INTERNATIONAL STANDARD ISO/IEC 14496-3:1999
TECHNICAL CORRIGENDUM 1

Published 2001-08-01

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 International Standard ISO/IEC 14496-3:1999 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 14496-3:1999/Cor.1:2001(E)**

Throughout the text of ISO/IEC 14496-3:1999 replace all occurrences of "AL PDU" *with* "SL packet" *and all occurrences of* "alPduPayload" *with* "slPacketPayload".

In subpart 1 replace all occurrences of " frameLength " *with* " frameLengthFlag ",

and in subpart 4 replace all occurrences of "FrameLengthFlag" *with* "frameLengthFlag".

In subclause 1.5.1, void tables 1.5.1 to 1.5 .4, and replace with

"

Table 1.5.1 — Audio Profiles

Audio Object Types	Speech Audio Profile	Synthesis Audio Profile	Scalable Audio Profile	Main Audio Profile
Null				
AAC LC			X	X
AAC main				X
AAC SSR				X
AAC LTP			X	X
AAC Scalable			X	X
TwinVQ			X	X
CELP	X		X	X
HVXC	X		X	X
TTSI	X	X	X	X
Main synthetic		X		X
Wavetable synthesis		(subset of Main synthetic)		(subset of Main synthetic)
General MIDI		(subset of Main synthetic)		(subset of Main synthetic)
Algorithmic Synthesis and Audio FX		(subset of Main synthetic)		(subset of Main synthetic)

".

In subclause 1.5.2, add "Audio" *to all profile names.*

In subclause 1.5.2, replace all " Synthesis Audio Profile" *with* " Synthetic Audio Profile".

In subclause 1.5.2.2, replace the first row of Table 1.5.6 – Complexity of Object Types with

"

Object Type	Parameters	PCU (MOPS per channel)	RCU (kWords per channel)	Remarks

".

Replace subclause 1.5.2.2 with

"

- **Levels for Synthetic Audio Profile**

Three levels are defined:

Synthetic Audio 1: All bitstream elements may be used with:

"Low processing" (exact numbers in ISO/IEC 14496-4:2000)

Only core sample rates may be used

No more than one TTSI object

Synthetic Audio 2: All bitstream elements may be used with:

"Medium processing" (exact numbers in ISO/IEC 14496-4:2000).

Only core sample rates may be used.

no more than four TTSI objects.

Synthetic Audio 3: All bitstream elements may be used with:

"High processing" (exact numbers in ISO/IEC 14496-4:2000).

No more than twelve TTSI objects.

- **Levels for Main Audio Profile**

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

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

Natural Audio 1: PCU < 40, RCU < 20

Natural Audio 2: PCU < 80, RCU < 64

Natural Audio 3: PCU < 160, RCU < 128

Natural Audio 4: PCU < 320, RCU < 256

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

Four Levels are then defined for Main Profile:

Natural Audio 1 + Synthetic Audio 1

Natural Audio 2 + Synthetic Audio 1

Natural Audio 3 + Synthetic Audio 2

Natural Audio 4 + Synthetic Audio 3

".

In subclause 1.5.2, add "Algorithmic synthesis and AudioFX object type "*in Object Type definitions for Audio and in the Profiles and Levels table (Table 1.5.6 Complexity of Object Types).*

Replace Table 1.5.6 with

"

The following table gives complexity estimates for the different object types and Sampling Rate conversion:

Table 1. 5. 6 - Complexity of Object Types and SR conversion

Object Type	Parameters	PCU (MOPS per channel)	RCU (kWords per channel)	Remarks
AAC Main	fs = 48 kHz	5	5	1)
AAC LC	fs = 48 kHz	3	3	1)
AAC SSR	fs = 48 kHz	4	3	1)
AAC LTP	fs = 48 kHz	4	4	1)
AAC Scalable	fs = 48 kHz	5	4	1), 2)
TwinVQ	fs = 24 kHz	2	3	1)
CELP	fs = 8 kHz	1	1	
CELP	fs = 16 kHz	2	1	
CELP	fs = 8/16 kHz (bandwidth scalable)	3	1	
HVXC	fs = 8 kHz	2	1	
TTSI		-	-	4)
General MIDI		4	1	
Wavetable Synthesis	fs = 22.05 kHz	depends on bitstreams (3)	depends on bitstreams (3)	
Main Synthetic		depends on bitstreams (3)	depends on bitstreams (3)	
Algorithmic Synthesis and AudioFX		depends on bitstreams (3)	depends on bitstreams (3)	
Sampling Rate Conversion	rf = 2, 3, 4, 6	2	0.5	

Definitions:

fs = sampling frequency

rf = ratio of sampling rates

Notes -

1) PCU proportional to sampling frequency.

2) Includes core decoder.

3) See ISO/IEC 14496-4:2000.

4) The complexity for speech synthesis is not taken into account.

".

In subclause 1.6.2, replace all "AudioSpecificInfo()" *with* "AudioSpecificConfig()".

To the end of subclause 1.6.2.7, add

"

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

".

In subclause 1.6.3.3, replace the table header of Table 1.6.2 with

"

samplingFrequencyIndex	Value

".

Remove subclause 1.A.2.3 "MPEG-4 Audio Transport Stream (MATS)" .

In subclause 2.3.1, replace

"

HVXC Base Layer –Configuration

For HVXC object type in unscalable mode or as the base layer in scalable mode requires the following HvxcSpecificConfig() required:

```
    HvxcSpecificConfig() {
        HVXCconfig();
    }
```

HVXC Enhancement Layer –Configuration

HVXC object type provides a 2kbit/s base layer plus a 2kbit/s enhancement layer scalable mode. In this scalable mode the basic layer configuration must be as follows:

HVXCvarMode = 0	HVXC fixed bit rate
HVXCrateMode = 0	HVXC 2kbps

For the enhancement layer, there is no HvxcSpecificConfig() required:

```
    HvxcSpecificConfig() {
    }
```

"

with

"

The following HvxcSpecificConfig() is required:

```
HvxcSpecificConfig ( ) {
    isBaseLayer                              1     uimsbf
    if (isBaseLayer) {
        HVXCconfig()
    }
}
```

HVXC object type provides unscalable modes and a 2kbit/s base layer plus a 2kbit/s enhancement layer scalable mode. In this scalable mode the basic layer configuration must be as follows:

HVXCvarMode = 0	HVXC fixed bit rate
HVXCrateMode = 0	HVXC 2kbps
isBaseLayer=1	base layer

",

and at the end of subclause 2.4.1, add

"

isBaseLayer A one-bit identifier representing whether the corresponding layer is the base layer (1) or the enhancement layer (0).

".

In subclause 2.5.3.3, add

"

If the pitch modification is controlled by the **pitch** field in the AudioSource BIFS node, the modification factor is:

pch_mod = pitch

"

after

"

Pitch modification can be done by dividing pch by pitch modification factor pch_mod:

$$pch = pch / pch\_mod$$

".

In subclause 2.5.5.3, add the sentence

"

If the speed is controlled by the time scaling factor in the **speed** field of the AudioSource BIFS node, the speed change ratio is:

spd = 1 / speed

"

after

"

where N_1 is the duration of the original speech and N_2 is the duration of the speed controlled speech. Therefore, $0 \le n < N_1$ and $0 \le m < N_2$.

".

In subclause 3.3, replace the following paragraphs:

"

CelpSpecificConfig()

CELP Base Layer

The CELP core in the unscalable mode or as the base layer in the scalable mode requires the following CelpSpecificConfig():

```
class CelpSpecificConfig (uint(4) samplingFrequencyIndex ) {
        CelpHeader (samplingFrequencyIndex);
}
```

CELP Enhancement Layer

The CELP core is used for both bitrate and bandwidth scalable modes. In the bitrate scalable mode, the enhancement layer requires no CelpSpecificConfig(). In the bandwidth scalable mode, the enhancement layer has the following CelpSpecificConfig():

```
class CelpSpecificConfig() {
        CelpBWSenhHeader();
}
```

"

with

"

CelpSpecificConfig()

The following CelpSpecificConfig() is required:

```
class CelpSpecificConfig (uint(4) samplingFrequencyIndex ) {
   isBaseLayer                1   uimsbf
   if (isBaseLayer) {
      CelpHeader(samplingFrequencyIndex)
   } else {
      isBWSLayer              1   uimsbf
      if (isBWSLayer) {
         CelpBWSenhHeader()
      } else {
         CELP-BRS-id          2   uimsbf
      }
   }
}
```

",

and at the end of subclause 3.3.4, add

"

isBaseLayer see subclause 2.4.1 of subpart 2.

isBWSLayer A one-bit identifier representing whether the corresponding layer is the bandwidth scalable enhancement layer (1) or the bitrate scalable enhancement layer (0).

CELP-BRS-id A two-bit identifier representing the order of the bitrate scalable enhancement layers, where the first enhancement layer has the value of '1'. The value of '0' should not be used.

".

In subclause 4.4.2 replace Table 4.4.28

"

Syntax	No. of bits	Mnemonic
ltp_data()		
{		
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<max_sfb; sfb++) {		
ltp_long_used[sfb]	**1**	**uimsbf**
}		
}		
}		

"

with

"

Syntax	No. of bits	Mnemonic
ltp_data()		
{		
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<max_sfb; sfb++) {		
ltp_long_used[sfb]	**1**	**uimsbf**
}		
}		
}		

".

In subclause 4.4, remove

"

Two types of data are part of the MPEG-4 GA coder syntax. These are

1. Configuration information
2. Actual Payload

The payload is intended to be transported via the MPEG-4 Systems layer. These data contain all information variing on a frame to frame basis, and therefore carry the actual audio information.

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

The configuation information and the payload are abstract elements which define all information for the decoding and parsing of the bitstream. However, for real applications these streams need a transport layer which cares for the delivery of these elements. Normally, this transport mechanism will be handled by MPEG-4 Systems. However, the interface format streams defined in the Annex A of subpart 1 define a simple way of multiplexing the header and the raw data streams.

".

In subclause 4.4.1, 4.5.1, 4.5.2 and 4.6.14, replace

"GASpecificConfiguration()", "GA_SpecificConfig", *and* "GA_SpecificConfig()"

with

"GASpecificConfig()".

In subclause 4.4.1, replace the heading

"GA Specific configuration"

with

"Decoder configuration (GASpecificConfig)".

In table 4.15 (Syntax of aac_scalable_main_header()) in subclause 4.4.2,

replace the term "tvq_layer_pesent"

with

"tvq_layer_present".

In subclause 4.5.1.1, replace the description

"

ExtensionFlag: Set to '0' in MPEG-4 Phase 1. Set to ' 1' in MPEG-4 Phase 2.

"

with

"

ExtensionFlag: Shall be '0' for audio object types 1, 2, 3, 4, 6, 7. Shall be '1' for audio object types 17, 19, 20, 21, 22, 23.

".

At the end of subclause 4.5.1.1, add

"

Restriction:

An MPEG-4 Audio decoder is only required to follow the Program Configuration Element in GASpecificConfig(). The decoder shall ignore any Program Configuration Elements that may occur in raw data blocks. PCEs transmitted in raw data blocks cannot be used to convey decoder configuration information.

",

and in subclause 4.5.1.2.1, replace

"

For more complicated configurations a **Program Configuration Element** (PCE) is defined. There are 16 available PCE' s, and each one can specify a distinct program that is present in the raw data stream. All available PCE' s within a raw_data_block must come before all other syntactic elements. Programs may or may not share audio syntactic elements, for example, programs could share a channel_pair_element and use distinct coupling channels for voice over in different languages. A given program configuration element contains information pertaining to only one program out of many that may be included in the raw data stream. Included in the PCE are „list of front channels", again using the rule center outwards, left before right. In this list, a center channel SCE, if any, must come first, and any other SCE' s must appear in pairs, constituting an LR pair. If only two SCE' s are specified, this signifies one LR stereophonic pair.

After the list of front channels, there is a list of "side channels" consisting of CPE' s, or of pairs of SCE' s. These are listed in the order of front to back. Again, in the case of a pair of SCE' s, the first is a left channel, the second a right channel.

After the list of side channels, a list of back channels is available, listed from outside in. Any SCE' s except the last

SCE must be paired, and the presence of exactly two SCE' s (alone or preceeded by a CPE) indicates that the two

SCE' s are Left and Right Rear center, respectively.

The configuration indicated by the PCE takes effect at the raw_data_block containing the PCE. The number of front, side and back channels as specified in the PCE must be present in that block and all subsequent raw_data_blocks until a raw_data_block containing a new PCE is transmitted.

Other elements are also specified. A list of one or more LFE' s is specified for application to this program. A list of one or more CCE' s is also provided, in order to allow for dialog management as well as different intensity coupling streams for different channels using the same main channels. A list of data streams associated with the program can also associate one or more data streams with a program. The program configuration element also allows for the specification of one monophonic and one stereophonic simulcast mixdown channel for a program.

Note that the MPEG-4 Systems standard supports alternate methods of simulcast.

The PCE element is not intended to allow for rapid program changes. At any time when a given PCE, as selected by its element_instance_tag, defines a new (as opposed to repeated) program, the decoder is not obliged to provide audio signal continuity.

"

with

"

For more complicated configurations a **Program Configuration Element** (PCE) is defined. The same restrictions apply with respect to the PCE as defined in ISO/IEC 14496-3:1999. However, an MPEG-4 decoder is only required to parse PCEs in raw_data_blocks(), without interpreting them. Only the PCE provided within GASpecificConfig() describes the decoder configuration for the elementary stream under consideration. This implies that only one program can be configured at a certain time.

".

In subclause 4.5.1.1, replace

"

If the sampling rate is not one of the rates listed in the right column in the table below, the sampling frequency dependent tables (code tables, scale factor band tables etc.) must be deduced in order for the bit stream to be parsed. Since a given sampling frequency is associated with only one sampling frequency table, and since maximum flexibility is desired in the range of possible sampling frequencies, the following table shall be used to associate an implied sampling frequency with the desired sampling frequency dependent tables. However, there is one exception to this rule, which is described in subclause 4.6.13.1 for Table 4.6.12.

Table 4.5.1

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

"

with

"

If the sampling rate is not one of the rates listed in the right column in Table 4.5.1, the sampling frequency dependent tables (code tables, scale factor band tables etc.) must be deduced in order for the bit stream to be parsed. Since a given sampling frequency is associated with only one sampling frequency table, and since maximum flexibility is desired in the range of possible sampling frequencies, the following table shall be used to associate an implied sampling frequency with the desired sampling frequency dependent tables. However, there is one exception to this rule, which is described in subclause 4.6.13.1 for Table 4.6.12.

Table 4.5.1 Sampling frequency mapping

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

If a certain sampling frequency dependent table stated in the right column of Table 4.5.1 is not defined, the nearest defined table shall be used.

".

In subclause 4.5.2.1.1, replace

"

raw_data_block(:) block of raw data that contains audio data for a time period of 1024 or 960 samples, related information and other data. There are 8 bitstream elements, identified as bitstream element id_syn_ele. The audio elements in one raw data stream and one raw data block must have one and only one sampling rate. In the raw data block, several instances of the same id_syn_ele may occur, but each such instance of an id_syn_ele except for a data_stream_element must have a different 4-bit element_instance_tag. Therefore, in one raw data block, there can be from 0 to at most 16 of any id_syn_ele. The exceptions to this are the data_stream_element, the fill_element and the terminator element. If multiple data stream elements occur which have unique element_instance_tags then they are part of distinct data streams. If multiple data stream elements occur which have the same element_instance_tag then they are part of the same data stream. The fill_element has no element_instance_tag (since the content does not require subsequent reference) and can occur any number of times. The terminator element has no element_instance_tag and must occur exactly once, as it marks the end of the raw_data_

"

with

"

raw_data_block(): block of raw data that contains audio data for a time period of 1024 or 960 samples, related information and other data. There are 8 syntactic elements, identified as syntactic element id_syn_ele. The audio elements in one raw data block must have one and only one sampling rate. In the raw data block, several instances of the same id_syn_ele may occur, but each such instance of an id_syn_ele except for a data_stream_element must have a different 4-bit element_instance_tag. Therefore, in one raw data block, there can be from 0 to at most 16 of any id_syn_ele. The exceptions to this are the data_stream_element, the fill_element and the terminator element. If multiple data stream elements occur which have unique element_instance_tags then they are part of distinct data streams. If multiple data stream elements occur which have the same element_instance_tag then they are part of the same data stream. The fill_element has no element_instance_tag (since the content does not require subsequent reference) and can occur any number of times. The terminator element has no element_instance_tag and must occur exactly once, as it marks the end of the raw_data_block**.**

".

In subclause 4.5.2.2.4, replace

"

For all scale factor bands where M/S or Intensity coding is selected, the M''-Signal is calculated by adding M'' and M' (The restrictions given in subclause 5.2.2.7 have to be followed which prohibit the addition under certain circumstances).

"

with

"

For all scale factor bands where M/S coding is selected, the M-Signal is calculated by adding M'' and M' (The restrictions given in subclause 5.2.2.7 have to be followed which prohibit the addition under certain circumstances).

".

In Table 5.7 (raw 1, column 1) in subclause 4.5.2.2.5.2, replace

"Sampling rate (Hz) " *with* " Sampling rate (kHz) ".

Replace complete subclause 4.5.2.2.5.3 (including Table 4.5.8) with

"

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

AAC frames using sampling rates of 88.2 kHz, 44.1 kHz, or 22.05 kHz can be achieved by adjusting the sampling rate of the CELP core coder such, that an integer ratio between these two sampling rate is achieved. Table 4.6.12 shows the mapping of the AAC sampling rates to the CELP core coder sampling rates. The CELP core coder runs with the sampling rate listed in this table. The CELP decoding process is completely identical to the methods defined for a sampling rate of 8 kHz for the narrow band CELP coder. Table 4.5.8 shows the super frame parameters for the AAC sampling rates 88.2 kHz, 44.1 kHz, and 22.05 kHz.

Table 4.5.8 – 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
Sampling rate CELP (Hz)	7350	7350	7350
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 (30 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 (20 ms)	2/1	1 / 1	1 / 2
Super-frame length (10.884ms core frame) (ms)	10.884	21.768	43.537
AAC / CELP frames per super-frame (10 ms)	1/1	1 / 2	1 / 4

".

Replace heading and content of subclause 4.5.2.2.7 (Combining AAC and TwinVQ layer, if PNS, MS, or Intensity tools are used in a particular scale factor band) with

".

4.5.2.2.7 Combining AAC layers, if PNS, MS, or Intensity tools are used in a particular scale factor band

The following tables specify the output spectrum of a particular scale factor band of the combined layers N and N+1 for various combinations of the PNS, Intensity, and MS coding tools in layer N and layer N+1 for different layer combinations:

Mono-mono layer combination:

Tool used in Layer N	Tool used in Layer N+1	Output of the combined Layers:
No Tool	No Tool	Sum of Layer N and Layer N+1
No Tool	PNS	Invalid combination
PNS	No Tool	see subclause 6.12.6
PNS	PNS	Layer N+1

Stereo-stereo layer combination:

Tool used in Layer N	Tool used in Layer N+1	Output of the combined Layers:
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 6.12.6
PNS	MS	Layer N+1
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	Invalid combination
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 Layers:
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	see subclause 6.12.6
PNS	MS	Layer N+1
PNS	Intensity	Layer N+1
PNS	PNS	Layer N+1
PNS	PNS & Intensity	Layer N+1

".

In subclause 4.5.2.4.2, replace

"

N_DIV_P =2

"

with

"

N_DIV_P = 2* N_CH

",

In subclause 4.5.2.4.3.1, replace

"

Table 4.5.10 - Bit allocation of syntax elements

Name of variables	Name of number of bits	**lyr** = 0 LONG	**lyr** = 0 SHORT	**lyr** >= 1 LONG	**lyr** >= 1 SHORT	Times per frame
fb_shift	-	0	0	2	2	1
index_blim_h	-	2/0	2/0	0	0	1
index_blim_l	-	1/0	1/0	0	0	1
index_env	FW_N_BIT	6	0	6	0	FW_N_DIV
index_fw_alf	-	1	0	1	0	1
index_gain	GAIN_BIT	9	9	8	8	1
index_gain_sb	SUB_GAIN_BIT	0	4	0	4	N_SF
index_lsp0	LSP0_BIT	1	1	1	1	1
index_lsp1	LSP1_BIT	6	6	6	6	1
index_lsp2	LSP2_BIT	4	4	4	4	1
index_shape0_p	MAXBIT_P	7/0	0	0	0	N_DIV_P
index_shape1_p	MAXBIT_P	7/0	0	0	0	N_DIV_P
index_pit	BASF_BIT	8/0	0	0	0	1
index_pgain	PGAIN_BIT	7/0	0	0	0	1

"

with

"

Table 4.5.10 - Bit allocation of syntax elements

Name of variables	Name of number of bits	**lyr** = 0 LONG	**lyr** = 0 SHORT	**lyr** >= 1 LONG	**lyr** >= 1 SHORT	Times per frame
fb_shift	-	0	0	2	2	N_CH
index_blim_h	-	2/0	2/0	0	0	N_CH
index_blim_l	-	1/0	1/0	0	0	N_CH
index_env	FW_N_BIT	6	0	6	0	FW_N_DIV*N_CH
index_fw_alf	-	1	0	1	0	N_CH
index_gain	GAIN_BIT	9	9	8	8	N_CH
index_gain_sb	SUB_GAIN_BIT	0	4	0	4	N_SF*N_CH
index_lsp0	LSP0_BIT	1	1	1	1	N_CH
index_lsp1	LSP1_BIT	6	6	6	6	N_CH
index_lsp2	LSP2_BIT	4	4	4	4	LSP_SPLIT*N_CH
index_shape0_p	MAXBIT_P+1	7/0	0	0	0	N_DIV_P
index_shape1_p	MAXBIT_P+1	7/0	0	0	0	N_DIV_P
index_pit	BASF_BIT	8/0	0	0	0	N_CH
index_pgain	PGAIN_BIT	7/0	0	0	0	N_CH

",

and in subclause 4.5.2.4.3.2, replace

"

```
if (ppc_present == TRUE}
    PIT_TBIT = PIT_N_BIT + (BASF_BIT + PGAIN_BIT) * N_CH;
else
    PIT_TBIT = 0;
```

"

with

"

```
if (ppc_present == TRUE)
    PIT_TBIT = (MAXBIT_P+1)*N_DIV_P*2+ (BASF_BIT + PGAIN_BIT) * N_CH;
else
    PIT_TBIT = 0;
```

".

In subclause 4.5.2.4.3.2, replace

"

available_vq =

(int)(FRAME_SIZE * BITRATE/SANPLING_FREQUENCY)-bits_for_side_information

"

with

"

bits_available_vq =

(int)(((FRAME_SIZE * bitrate/sampling_frequency)/8+0.5)*8) - bits_for_side_information,

where bitrate is given by a system parameter in [bit/s] and sampling frequency is given in the right column of table 4.5.1.

".

In subclause 4.5.4.4.4, replace

"

ISAMPF is an integer sampling frequency in [kHz]

"

with

"

ISAMP is an integer sampling frequency in [kHz] truncated from the standard frequency values listed in the right column of table 5.1 in subpart 4.

".

In subclause 4.6.4.2, replace

"

sp_cv0[][] shape codebook of conjugate channel 0

sp_cv1[][] shape codebook of conjugate channel 1

"

with

"

sp_cv0[][] shape codebook of conjugate channel 0 (Elements are given in tables 4.A.19, 21, 23, 25.)

sp_cv1[][] shape codebook of conjugate channel 1 (Elements are given in tables 4.A. 20, 22, 24, 26.)

",

and in subclause 4.6.9.2, replace

"

sp_cv0[] reconstructed shape of conjugate channel 0 for periodic peak components quantization

sp_cv1[] reconstructed shape of conjugate channel 1 for periodic peak components quantization

"

with

"

pit_cv0[] reconstructed shape of conjugate channel 0 for periodic peak components quantization (Elements are given in the first 64 rows of table 4.A.28.)

pit_cv1[] reconstructed shape of conjugate channel 1 for periodic peak components quantization (Elements are given in the last 64 rows of table 4.A.28.)

".

Also in subclause 4.6.9.3.4.1, replace

"

The cv_env[][] is the Bark-scale envelope codebook.

"

with

"

The cv_env[][] is the Bark-scale envelope codebook listed in 4.A.27.

".

In subclause 4.6.4.4.1, replace

"

Number of available bits, bits_available_vq, is calculated as follows:

bits_available_vq =

(int)(FRAME_SIZE * bitrate/sampling_frequency) - bits_for_side_information,

where bitrate is represented in [bits/s] and sampling_frequency is represented in [Hz].

N_DIV and the length of each subvector, length[], is calculated by

"

with

"

Based on the bits_available_vq defined in 4.5.2.4.3.2,

N_DIV and the length of each subvector, length[], are calculated by

".

And in 4.6.4.4.4, replace

"

BPS is bitrate in [bits/s/ch].

"

with

"

BPS is bitrate in [bits/s/ch] based on the byte-aligned bits for a frame and it equals

(int)(((FRAME_SIZE * bitrate/sampling_frequency)/8+0.5)*8)/sampling_frequency/FRAME_SIZE/N_CH.

",

and replace

"

the value of bitrate per channel BPS_SCL[lyr] in [bits/s/ch] for each enhancement layer.

"

with

"

the value of bitrate per channel BPS_SCL[lyr] in [bits/s/ch] for each enhancement layer. Note that BPS_SCL[lyr] should be based on the byte-aligned bits for a frame and it equals

(int)(((FRAME_SIZE * bitrate/sampling_frequency)/8+0.5)*8)/sampling_frequency/FRAME_SIZE/N_CH.

".

In subclause 4.6.4.4.4, replace

"

LOWER_BOUNDARY[lyr][i_ch] = AC_BTM[lyr][i_ch] * N_FR

UPPER_BOUNDARY[lyr][i_ch] = AC_TOP[lyr][i_ch] * N_FR;

"

with

"

LOWER_BOUNDARY[lyr][i_ch] = AC_BTM[lyr][i_ch][**fb_shft**] * N_FR;

UPPER_BOUNDARY[lyr][i_ch] = AC_TOP[lyr][i_ch][**fb_shift**] * N_FR;

".

In subclause 4.6.4.4.4, replace

"

qsample = min(1.0, 0.95*BPS/1000./(double)ISAMPF);

"

with

"

bandUpper_i = 95 * BPS/ ISAMPF;

bandUpper_i = min(100000, bandUpper_i);

bandUpper_i *= 16384;

bandUpper_i += 1562

bandUpper_i /= 3125;

qsample = (double)(bandupper_i)/524288.;

",

replace

"

upperlimit = min(1.0, totalbps / 1000.0 / (double)ISAMPF);

"

with

"

upperlimit_i = (totalbps* 100) / ISAMPF;

upperlimit_i = min(100000, upperlimit_i);

upperlimit_i *= 16384;

upperlimit_i += 1562;

upperlimit_i /= 3125;

upperlimit = (double)(upperlimit_i)/524288.;

",

and replace

"

qsample= min(1.0, BPS_SCL[lyr]/1000./(double)ISAMPF * 1.3);

bias = (upperlimit - qsample) /4;

```
if (qsample < bias){
   bias = upperlimit/4;
   qsample = bias;
}
```

"

with

"

```
qsample_i = (BPS_SCL[lyr]* 130) / ISAMPF;
qsample_i = min(100000, qsample_i);
qsample_i *= 16384;
qsample_i += 1562;
qsample_i /= 3125;
bias_i = (upperlimit_i- qsample_i)/4;
if(qsample_i < bias_i){
  bias_i = upperlimit_i/4;
  qsample_i = bias_i;
}
bias = (double) bias_i;
qsample = (double)(qsample_i)/524288./* 16384*32 */
```

".

In subclause 4.6.6.3, replace

"

For long windows, the LTP parameters are used to calculate the predicted time domain signals using the following formula:

$$x\_est(i) = \text{ltp\_coef} * x\_rec(i - \text{ltp\_lag}),$$
$$i = 0,...,\text{ltp\_lag}$$

where $x\_est(i)$ are the predicted samples

$x\_rec(i)$ are reconstructed time domain samples

"

with

"

For long windows, the LTP parameters are used to calculate the predicted time domain signals using the following formula:

$$x\_est(i) = \text{ltp\_coef}*x\_rec(i - \text{M} - \text{ltp\_lag})$$

i = 0,…,N-1

where $x\_est(i)$ are the predicted samples

$x\_rec(i)$ are reconstructed time domain samples

N is the length of the transform window

M = N/2 in the AAC Low Delay profile, otherwise M = 0

The different value for M used in the Low Delay profile is to achieve a similar range of possible lag values in absolute time, despite of the shorter frame.

The reference point for index *i* and the content of the buffer *x_rec* are arranged so that *x_rec*(0 ... N/2 - 1) contains the last aliased half window from the IMDCT, and *x_rec*(N/2 ... N-1) is always all zeros. The rest of *x_rec* (*i*<0) contains the previous fully reconstructed time domain samples, i.e., output of the decoder.

".

In subclause 4.6.6.4, replace

"

In case both LTP, and PNS are enabled on the same scalefactor band, LTP takes precedence, and no noise is added to this band.

"

with

"

If both LTP, and PNS are enabled on the same scalefactor band, PNS takes precedence, and no prediction is applied to this band.

".

In subclause 4.6.7.2.4, replace

"

The function of Long Term Prediction does not depend on Intensity Stereo.

"

with

"

In case of a non-scalable configuration the function of Long Term Prediction does not depend on Intensity Stereo.

",

in subclause 4.6.7.2.5, add

"

In a scalable configuration the simultaneous use of Intensity Stereo and LTP is not prevented in the syntax. However if both Intensity Stereo and LTP are enabled in the same scalefactor band in the first GA layer, Intensity Stereo takes precedence and no prediction is applied to this band.

",

and in subclause 4.6.6.5, add

"

In a scalable configuration the simultaneous use of LTP and Intensity Stereo is not prevented in the syntax. However if both LTP and Intensity Stereo are enabled in the same scalefactor band in the first GA layer, Intensity Stereo takes precedence and no prediction is applied to this band.

".

In subclause 4.6.7.2.3, replace

"

In addition, the phase relationship of the Intensity Stereo coding can be reversed by means of the ms_used field:

"

with

"

In addition, in case of a non-scalable GA decoder the phase relationship of the Intensity Stereo coding can be reversed by means of the ms_used field:

",

in subclause 4.6.7.2.3, replace

"

```
function invert_intensity(group,sfb)   {
   1-2*ms_used[group][sfb]    if (ms_mask_present == 1)
   +1                                  otherwise
}
```

"

with

"

```
function invert_intensity(group,sfb)   {
   1-2*ms_used[group][sfb]    if (ms_mask_present == 1) && aot != AAC_scalable
   +1                         otherwise
}
```

",

in subclause 4.6.7.2.5, add

"

In case of an AAC scalable configuration the ms_used field is ignored in Intensity Stereo decoded scalefactor bands but may still signal the use of M/S stereo decoding in higher (enhancement) layers.

",

and in subclause 4.6.7.2.5, replace

"

(i.e. a Huffman codebook != INTENSITY_HCB?).

"

with

"

(i.e. a Huffman codebook != INTENSITY_HCB).

".

In subclause 4.6.8.4, replace

"

For the AAC SSR audio object type the parameter TNS_MAX_ORDER is 12. For all other object types using TNS, the value for the constant TNS_MAX_ORDER is set as follows:

For long windows: TNS_MAX_ORDER is 20 for sampling rates of 32 kHz and below. For sampling rates above 32 kHz it is 12.

For short windows: TNS_MAX_ORDER is 7."

"

with

"

The value for the constant MAX_TNS_ODER depends on audio object type, windowing, and sampling rate. The following table defines MAX_TNS_ORDER depending on these parameters.

Definition of TNS_MAX_ORDER depending on AOT, windowing, and sampling rate

Windowing	short windows	long windows	long windows
sampling rate	-	> 32kHz	<= 32kHz
AOT 1 (AAC Main)	7	20	20
AOT 2 (AAC LC)	7	12	12
AOT 3 (AAC SSR)	7	12	12
other AOT using TNS	7	20	12

″.

Replace table 4.6.4 of sublcause 4.6.8.5 with:

tns_data_present M-Channel	tns_data_present L-Channel	tns_data_present R-Channel	TNS Info M Source Chan.	TNS Info L Source Chan.	TNS Info R Source Chan.
0	0	0	-	-	-
1	0	0	M	M	M
0	1	1	-	L	R
0	1	0	-	L	-
0	0	1	-	-	R
1	0	1	M	M	R/M
1	1	0	M	L/M	M
1	1	1	M	L/M	R/M

In subclause 4.6.9.3.3.3, replace

″

```
for (idiv=0; idiv<N_DIV_P; idiv++){
    for (icv=0; icv<lengthp[idiv]; icv++){
        ismp = idiv + icv * N_DIV_P;
        pit[ismp] = gain_p * (pol0[idiv]*pit_cv0[index0[idiv][icv]] + pol1[idiv]*pit_cv1[index1[idiv]][icv]) / 2;
    }
}
```

″

with

″

```
for (idiv=0; idiv<N_DIV_P; idiv++){
    if(N_CH==1) {
        for (icv=0; icv<lengthp[idiv]; icv++){
            ismp = idiv + icv * N_DIV_P;
            pit[ismp] =  (pol0[idiv]*pit_cv0[index0[idiv][icv]] + pol1[idiv]*pit_cv1[index1[idiv]][icv]) / 2;
        }
    }
    else{
        for (icv=0; icv<lengthp[idiv]-1; icv++){
            ismp = ((icv+idiv)%N_DIV_P)+ icv * N_DIV_P;
            ismp = ismp/2+(ismp%2)*20;
            pit[ismp] =  (pol0[idiv]*pit_cv0[index0[idiv][icv]] + pol1[idiv]*pit_cv1[index1[idiv]][icv]) / 2;
        }
            icv= lengthp[idiv]-1;
```



```
            ismp = idiv + icv* N_DIV_P;
            ismp = ismp/2+(ismp%2)*20;
            pit[ismp] =  (pol0[idiv]*pit_cv0[index0[idiv][icv]] + pol1[idiv]*pit_cv1[index1[idiv]][icv]) / 2;
    }
}
```

″,

in subclause 4.6.9.3.4.2, replace

″

After the crb_tbl[] is determined, the projecting process is done as follows:

```
    for (isf=0; isf<N_SF; isf++){
        ismp=0
        for (ienv=0; ienv<N_CRB; ienv++){
            while (ismp<crb_tbl[ienv]){
                lnenv[i_ch][isf][ismp] = env[i_ch][isf][ienv];
                ismp++;
            }
        }
    }
```

″

with

″

After the crb_tbl[][] is determined, the projecting process is done as follows:

```
for (i_ch=0; i_ch<N_CH ; i_ch++){
    for (isf=0; isf<N_SF; isf++){
        ismp=0
        for (ienv=0; ienv<N_CRB; ienv++){
            while (ismp<crb_tbl[i_ch][ienv]){
                lnenv[i_ch][isf][ismp] = env[i_ch][isf][ienv];
                ismp++;
            }
        }
    }
}
```

″,

and in subclause 4.6.9.3.6, replace

″

```
for (i_ch=0; i_ch<N_CH; i_ch++){
    for (isf=0; isf<N_SF; isf++){
        for (ismp=0; ismp<N_FR; ismp++){
            spec[isf][ismp] =
            (x_flat[ismp+(isf+i_ch*N_SF)*N_FR]*lpenv[i_ch][ismp]*lnenv[i_ch][isf][ismp]
             + pit_seq[i_ch][ismp]) * gain[i_ch][isf];
        }
    }
```

```
}
```
"

with

"

```
for (i_ch=0; i_ch<N_CH; i_ch++){
    for (isf=0; isf<N_SF; isf++){
        for (ismp=0; ismp<N_FR; ismp++){
            spec[isf][ismp] =
             (x_flat[ismp+(isf+i_ch*N_SF)*N_FR]
            *lnenv[i_ch][isf][ismp]* gain[i_ch][isf]+ p_gain[i_ch]*pit_seq[i_ch][ismp])
            *lpenv[i_ch][ismp];
        }
    }
}
```
".

In subclause 4.6.9.3.3.4, replace

"

ISAMPF is an integer sampling frequency in [kHz]

"

with

"

ISAMP is an integer sampling frequency in [kHz] truncated from the standard frequency values listed in the right column of table 5.1 in subpart 4.

".

In subclause 4.6.9.3.3.4, replace

"

```
for (i_ch=0; i_ch<N_CH; i_ch++){
    dtmp = (double)index_pit[i_ch]/ (double) BASF_STEP;
    dtmp = dtmp * (fcmax-fcmin) + fcmin;
    bfreq[i_ch] = (double)pow2(dtmp);
}
```
"

with

"

```
for (i_ch=0; i_ch<N_CH; i_ch++){
    pow_i = (int)(pow( 1.009792f, (float)index_pit[i_sup] ) *4096.+0.5);
    bl_i = (int)((float)block_size_samples/(float)isampf * 0.2*1024+0.5);
    pitch_i = pow_i *bl_i /256.;
    bfreq[I_ch] = pitch_i/16384.;
}
```
",

replace

"

```
npcount = (int)( N_FR_P*bandwidth/(N_FR/bfreq[i_ch]));
```

"

with

"

```
if (bandwidth * upperlimit_i < 16384 ) {
        tmpnp0_i = pitch_i*16384. / (upperlimit_i);
        tmpnp1_i = tmpnp0_i *N_FR_P;
        tmpnp0_i = tmpnp1_I / N_FR;
        npcount = tmpnp0_i/16384.;
} else {
        tmpnp0_i = pitch_i * bandwidth*2;
        tmpnp1_i = tmpnp0_i *N_FR_P;
        tmpnp0_i = tmpnp1_i/N_FR;
        npcount = tmpnp0_i / 32768;
}
```

",

and replace

"

```
for ( ii=0; ii<(ntt_N_FR_P)&& (iscount<ntt_N_FR_P); ii++) {
            i_smp = (int)(bfreq[i_ch]*(ii+1)+0.5);
```

"

with

"

```
for ( ii=0; ii<(ntt_N_FR_P)&& (iscount<ntt_N_FR_P); ii++) {
            tmpnp0_i = pitch_i * (ii+1);
            tmpnp0_i += 8192;
            i_smp = tmpnp0_i / 16384;
```

".

In subclause 4 6.9.3.4.2, replace

"

```
for (i_ch=0; i_ch<N_CH; i_ch++){
    lower_band =LOWER_BOUNDARY[lyr][i_ch];
    upper_band = UPPER_BOUNDARY[lyr][i_ch];
    average_number_of_lines=
        (int)(frame_length*(upper_band-lower_band))/N_CRB;
        for (i=0; i<N_CRB-1; i++){
            crb_tbl[i] = (int)(frame_length*lower_band);
            +(int)((i+1)*(i+1)* average_number_of_lines/ N_CRB/2.0
            +(i+1)* average_number_of_lines/2.0 +0.5);
        }
            crb_tbl[N_CRB-1]=
            (int)(frame_length*lower_band)
            +(int)(frame_length*(upper_band-lower_band));
```

```
}
```

"

with

"

```
for (i_ch=0; i_ch<N_CH; i_ch++){
    lower_band_i =(int)(AC_BTM[lyr][i_ch][fb_shift]*16384.);
    upper_band_i =(int)( AC_TOP[lyr][i_ch][fb_shift]*16384.);
    average_number_of_lines=
        (int)(frame_length*(upper_band_i-lower_band_i))/N_CRB;
        for (i=0; i<N_CRB-1; i++){
                crb_tbl[i_ch][i]  =  (int)((i+1)*(i+1)* average_number_of_lines/ N_CRB/2.0)
            crb_tbl[i_ch][I]  +=  (i+1)* average_number_of_lines/2.0 +8192;
            crb_tbl[i_ch][I]  /= 16384.
            crb_tbl{i_ch][i]  += (int)(frame_length*lower_band_i)/16384.;
        }
            crb_tbl[i_ch][N_CRB-1] = (int)(frame_length*lower_band_i)/16384.
            +(int)(frame_length*(upper_band_i-lower_band_i))/16384.;
}
```

",

and in subclause 4.6.9.3.5.5, replace

"

```
    for(ismp=0; ismp<nfr_lu; ismp++){
        lpenv_tmp[i_ch][ismp+LOWER_BOUNDARY[lyr][i_ch]] =
            lpenv[i_ch][(int)(ismp/(AC_TOP[lyr][i_ch][fb_shift] - AC_BTM[lyr][i_ch][fb_shift]))];
    }
```

"

with

"

```
    upperband_i = (int)( AC_TOP[lyr][i_ch][fb_shift] * 16384.);
    lowerband_i = (int)( AC_BTM[lyr][i_ch][fb_shift] * 16384.);
    ftmp = (16384*16384)/( upperband_i  - lowerband_i );
    for(ismp=0; ismp<nfr_lu; ismp++){
        ftmp = (int)(ismp*ftmp)/16384;
        lpenv_tmp[i_ch][ismp+LOWER_BOUNDARY[lyr][i_ch]] = lpenv[i_ch][ftmp];
    }
```

".

In subclause 4.6.9.3.5.5, replace

"

The LPC spectrum corresponding to ii-th MDCT coefficient, lpenv[][], is defined as follows:

```
for (i_ch=0; i_ch<N_CH; i_ch++){
    for (ii=1; ii<=N_FR-1; ii++){
        for (i=2, P[i_ch]=1.0; i<=N_PR; i+=2)
            P[i_ch] *= (cos(PI*ii/N_FR)-cos(lsp[i]))^2;
```



```
        for (i=1, Q[i_ch]=1.0; i<=N_PR; i+=2)
          Q[i_ch] *= (cos(PI*ii/N_FR)-cos(lsp[i]))^2;
        lpenv[ii] = 1/((1-cos(PI*ii/N_FR))*P[i_ch] + (1+cos(PI*ii/N_FR))*Q[i_ch]);
    }
}
```

"

with

"

The LPC amplitude spectrum envelope corresponding to the ii-th MDCT coefficient, lpenv[][ii], is defined as follows: Note that lpenv[][] seems to represent the power spectrum envelope, but actually represents the amplitude envelope, since the original LPC spectral envelope is derived from the square root of the power spectrum at the encoder.

```
for (i_ch=0; i_ch<N_CH; i_ch++){
    for (ii=1; ii<=N_FR-1; ii++){
        for (i=2, P[i_ch]=1.0; i<=N_PR; i+=2)
           P[i_ch] *= (cos(PI*ii/N_FR)-cos(lsp[i]))^2;
        for (i=1, Q[i_ch]=1.0; i<=N_PR; i+=2)
          Q[i_ch] *= (cos(PI*ii/N_FR)-cos(lsp[i]))^2;
        lpenv[i_ch][ii] = 1/((1-cos(PI*ii/N_FR))*P[i_ch] + (1+cos(PI*ii/N_FR))*Q[i_ch]);
    }
}
```

In the case of long frames (N_FR==1024, or 960), lpenv[][ii] shall be calculated only at frequency points ii that satisfy the following conditions:

```
(
      ((ii= 0 <= ii <N_FR/2) &&  (ii%4==0))
    || ((ii= 0 <= ii <N_FR/2-4) &&  ((ii%2==0) && (lpenv[][ii/4*4]>lpenv[][ii/4*4+8] ) &&
                                    (lpenv[][ii/4*4+4]*1.95 > lpenv[][ii/4*4] + lpenv[][ii/4*4+8] )))
    || ((ii= N_FR/2<= ii <N_FR) && (ii%8==0))
    || ((ii= N_FR/2<= ii <N_FR-8) &&  ((ii%4==0) && (lpenv[][ii/8*8]>lpenv[][ii/8*8+16] ) &&
                                    (lpenv[][ii/8*8+8]*1.95 > lpenv[][ii/8*8] + lpenv[][ii/8*8+16] )))
).
```

For the remaining frequency points, the lpenv[ii] values are calculated by linear interpolation from the values already calculated at the nearest two frequency points. If frequency points are larger than N_FR-8, lpenv[][ii] shall be equal to lpenv[][N_FR-8].

".

In subclause 4.6.12.3, replace

"

Furthermore, if the same scalefactor band and group is coded by perceptual noise substitution in both channels of a channel pair, the correlation of the noise signal can be controlled by means of the ms_used field: While the default noise generation process works independently for each channel (separate generation of random vectors), the same random vector is used for both channels if ms_used[] is set for a particular scalefactor band and group. In this case, no M/S stereo coding is carried out (because M/S stereo coding and noise substitution coding are mutually exclusive).

"

with

"

Furthermore, if the same scalefactor band and group is coded by perceptual noise substitution in both channels of a channel pair, the correlation of the noise signal can be controlled by means of the ms_used field: While the default noise generation process works independently for each channel (separate generation of random vectors), the same random vector is used for both channels if ms_used[] is set for a particular scalefactor band and group. In this case, no M/S stereo coding is carried out (because M/S stereo coding and noise substitution coding are mutually exclusive). If the same scalefactor band and group is coded by perceptual noise substitution in only one channel of a channel pair the setting of ms_used[] is not evaluated.

".

In subclause 4.6.12.6, replace

"

If a particular scalefactor band and group is coded by perceptual noise substitution, its contribution to the spectral components of the output signal is omitted if spectral coefficients are transmitted for this scalefactor band and group in any of the higher (enhancement) layers (that contributes to the output signal) by means of a non-zero codebook number (i.e. a Huffman codebook != ZERO_HCB).

"

with

"

If a particular scalefactor band is coded by perceptual noise substitution in layer N, it only contributes to the output spectrum of the combined layers N and N+1 if all spectral coefficients in this scalefactor band equal zero in the higher layer and no M/S or Intensity Stereo decoding takes place (see subclause 5.2.2.7). In case of a mono-stereo layer combination the condition of all spectral coefficients being equal to zero is expanded to both stereo channels and the FSS tool is used to control the addition.

".

In subclause 5.5.2, under pf[] array in table_event add

"

// when coding sample generator, leave a blank array slot

// for "which" parameter, to maintain alignment for "skip" parameter

".

In class table_event in subclause 5.5.2, replace

"

```
float(32) pf[num_pf] ;
```

"

with

"

```
if (tgen==0x7D) {  // concat
   float(32) size;
   symbol ft[num_pf - 1];
} else {
   float (32) pf[num_pf] ;
}
```

".

In subclause 5.5.2, at the end of second paragraph, add

"

The presence of a zero-length string in a symbol table entry indicates that a name for this symbol is not included in the symbol table.

".

In the first line of points 1 to 7 in subclause 5.7.3.3.6, replace "earlier than" *with* "earlier than or equal to".

In subclause 5.8.5.3.3, replace

"

The order of creation of wavetables is not deterministic;

"

with

"

The order of creation of wavetables is not deterministic, with the exception of the table arguments of a **concat** generator, which are always generated before the **concat** generator that uses them. In this case, the tables used as arguments to the **concat** generator must be appear before the table which uses the **concat** generator, to prevent dependency loops.

".

At the end of subclauses 5.8.6.6.4 and 5.8.6.6.5, add

"

If a block of code executing at the **a-rate** or **k-rate** has **i-rate** statements, these statements should only be executed the first time the block executes, with regards to a particular state.

If a block of code executing at the **a-rate** has **k-rate** statements, these statements should only be executed the first time the block executes in a kcycle.

"

To paragraph six in subclause 5.8.6.6.7, add

"

To clarify, note that the values of the first and second expressions are considered to have units of score beats, not absolute time, and they are consequently scaled according to the actual **tempo** of the orchestra.

".

To the end of subclause 5.8.6.6.7, add

"

A dynamically created instrument has access to the same **MIDIctrl** (subclause 5.8.6.8.9), **MIDItouch** (subclause 5.8.6.8.10), **MIDIbend** (subclause 5.8.6.8.11), **channel** (subclause 5.8.6.8.12) and **preset** (subclause 5.8.6.8.13) standard name state as its parent. However, the dynamically created instrument is not scheduled for termination when the parent is terminated under MIDI control.

".

To the end of subclause 5.8.6.6.10, add

"

In addition, an **outbus** statement may not write to the special bus **input_bus**.

".

In the second to last paragraph of subclause 5.8.6.7.6 (not including NOTES), replace

"

unless the parameter is a standard name

"

with

"

unless the parameter is a standard name which may not be used as an lvalue

".

After paragraph 7 in subclause 5.8.6.7.6, add

"

If a **kopcode** opcode call occurs in a expression that runs at the **a-rate**, the first time this expression is executed in a k-cycle with regards to a particular opcode state, the **kopcode** is called, following the semantics described in this subclause. For all subsequent evaluations of the expression in the same k-cycle, the **kopcode** is not executed; instead, the return value from the first execution is used in the expression evaluation.

If an **iopcode** opcode call occurs in a expression that runs at the **a-rate** or the **k-rate**, the first time this expression is executed with regards to a particular opcode state, the **iopcode** is called, following the semantics described in this subclause. For all subsequent evaluations of the expression, the **iopcode** is not executed; instead, the return value from the first execution is used in the expression evaluation.

If a **specialop** core opcode call occurs in a expression that runs at the **a-rate**, the **k-rate** semantics of the **specialop** opcode call follows the rules for **kopcode** calls described above, while the **a-rate** semantics of the **specialop** opcode call happen at every a-cycle as described in subsection 5.9.2."

".

In subclause 5.8.6.7.7, replace paragraph 6 with

"

The context of the **oparray** call expression is restricted in the same way as described for the **opcode** call expression in subclause 5.8.6.7.6. The rate semantics for **oparray** call execution follows the same rules described for the **opcode** call expression in subclause 5.8.6.7.6.

".

In subclause 5.8.6.7.14, replace Table 5.13 with

"

Table 5.3- Order of operations

Operator	Function
!, -	not, unary
*, /	multiply, divide
+, -	add, subtract
<, >, <=, >=	relational
==, !=	equality
&&	logical and
||	logical or
?:	switch

",

and replace the last line of text in subclause 5.8.6.7.14 with

"

Operations listed on the first and last row associate right-to-left, that is, the rightmost expression is performed first. Operations listed on the remaining rows associate left-to-right, that is, the leftmost expression is performed first.

".

In Annex 5.C.3, replace the paragraph starting with "%start orcfile" *with*

"

```
%start orcfile
%right Q
%left OR
%left AND
%left EQEQ NEQ
%left LT GT LEQ NEQ
%left PLUS MINUS
%left STAR SLASH
%right UNOT UMINUS
%token HIGHEST
```

".

To the end of subclause 5.8.6.8.26, add

"

Instruments may use **params** as an lvalue, that is, to assign new values to it using the **=** statement (subclause 5.8.6.6.2). In this case, when an instrument assigns to **params**, the value for the indicated controller shall be changed on the channel to which the instrument instance associated with that scope is assigned. The value of **params** is changed in all other instrument instances associated with that channel to the new value, and this change shall take effect the next time each of these instrument instances is executed at the k-rate (see subclause 5.7.3.3.6, list item 10).

",

and in subclause 5.8.6.6.2, replace

"

An lvalue shall not be a standard name other than **MIDIctrl** (see subclause 5.8.6.8.9).

"

with

"

An lvalue shall not be a standard name other than **MIDIctrl** and **params** (see subclause 5.8.6.8.9 and 5.8.6.8.26).

".

In subclause 5.8.7.6.1, replace the end of the second paragraph of text with

"

No statement in an opcode shall be faster than the rate of the opcode, as defined in subclause 5.8.7.7. A statement that is slower than the rate of the opcode shall be executed as described in subclause 5.8.7.7.3.

".

In subclause 5.8.7.7.2 before the example, add

"

Rate-polymorphic opcodes shall not contain variable declarations and statements faster than the fastest formal parameter in the opcode declaration. In particular an opcode with all **xsig** formal parameters shall not contain variable declarations other than **xsig** and **ivar** and it shall not contain statements at a defined rate faster than the initialization rate.

".

Add a subclause 5.8.7.7.3

"

5.8.7.7.3 Shared variables and statements slower than the rate of the opcode

This subclause describes the rules for updating shared variables and for executing statements inside an opcode that are slower than the rate of the opcode.

In a **kopcode**, statements at the initialization rate are executed at the first call to the opcode with regard to a particular opcode state. At this call the **imports**, **exports** and **imports exports** ivars and tables are updated, as described in 5.8.6.5.3 and 5.8.6.5.4, and any wavetable generations are executed, as described in 5.8.6.5.2.

In an **aopcode**, statements at the initialization rate are executed at the first call to the opcode with regard to a particular opcode state. At this call the **imports**, **exports** and **imports exports** ivars and tables are updated, as described in 5.8.6.5.3 and 5.8.6.5.4, and any wavetable generations are executed, as described in 5.8.6.5.2.

Statements at the control rate are executed at the first call of every k-cycle to the opcode with regard to a particular opcode state. At this call the **imports**, **exports** and **imports exports** ksigs and tables are updated, as described in 5.8.6.5.3 and 5.8.6.5.4.

In an **opcode**, once the rate is determined as specified in subclause 5.8.7.7.2 the same rules apply for the cases of call at **k-rate** or **a-rate** respectively.

".

To the last paragraph of subclause 5.8.6.7.6, add

"

If the rate of a formal parameter is slower than the rate of the opcode, the following rules apply:

in an opcode call at the **k-rate**, actual variables associated to an **ivar** formal parameters are updated only the first time this opcode is executed with regards to a particular opcode state;

in an opcode call at the **a-rate**, actual variables associated to an **ivar** formal parameters are updated only the first time this opcode is executed with regards to a particular opcode state; actual variables associated to a **ksig** formal parameters are updated only the first time this opcode is executed in a k-cycle with regards to a particular opcode state.

".

