and is up to the implementation, while the interface to the audio decoder shall be aligned to the standard bitstream to avoid additional signaling from the encoder side.

The order of the class after this re-ordering is signaled as **class_output_order[i][j]** in the out-of-band information. The ep decoder re-orders the classes in the EP frame transmitted 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 when output to the audio decoder.

“

with

„

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 1.17a 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.

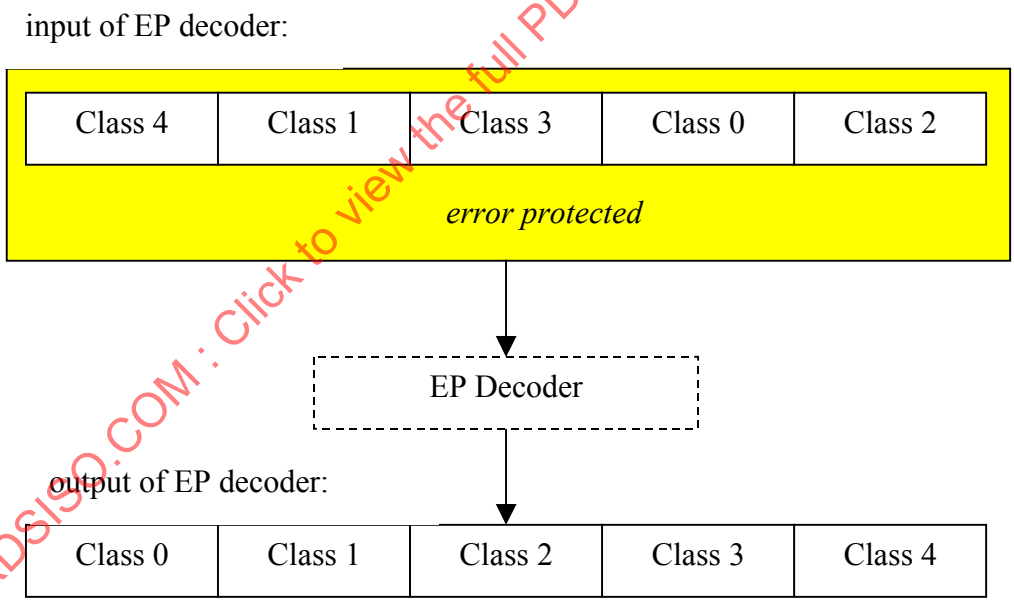


Figure 1.17a – Example for class reordered output with class_output_order 4, 1, 3, 0, 2

“

In subclause 1.A.2.1 (MPEG-2 AAC Audio_Data_Interchange_Format, ADIF), replace Table 1.A.1 (Syntax of adif_sequence())

”

Syntax	No. of bits	Mnemonic

adif_sequence()
{
adif_header()
raw_data_stream()
}		

”

with

”

Syntax	No. of bits	Mnemonic

adif_sequence()
{
adif_header();
byte_alignment();
raw_data_stream();
}		

”.

In subclause 1.A.2.1 (MPEG-2 AAC Audio_Data_Interchange_Format, ADIF), replace 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**	**uimsbf**
for (i = 0; i < num_program_config_elements + 1; i++) {		
if(bitstream_type == '0')		
adif_buffer_fullness	**20**	**uimsbf**
program_config_element()		
}		
}		

”

with

”

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();		
}		
}		

”.

In subclause 1.A.2.1 (MPEG-2 AAC Audio_Data_Interchange_Format, ADIF), replace Table 1.A.3 (Syntax of raw_data_stream())

”

Syntax	No. of bits	Mnemonic
raw_data_stream()		
{		
while (data_available()) {		
raw_data_block()		
byte_alignment()		
}		
}		

“

with

”

Syntax	No. of bits	Mnemonic
raw_data_stream()		
{		
while (data_available()) {		
raw_data_block();		
}		
}		

“.

In subclause 1.A.2.2.1 (Fixed Header of ADTS), remove the data element

”

Emphasis

”.

In subclause 1.A.2.2 (Audio_Data_Transport_Stream frame, ADTS), replace table 1.A.5 (Syntax of adts_frame()) with

”

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();
}
}
}		

”.

In subclause 1.A.2.2.3 (Error detection), add Table 1.A.8a (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**
} }		

”.

In subclause 1.A.2.2.3 (Error detection), add Table 1.A.8b (Syntax of adts_raw_data_block_error_check()):

”

Syntax	No. of bits	Mnemonic

adts_raw_data_block_error_check()
{
if (protection_absent == ‘0’)
crc_check;
	16	**rpchof**

}		

”.

In subclause 1.A.3.1.1 (Definitions: Bitstream elements for ADIF), replace

”

adif_buffer_fullness: State of the bit reservoir after encoding the first raw_data_block() in the aidf_sequence. It is transmitted as the number of available bits in the bit reservoir.

”

with

„

adif_buffer_fullness: State of the bit reservoir after encoding the first raw_data_block() in the adif_sequence(). It is transmitted as the number of available bits in the bit reservoir (table 1.A.2).

“.

In subclause 1.A.3.1.1 (Definitions: Bitstream elements for ADIF), replace the reference to

„

Table 6.1

“

with the reference to

„

Table 1.A.1

“.

In subclause 1.A.3.1.1 (Definitions: Bitstream elements for ADIF), replace the reference to

„

Table 6.2

“

with the reference to

„

Table 1.A.2

“.

In subclause 1.A.3.1.1 (Definitions: Bitstream elements for ADIF), replace

„

num_program_config_element Number of program config elements specified for this adif_sequence().

“

with

„

num_program_config_element Number of program configuration elements specified for this adif_sequence() equals num_program_config_element+1. The minimum value is 0 indicating 1 program configuration element.

“.

In subclause 1.A.3.2.1 (Definitions: Bitstream elements for ADTS), replace the reference to

„

Table 6.3

“

with the reference to

„

Table 1.11

“.

In subclause 1.A.3.2.1 (Definitions: Bitstream elements for ADTS), replace the reference to

„
subclause 6.3.4
”

with the reference to

„
subclause 1.6.3.4
”.

In subclause 1.A.3.2.1 (Definitions: Bitstream elements for ADTS), replace the reference to

„
subclause 6.2.2
”

with the reference to

“
subclause 1.6.3.3
”.

In subclause 1.A.3.2.1 (Definitions: Bitstream elements for ADTS), replace the reference to

„
subclause 5.1.1
”

with the reference to

„
subclause 1.5.2.1
”.

In subclause 1.A.3.2.1 (Definitions: Bitstream elements for ADTS), replace the reference to

„
Table A.4
”

with the reference to

„
Table 1.A.4
”.

In subclause 1.A.3.2.1 (Definitions: Bitstream elements for ADTS), replace the reference to

„
Table A.5
”

with the reference to

”

Table 1.A.5

”.

In subclause 1.A.3.2.1 (Definitions: Bitstream elements for ADTS), replace the reference to

”

Table A.6

”

with the reference to

”

Table 1.A.6

”.

In subclause 1.A.3.2.1 (Definitions: Bitstream elements for ADTS), replace the reference to

”

Table A.7

”

with the reference to

”

Table 1.A.7

”.

In subclause 1.A.3.2.1 (Definitions: Bitstream elements for ADTS), replace the reference to

”

Table A.8

”

with the reference to

”

Table 1.A.7

”.

In subclause 1.A.3.2.1 (Definitions: Bitstream elements for ADTS), remove from the definition of adts_error_check():

”

CRC error detection data generated as described in ISO/IEC 11172-3, subclause 2.4.3.1.

”.

In subclause 1.A.3.2.1 (Definitions: Bitstream elements for ADTS), replace:

”

adts_buffer_fullness	State of the bit reservoir after encoding the first raw_data_block() in the ADTS frame. It is transmitted as the number of available bits in the bit reservoir divided by the number of audio channels divided by 32 and truncated to an integer value. A value of hexadecimal 7FF signals that the bitstream is a variable rate bitstream. In this case, buffer fullness is not applicable.

”

with

„

adts_buffer_fullness	State of the bit reservoir in the course of encoding the ADTS frame, up to and including the first raw_data_block(). It is transmitted as the number of available bits in the bit reservoir divided by NCC divided by 32 and truncated to an integer value (table 1.A.7). A value of hexadecimal 7FF signals that the bitstream is a variable rate bitstream. In this case, buffer fullness is not applicable.

".

In subclause 1.A.3.2.1 (Definitions: Bitstream elements for ADTS), add the following definitions:

„

crc_check	CRC error detection data generated as described in ISO/IEC 11172-3, subclause 2.4.3.1 (Table 1.A.8, Table 1.A.8a, Table 1.A.8b).
raw_data_block_position[i]	Start position of the i-th raw_data_block() in the adts_frame(), measured as an offset in bytes from the start position of the first raw_data_block() in the adts_frame().
adts_header_error_check()	The following bits are protected and fed into the CRC algorithm in order of their appearance: • all bits of adts_fixed_header() • all bits of adts_variable_header() • all bits of every raw_data_block_position[i]

adts_raw_data_block_error_check()	With regard to the i-th adts_raw_data_block_error_check(), the following bits of the i-th raw_data_block() are protected and fed into the CRC algorithm in order of their appearance:
	• First 192 bits of any
	  • single_channel_element (SCE)
	  • channel_pair_element (CPE)
	  • coupling_channel_element (CCE)
	  • low frequency enhancement channel (LFE)
	• First 128 bits of the second individual_channel_stream (ICS) in the channel_pair_element must be protected.
	• All information in any program_configuration_element (PCE) or data_stream_element (DSE) must be protected.
	For any element where the specified length of 128 or 192 bits exceeds its actual length, the element is zero padded to the specified length for CRC calculation. The id_syn_ele bits shall be excluded from CRC protection. If the length of a CPE is shorter than 192 bits, zero data are appended to achieve the length of 192 bits. Furthermore, if the first ICS of the CPE ends at the Nth bit (N<192), the first (192 - N) bits of the second ICS are protected twice, each time in order of their appearance. For example, if the second ICS starts at the 190th bit of CPE, the first 3 bits of the second ICS are protected twice. Finally, if the length of the second ICS is shorter than 128 bits, zero data are appended to achieve the length of 128 bits.

".

Remove subclause 1.B.1 (Text format of out-of-band information).

Add subclause 1.B.2.0

„

1.B.2.0 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.

".

In subclause 1.B.2.3 (Example forTwin-VQ) , remove the given examples plotted in the rectangular boxes.

In subclause 1.B.2.4 (Example for HVXC) , remove the given examples plotted in the rectangular boxes.

In subclause 1.B.2.5 (Example for ER BSAC) , remove the given example plotted in the rectangular box.

In subclause 3.3.2.2.1.3 (RPE wideband syntax), replace Table 3.40 (Syntax of RPE_WideBand_ESC2()) with

”

Syntax	No. of bits	Mnemonic
RPE_WideBand_ESC2()		
{		
if (FineRateControl == ON) {		
if (LPC_Present == YES) {		
lpc_indices [1], 4;	1	uimsbf
lpc_indices [3], 6;	1	uimsbf
lpc_indices [3], 1;	1	uimsbf
lpc_indices [5], 2;	1	uimsbf
lpc_indices [6], 3;	1	uimsbf
lpc_indices [7], 6;	1	uimsbf
lpc_indices [7], 4;	1	uimsbf
lpc_indices [7], 1-0;	2	uimsbf
lpc_indices [9];	1	uimsbf
}		
} else {		
lpc_indices [1], 4;	1	uimsbf
lpc_indices [3], 6;	1	uimsbf
lpc_indices [3], 1;	1	uimsbf
lpc_indices [5], 2;	1	uimsbf
lpc_indices [6], 3;	1	uimsbf
lpc_indices [7], 6;	1	uimsbf
lpc_indices [7], 4;	1	uimsbf
lpc_indices [7], 1-0;	2	uimsbf
lpc_indices [9];	1	uimsbf
}		
for (subframe = 0; subframe < nrof_subframes; subframe++) {		
shape_delay[subframe], 4-3;	2	uimsbf
gain_indices[0][subframe], 2;	1	uimsbf
if (subframe == 0) {		
gain_indices[1][subframe], 2;	1	uimsbf
} else{		
gain_indices[1][subframe], 1;	1	uimsbf
}		
}		
}		

”.

In subclause 3.3.2.2.1.3 (RPE wideband syntax), replace Table 3.42 (Syntax of RPE_WideBand_ESC4()) with

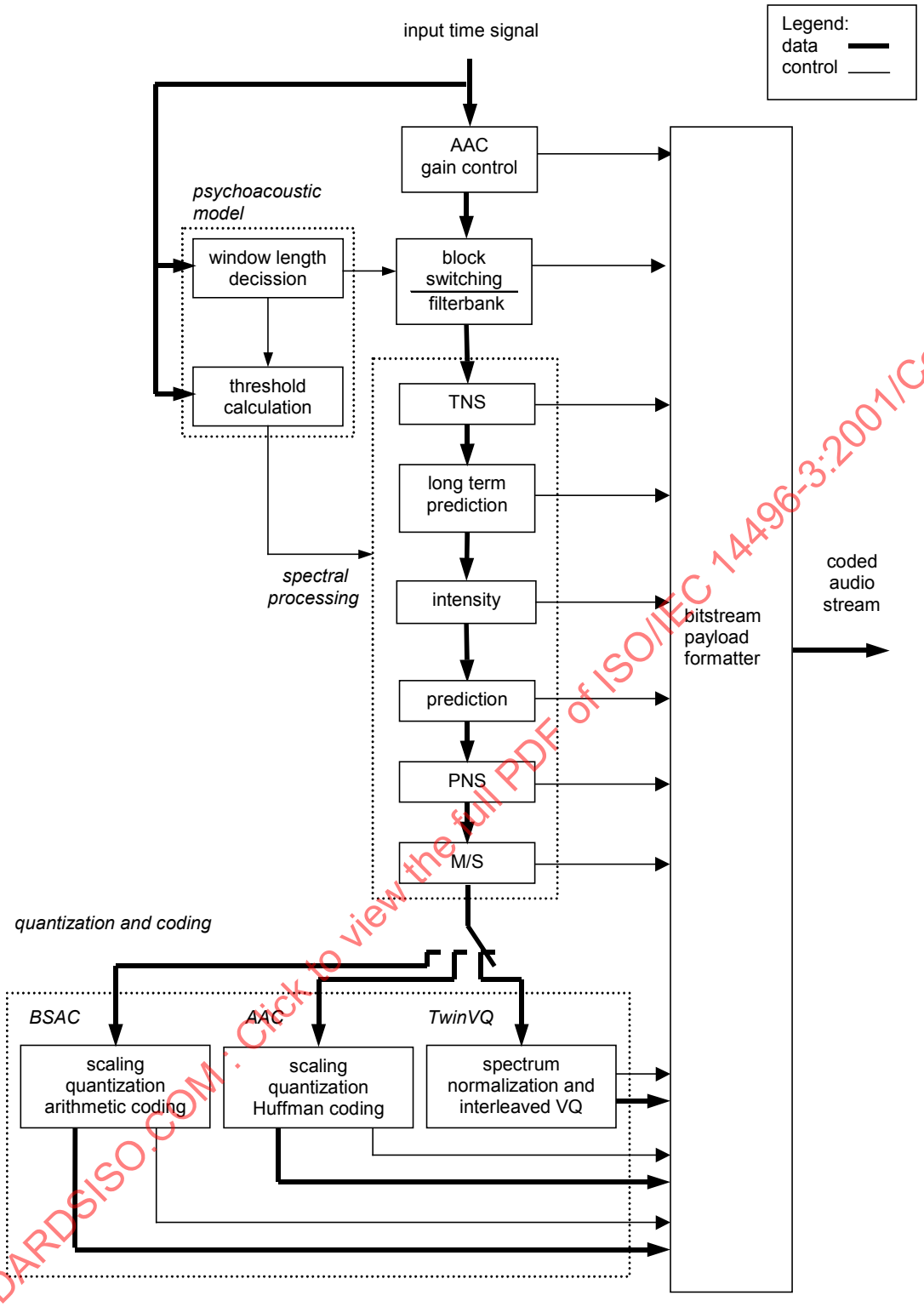
"

Syntax	No. of bits	Mnemonic
RPE_WideBand_ESC4()		
{		
if (FineRateControl == ON) {		
if (LPC_Present == YES) {		
lpc_indices [3], 5;	1	uimsbf
lpc_indices [8], 0;	1	uimsbf
}		
} else {		
lpc_indices [3], 5;	1	uimsbf
lpc_indices [8], 0;	1	uimsbf
}		
for (subframe = 0; subframe < nrof_subframes; subframe++) {		
shape_delay[subframe], 0;	1	uimsbf
shape_index[subframe];	11, 12	uimsbf
gain_indices[0][subframe], 1-0;	2	uimsbf
if (subframe == 0) {		
gain_indices[1][subframe], 1-0;	2	uimsbf
} else{		
gain_indices[1][subframe], 0;	1	uimsbf
}		
}		
}		

".

In subclause 4.1.1.1 (Encoder and Decoder Block Diagrams), replace Figure 4.1 with

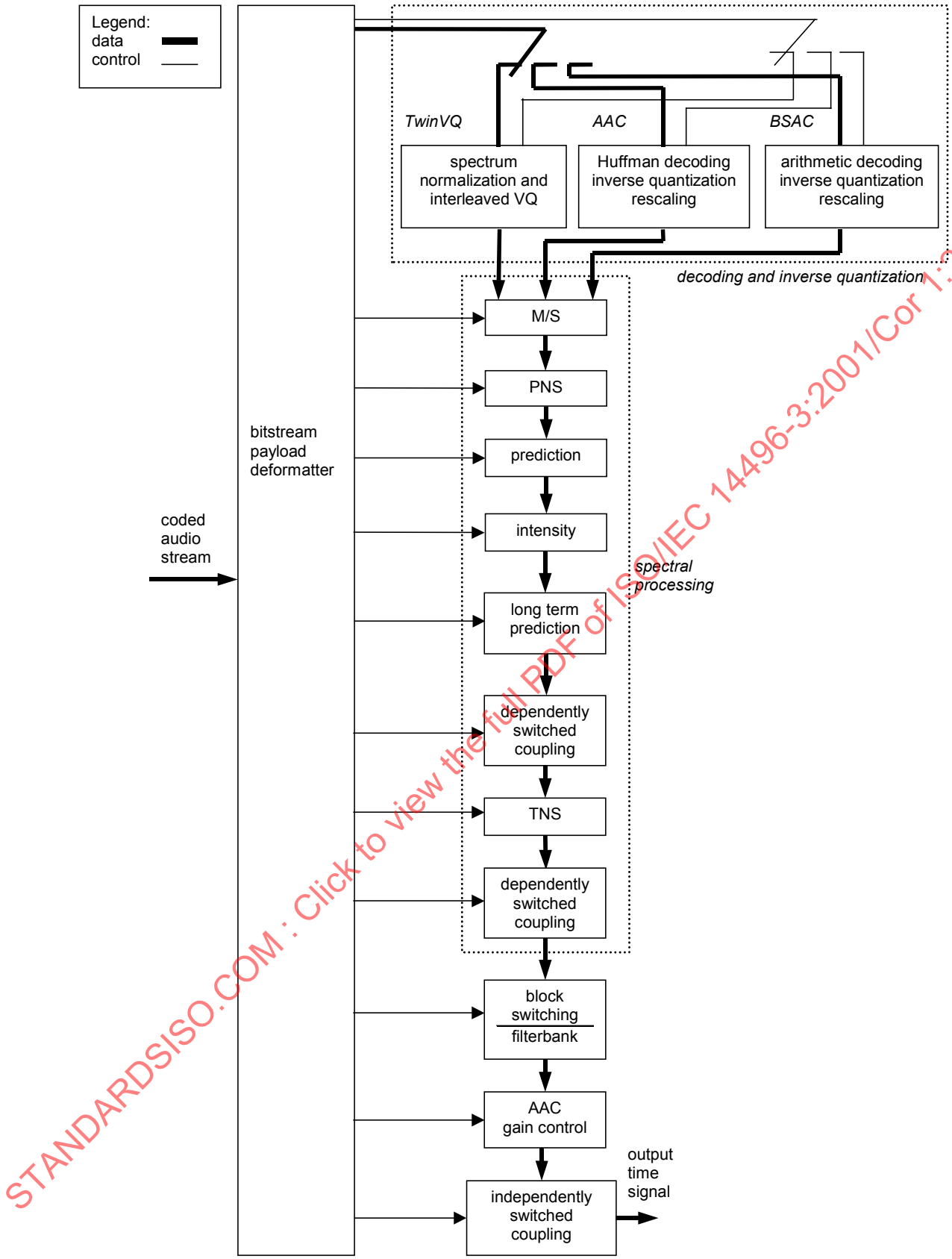
„



".

In subclause 4.1.1.1 (Encoder and Decoder Block Diagrams), replace Figure 4.2 (Block diagram of the GA non scalable decoder) with

”



”.

In subclause 4.1.1.2 (Overview of the Encoder and Decoder Tools), replace

„

The intensity stereo / coupling tool implements intensity stereo decoding on pairs of spectra. In addition, it adds the relevant data from a dependently switched coupling channel to the spectra at this point, as directed by the coupling control information.

The inputs to the intensity stereo / coupling tool are:

- The inversely quantized spectra
- The intensity stereo control information and coupling control information

The output from the intensity stereo / coupling tool is:

- The inversely quantized spectra after intensity and coupling channel decoding.

Note: If either part of this block is disabled, the scaled, inversely quantized spectra are passed directly through this part without modification. The intensity stereo tool and M/S tools are arranged so that the operation of M/S and Intensity stereo are mutually exclusive on any given scalefactor band and group of one pair of spectra.

”

with

„

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

The inputs to the intensity stereo tool are:

- The inversely quantized spectra
- The intensity stereo control information

The output from the intensity stereo tool is:

- The inversely quantized spectra after intensity channel decoding.

Note:The scaled, inversely quantized spectra of individually coded channles are passed directly through this tool without modification. The intensity stereo tool and M/S tool are arranged so that the operation of M/S and intensity stereo are mutually exclusive on any given scalefactor band and group of one pair of spectra.

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

The inputs to the coupling tool are:

- The inversely quantized spectra
- The coupling control information

The output from the coupling tool is:

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

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

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

The inputs to the coupling tool are:

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

The output from the coupling tool is:

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

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

”.

In subclause 4.1.1.2 (Overview of the Encoder and Decoder Tools), replace

„

scalefactor tool

"

with

„

rescaling tool

".

In subclause 4.1.1.2 (Overview of the Encoder and Decoder Tools), replace

„

The filterbank tool applies the inverse of the frequency mapping that was carried out in the encoder, as indicated by the filterbank control information and the presence or absence of gain control information. An inverse modified discrete cosine transform (IMDCT) is used for the filterbank tool. If the gain control tool is not used, the IMDCT input consists of either 1024 or 128 (depending on window_sequence spectral coefficients (if frameLengthFlag is set to '0'), or of 960 or 120 spectral coefficients (if frameLengthFlag is set to '1'), respectively. If the gain control tool is used, the filterbank tool is configured to use four sets of either 256 or 32 coefficients, depending of the value of window_sequence.

The inputs to the filterbank tool are:

- The inversely quantized spectra
- The filterbank control information

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

- The time domain reconstructed audio signal(s).

Two alternative, but very similar versions of this tool are available. The version with a frame length of 960 samples, which is not available in ISO/IEC 13818-7, allows for an integer frame length. For example at 48 kHz sampling rate, the frame length is exactly 20 ms with this version. This is especially useful for the CELP/AAC bitrate scalability combinations, where this allows the construction of combined CELP layer frames, which have a length of a multiple of 10 ms, and AAC enhancement layer frames. However, this feature can be used for configurations with only AAC or TwinVQ coding as well.

"

with

„

The filterbank / block switching tool applies the inverse of the frequency mapping that was carried out in the encoder. An inverse modified discrete cosine transform (IMDCT) is used for the filterbank tool. The IMDCT can be configured to support either one set of 120, 128, 480, 512, 960, or 1024, or four sets of 32 or 256 spectral coefficients.

The inputs to the filterbank tool are:

- The inversely quantized spectra
- The filterbank control information

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

- The time domain reconstructed audio signal(s).

".

In subclause 4.4.2.1 (Payloads for the audio object types AAC main, AAC SSR, AAC LC and AAC LTP), replace Table 4.3 (Syntax of top level payload for audio object types AAC Main, SSR, LC, and LTP (raw_data_block())) with

”

Syntax	No. of bits	Mnemonic

raw_data_block()
{
while((id = **id_syn_ele**) != ID_END){
switch (id) {
case ID_SCE: single_channel_element();
break;
case ID_CPE: channel_pair_element();
break;
case ID_CCE: coupling_channel_element();
break;
case ID_LFE: lfe_channel_element();
break;
case ID_DSE: data_stream_element();
break;
case ID_PCE: program_config_element();
break;
case ID_FIL: fill_element();
break;
}
}
byte_alignment();
}
	3	**uimsbf**

”.

In subclause 4.4.2.2 (Payloads for the audio object type AAC scalable), replace Table 4.13 (Syntax of the ASME top level payload for the audio object type AAC scalable (aac_scalable_main_element)) with

”

Syntax	No. of bits	Mnemonic

aac_scalable_main_element()
{
aac_scalable_main_header();
for (ch=0; ch<(this_layer_stereo ? 2:1); ch++){
individual_channel_stream(1, 1);
}
cnt = bits_to_decode() / 8;
while (cnt >= 1) {
cnt -= extension_payload(cnt);
}
byte_alignment();
}		

”.

In subclause 4.4.2.2 (Payloads for the audio object type AAC scalable), replace Table 4.14 (Syntax of the ASEE top level payload for the audio object type AAC scalable (aac_scalable_extension_element)) with

”

Syntax	No. of bits	Mnemonic

aac_scalable_extension_element()
{
aac_scalable_extension_header()
for (ch=0; ch<(this_layer_stereo ? 2:1); ch++){
individual_channel_stream(1, 1)
}
cnt = bits_to_decode() / 8
while (cnt >= 1) {
cnt -= extension_payload(cnt)
}
byte_alignment();
}		

”.

In subclause 4.4.2.3 (Payloads for the audio object types ER AAC LC, ER AAC LTP and ER AAC LD), replace Table 4.19 (Syntax of top level payload for audio object types ER AAC LC, ER AAC LTP and ER AAC LD (er_raw_data_block())) with

”

Syntax	No. of bits	Mnemonic

er_raw_data_block()
{
if (channelConfiguration == 0) {
/* reserved */
}
if (channelConfiguration == 1) {
single_channel_element ();
}
if (channelConfiguration == 2) {
channel_pair_element ();
}
if (channelConfiguration == 3) {
single_channel_element ();
channel_pair_element ();
}
if (channelConfiguration == 4) {
single_channel_element ();
channel_pair_element ();
single_channel_element ();
}
if (channelConfiguration == 5) {
single_channel_element ();
channel_pair_element ();
channel_pair_element ();
}
if (channelConfiguration == 6) {
single_channel_element ();
channel_pair_element ();
channel_pair_element ();
lfe_channel_element ();
}
if (channelConfiguration == 7) {
single_channel_element ();
channel_pair_element ();
channel_pair_element ();
channel_pair_element ();
lfe_channel_element ();
}		

```
    if (  channelConfiguration >= 8 ) {
        /* reserved */
    }
    cnt = bits_to_decode() / 8;
    while ( cnt >= 1 ) {
        cnt -= extension_payload(cnt);
    }
    byte_alignment();
}
```

”.

In subclause 4.4.2.7 (Subsidiary payloads), replace Table 4.47 (Syntax of scale_factor_data()) with

”

Syntax	No. of bits	Mnemonic
scale_factor_data()		
{		
if (! aacScalefactorDataResilienceFlag) {		
noise_pcm_flag = 1;		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = 0; sfb < max_sfb; sfb++) {		
if (sfb_cb[g][sfb] != ZERO_HCB) {		
if (is_intensity (g, sfb)) {		
hcod_sf[dpcm_is_position[g][sfb]];	**1..19**	**vlclbf**
}		
else {		
if (is_noise(g, sfb)) {		
if (noise_pcm_flag) {		
noise_pcm_flag = 0;		
dpcm_noise_nrg[g][sfb];	**9**	**uimsbf**
}		
else {		
hcod_sf[dpcm_noise_nrg[g][sfb]];	**1..19**	**vlclbf**
}		
}		
else {		
hcod_sf[dpcm_sf[g][sfb]];	**1..19**	**vlclbf**
}		
}		
}		
}		
}		
else {		
intensity_used = 0;		
noise_used = 0;		
sf_concealment;	**1**	**uimsbf**
rev_global_gain;	**8**	**uimsbf**
length_of_rvlc_sf;	**11/9**	**uimsbf**
for (g = 0; g < num_window_groups; g++) {		
for (sfb=0; sfb < max_sfb; sfb++) {		
if (sfb_cb[g][sfb] != ZERO_HCB) {		
if (is_intensity (g, sfb)) {		
intensity_used = 1;		
rvlc_cod_sf[dpcm_is_position[g][sfb]];	**1..9**	**vlclbf**
}		
else {		
if (is_noise(g,sfb)) {		
if (! noise_used) {		
noise_used = 1;		
dpcm_noise_nrg[g][sfb];	**9**	**uimsbf**

```
                        }
                        else {
                            rvlc_cod_sf[dpcm_noise_nrg[g][sfb]];        1..9    vlclbf
                        }
                    }
                    else {
                        rvlc_cod_sf[dpcm_sf[g][sfb]];                   1..9    vlclbf
                    }
                }
            }
        }
    }
    if ( intensity_used ) {
        rvlc_cod_sf[dpcm_is_last_position];                             1..9    vlclbf
    }
    noise_used = 0;
    sf_escapes_present;                                                 1       uimsbf
    if ( sf_escapes_present ) {
        length_of_rvlc_escapes;                                         8       uimsbf
        for ( g = 0; g < num_window_groups; g++ ) {
            for ( sfb = 0; sfb < max_sfb; sfb++ ) {
                if ( sfb_cb[g][sfb] != ZERO_HCB ) {
                    if ( is_intensity ( g, sfb ) &&
                        dpcm_is_position[g][sfb] == ESC_FLAG ) {
                        rvlc_esc_sf[dpcm_is_position[g][sfb]];          2..20   vlclbf
                    }
                    else {
                        if ( is_noise ( g, sfb ) {
                            if ( ! noise_used ) {
                                noise_used = 1;
                            }
                            else {
                                if (dpcm_noise_nrg[g][sfb] == ESC_FLAG ) {
                                    rvlc_esc_sf[dpcm_noise_nrg[g][sfb]];    2..20   vlclbf
                                }
                            }
                        }
                        else {
                            if (dpcm_sf[g][sfb] == ESC_FLAG ) {
                                rvlc_esc_sf[dpcm_sf[g][sfb]];           2..20   vlclbf
                            }
                        }
                    }
                }
            }
        }
        if ( intensity_used &&
            dpcm_is_last_position == ESC_FLAG ) {
            rvlc_esc_sf[dpcm_is_last_position];                         2..20   vlclbf
        }
    }
    if ( noise_used ) {
        dpcm_noise_last_position;                                       9       uimsbf
    }
  }
}
```

”.

In subclause 4.4.2.7 (Subsidiary payloads), replace Table 4.49 (Syntax of ltp_data()) with

”

Syntax	No. of bits	Mnemonic
ltp_data()		
{		
if (AudioObjectType == ER_AAC_LD) {		
ltp_lag_update;	**1**	**uimsbf**
if (ltp_lag_update) {		
ltp_lag;	**10**	**uimsbf**
} else {		
ltp_lag = ltp_prev_lag;		
}		
ltp_coef;	**3**	**uimsbf**
for (sfb = 0; sfb < min(max_sfb, MAX_LTP_LONG_SFB); sfb++) {		
ltp_long_used[sfb];	**1**	**uimsbf**
}		
}		
else {		
ltp_lag;	**11**	**uimsbf**
ltp_coef;	**3**	**uimsbf**
if(window_sequence==EIGHT_SHORT_SEQUENCE) {		
for (w=0; w<num_windows; w++) {		
ltp_short_used[w];	**1**	**uimsbf**
if (ltp_short_used [w]) {		
ltp_short_lag_present[w];	**1**	**uimsbf**
if (ltp_short_lag_present[w]) {		
ltp_short_lag[w];	**4**	**uimsbf**
}		
}		
}		
} else {		
for (sfb=0; sfb<min(max_sfb, MAX_LTP_LONG_SFB); sfb++) {		
ltp_long_used[sfb];	**1**	**uimsbf**
}		
}		
}		
}		

”.

In subclause 4.4.2.7 (Subsidiary payloads), replace table 4.51 (Syntax of extension_payload()) with

”

Syntax	No. of bits	Mnemonic
extension_payload(cnt)		
{		
extension_type;	**4**	**uimsbf**
align = 4;		
switch(extension_type) {		
case EXT_DYNAMIC_RANGE:		
return dynamic_range_info();		
case EXT_FILL_DATA:		
fill_nibble; /* must be ‘0000’ */	**4**	**uimsbf**
for (i=0; i<cnt-1; i++) {		
fill_byte[i]; /* must be ‘10100101’ */	**8**	**uimsbf**
}		
return cnt;		
case EXT_DATA_ELEMENT:		
data_element_version;	**4**	**uimsbf**
switch(data_element_version) {		
case ANC_DATA:		
loopCounter = 0;		
dataElementLength = 0;		
do {		
dataElementLengthPart;	**8**	**uimsbf**
dataElementLength += dataElementLengthPart;		
loopCounter++;		
} while (dataElementLengthPart == 255);		
for (i=0; i<dataElementLength; i++) {		
data_element_byte[i];	**8**	**uimsbf**
}		
return (dataElementLength+loopCounter+1);		
case default:		
align = 0;		
}		
case EXT_FIL:		
case default:		
for (i=0; i<8*(cnt-1)+align; i++) {		
other_bits[i];	**1**	**uimsbf**
}		
return cnt;		
}		
}		

”.

In 4.5.1.1 (GASpecificConfig()), replace

”

frameLengthFlag The window length of the IMDCT: if set to 0 a 1024 lines IMDCT is used and frameLength is set to 1024, if set to 1 a 960 line IMDCT is used and frameLength is set to 960.

”

with

”

frameLengthFlag Length of the frame, number of spectral lines, respectively.
For all General Audio Object Types except AAC SSR and ER AAC LD: If set to “0” a 1024/128 lines IMDCT is used and frameLength is set to 1024, if set to “1” a 960/120 line IMDCT is used and frameLength is set to 960.

For ER AAC LD: If set to “0” a 512 lines IMDCT is used and frameLength is set

to 512, if set to “1” a 480 line IMDCT is used and frameLength is set to 480.

For AAC SSR: Must be set to “0”. A 256/32 lines IMDCT is used.

Note: The actual number of lines for the IMDCT (first or second value) is determined by the value of window_sequence.

”.

In subclause 4.5.2.1.3.1 (Description), replace

”

It is not applicable to any program with other than the 3/2 configuration and only decoders capable of decoding a 3/2 configuration must be able to decode this mode.

”

with

”

It is not applicable to any program with other than the 3/2 configuration.

”.

In subclause 4.5.2.1.6 (Fill element (FIL)), replace table 4.59 with

”

Symbol	Value of extension_type	Purpose
EXT_FILL	‘0000’	bitstream filler
EXT_FILL_DATA	‘0001’	bitstream data as filler
EXT_DATA_ELEMENT	’0010‘	data element
EXT_DYNAMIC_RANGE	‘1011’	dynamic range control
-	all other values	reserved

”.

In subclause 4.5.2.1.6 (Fill element (FIL)), add the following text

”

Table 4.59a: Values of the data_element_version

Symbol	Value of data_element_version	Purpose
ANC_DATA	‘0000‘	Ancillary data element
-	all other values	Reserved

align: a helper variable indicating the amount of bits which are already processed to fulfill byte alignment requirements in extension payload

loopCounter: a helper variable indicating the length of the variable dataElementLength in bytes;

dataElementLength: a helper variable indicating the length of the extension payload ‘data element‘

dataElementLengthPart: a field indicating the length of the extension payload ‘data element‘. The value 255 is used as an escape value and indicates that at least one more dataElementLengthPart value is following. The overall length of the transmitted ‘data element‘ is calculated by summing up the partial values.

data_element_byte[ ]: a variable indicating the partial values of the extension payload ‘data element‘ with type ‘ANC_DATA‘ in bytes

”.

In subclause 4.5.2.2 (Payloads for the audio object type AAC scalable), replace all occurrences of

„
IFFS
”

with

„
FSS
”.

In subclause 4.5.2.2.1 (Definitions), replace

„

bits_to_decode()	a helper function; returns the number of bits not yet decoded in the current ASME, or ASEE, respectively, if the length of these elements has been signaled by a system/transport layer. If the length of these elements is unknown, bits_to_decode() returns 0.

”

with

„

bits_to_decode()	a helper function; returns the number of bits not yet decoded in the current top level payload if the length of that payload is signaled by a system/transport layer. If the length of the top level payload is unknown, bits_to_decode() returns 0.

”.

In subclause 4.5.2.2.5.2 (Frame length adaptation / super-frame), replace Table 4.61 with

„

Table 4.61 – AAC frame length for 960 samples per frame and super-frame length of AAC/CELP combinations

Sampling rate (kHz):	96	64	48	32	24	16	8
AAC Frame length (ms)	10	15	20	30	40	60	120
Super-Frame length (40 ms core frame) (ms)	40	120	40	120	40	120	120
AAC / CELP frames per super-frame (40 ms)	4 / 1	8 / 3	2 / 1	4 / 3	1 / 1	2 / 3	1 / 3
Super-Frame length (30 ms core frame) (ms)	30	30	60	30	120	60	120
AAC / CELP frames per super-frame (30 ms)	3 / 1	2 / 1	3 / 2	1 / 1	3 / 4	1 / 2	1 / 4
Super-Frame length (20 ms core frame) (ms)	20	60	20	60	40	60	120
AAC / CELP frames per super-frame (20 ms)	2 / 1	4 / 3	1 / 1	2 / 3	1 / 2	1 / 3	1 / 6
Super-Frame length (15 ms core frame) (ms)	30	15	60	30	120	60	120
AAC / CELP frames per super-frame (15 ms)	3 / 2	1 / 1	3 / 4	1 / 2	3 / 8	1 / 4	1 / 8
Super-Frame length (10 ms core frame) (ms)	10	30	20	30	40	60	120
AAC / CELP frames per super-frame (10 ms)	1 / 1	2 / 3	1 / 2	1 / 3	1 / 4	1 / 6	1 / 12

”.

In subclause 4.5.2.2.5.3 (CELP core coder with AAC running at 88.2 kHz, 44.1 kHz, or 22.05 kHz sampling rate), replace Table 4.62 with

„

Table 4.62 – Super-frame parameters of AAC/CELP combinations at AAC sampling rates of 88.2 kHz, 44.1 kHz and 22.05 kHz

Sampling rate AAC (kHz)	88.2	44.1	22.05
AAC Frame length (ms)	10.884	21.768	43.537
Super-frame length (43.537 ms core frame) (ms)	43.537	43.537	43.537
AAC / CELP frames per super-frame (43.537 ms)	4/1	2 / 1	1 / 1
Super-frame length (32.653 ms core frame) (ms)	32.653	65.306	130.612
AAC / CELP frames per super-frame (32.653 ms)	3/1	3 / 2	3 / 4
Super-frame length (21.768 ms core frame) (ms)	21.768	21.768	43.537
AAC / CELP frames per super-frame (21.768 ms)	2/1	1 / 1	1 / 2
Super-frame length (16.326 ms core frame) (ms)	32.653	65.306	130.612
AAC / CELP frames per super-frame (16.326 ms)	3 / 2	3 / 4	3 / 8
Super-frame length (10.884 ms core frame) (ms)	10.884	21.768	43.537
AAC / CELP frames per super-frame (10.884 ms)	1/1	1 / 2	1 / 4

".

In subclause 4.5.2.2.5.3 (CELP core coder with AAC running at 88.2 kHz, 44.1 kHz, or 22.05 kHz sampling rate), add

„

The possible sampling rates for the CELP core in this case are 7350 Hz (narrow-band core) or 14700 Hz (wide-band core).

".

In subclause 4.5.2.2.5.5 (Alternative core coder), replace

„

Within MPEG-4, only the MPEG-4 narrow-band CELP coder is available as a core coder.

"

with

„

Within MPEG-4, only the MPEG-4 CELP coder in a single layer configuration is available as core coder.

".

In subclause 4.5.2.2.7 (Combining AAC layers, if PNS, MS, or Intensity tools are used in a particular scale factor band), replace the tables "Stereo-stereo layer combination" and "Mono-stereo layer combination" with

“

Stereo-stereo layer combination:

Tool used in Layer N	Tool used in Layer N+1	Output of the combined Layer:
No Tool or MS	No Tool or MS	Sum of Layer N and Layer N+1
No Tool or MS	PNS	Invalid combination
No Tool or MS	Intensity	Invalid combination
No Tool or MS	PNS & Intensity	Invalid combination
PNS	No Tool	see subclause 4.6.13.6
PNS	MS	see subclause 4.6.13.6
PNS	Intensity	Layer N+1
PNS	PNS	Layer N+1
PNS	PNS & Intensity	Layer N+1
Intensity	No Tool or MS	Layer N+1
Intensity	PNS	Invalid combination
Intensity	Intensity	Sum of Layer N and Layer N+1, only M/L-channel; Take Positions from Layer N+1
Intensity	PNS & Intensity	Invalid combination
PNS & Intensity	No Tool or MS	Layer N+1
PNS & Intensity	PNS	Invalid combination
PNS & Intensity	Intensity	Layer N+1
PNS & Intensity	PNS & Intensity	Layer N+1

Mono-stereo layer combination:

Tool used in Layer N	Tool used in Layer N+1	Output of the combined Layer:
No Tool	No Tool	Sum of Layer N and Layer N+1 (FSS-Tool)
No Tool	MS	Sum of Layer N and Layer N+1
No Tool	PNS	Invalid combination
No Tool	Intensity	Layer N+1
No Tool	PNS & Intensity	Layer N+1
PNS	No Tool	Layer N+1
PNS	MS	Layer N+1
PNS	Intensity	Layer N+1
PNS	PNS	Layer N+1
PNS	PNS & Intensity	Layer N+1

”.

In subclause 4.5.2.4.1 (Error sensitivity category assignment), replace table 4.64 with

”

category	payload	mandatory	leads / may lead to one instance per	Description
0	main	yes	CPE	commonly used side information
1	main	yes	ICS	channel dependent side information
2	main	no	ICS	error resilient scale factor data
3	main	no	ICS	TNS data
4	main	yes	ICS	spectral data
5	extended	no	EPL	extension type / data_element_version
6	extended	no	EPL	DRC data
7	extended	no	EPL	bit stuffing
8	extended	no	EPL	ANC data

”.

In subclause 4.5.2.4.2 (Category instances and its dependency structure), replace table 4.65 (Dependency structure within the ER AAC payload) with

„

hierarchy level	error resilient multi-channel syntax	error resilience scalable syntax
frame / layer	base payload followed by extension payload	
base payload	order of syntactic elements follows order stated in Table 4.19	commonly used side information followed by channel dependent information
extended payload	no rule regarding the order of multiple EPLs is given, the kind of extension payload can be identified by extension_type	
syntactic element in base payload	commonly used side information followed by channel dependent information	-
channel dependent information	left channel followed by right channel	
channel dependent information / EPL	dependency structure according to category numbers	

Note: Channel dependent information consists of individual_channel_stream() (ICS). As an exception, ltp_data() is treated as channel dependent information even if it is not part of ICS.

".

In subclause 4.5.2.4.2 (Category instances and its dependency structure), add at the end

„

Figure 4.23 shows an example for the error resilient multi-channel syntax. The order of data elements inside each instance is based on the syntax description, i. e. the logical order is kept.

"

with

„

Figure 4.23 shows an example for the error resilient multi-channel syntax. The order of data elements inside each instance is based on the syntax description, i. e. the logical order is kept. The bits of the data function byte_alignment() terminating the AAC top level payloads are always assigned to the last instance of the depency chain and stored at its end.

".

In subclause 4.5.3.1 (Minimum decoder input buffer), replace

„

Minimum decoder input buffer:

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

The input buffer size is 6144 bits per SCE or independently switched CCE, plus 12288 bits per CPE. Both the total buffer and the individual buffer sizes are limited, so that the buffering limit can be calculated for either the entire bitstream, any entire program, or the individual audio elements permitting the decoder to break a multichannel bitstream into separate mono and stereo bitstreams which are decoded by separate mono and stereo decoders, respectively. Although the 6144 bit/CCE must be obeyed for dependent CCE's as well, any bits for dependent CCE's must be supplied from the total buffer requirements based on the independent CCE's, SCE's, and CPE's.

"

with

„

Minimum decoder input buffer:

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

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

For any error protected payload a supplementary decoder input buffer is specified. It is (N+5) % larger than the input buffer for the unprotected payload as specified above, where N is the value of "max. redundancy by class FEC" as specified in the level definition of any appropriate profile.
".

In subsection 4.5.3.2 (Bit Reservoir), replace

"

Bit reservoir:

The bit reservoir is controlled at the encoder. The maximum bit reservoir in the encoder depends on the number of channels and the mean bitrate. The maximum bit reservoir size for constant rate channels can be calculated by subtracting the mean number of bits per block from the minimum decoder input buffer size. For example, at 96 kbit/s for a stereo signal at 48 kHz sampling frequency the maximum bit reservoir size is 12288 bit- (96000 bit/s / 48000 1/s * 1024) = 10240 bit. For variable bitrate channels the encoder must operate in a way that the input buffer requirements do not exceed the minimum decoder input buffer.

The state of the bit reservoir is transmitted in the buffer_fullness field as the number of available bits in the bit reservoir divided by the number of audio channels divided by 32 and truncated to an integer value.

"

with

"

Bit reservoir:

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

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

The bit_reservoir_state of subsequent frames can be derived as follows:

$$bit\_reservoir\_state[frame] = bit\_reservoir\_state[frame-1] + mean\_framelength - framelength[frame]$$

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

$$0 \le bit\_reservoir\_state[frame] \le max\_bit\_reservoir$$

".

In subsection 4.5.3.3 (Maximum bit rate), replace

"

The maximum bit rate depends on the audio sampling rate. The maximum bit rate per channel can be calculated based on the minimum input buffer according to the formula: