INTERNATIONAL STANDARD ISO/IEC 13818-7:1997
TECHNICAL CORRIGENDUM 1

Published 1998-12-01

Information technology — Generic coding of moving pictures and associated audio information —

Part 7:
Advanced Audio Coding (AAC)

TECHNICAL CORRIGENDUM 1

Technologies de l'information — Codage générique des images animées et du son associé —

Partie 7: Codage du son avancé (AAC)

RECTIFICATIF TECHNIQUE 1

Technical Corrigendum 1 to International Standard ISO/IEC 13818-7:1997 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information.*

---

1) Add the following paragraph at the end of clause 5:
"
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.
"

2) Replace Table 6.13 in subclause 6.3 with the following:

“

```
Syntax                                                              No. of bits   Mnemonic
section_data()
{
    if( window_sequence == EIGHT_SHORT_SEQUENCE )
        sect_esc_val = (1<<3) - 1
    else
        sect_esc_val = (1<<5) - 1

    for( g=0; g < num_window_groups; g++ ) {
        k=0
        i=0
        while (k<max_sfb) {
            sect_cb[g][i]                                           4             uimsbf
            sect_len=0
            while (sect_len_incr == sect_esc_val)                   {3;5}         uimsbf
                sect_len += sect_esc_val
            sect_len += sect_len_incr
            sect_start[g][i] = k
            sect_end[g][i] = k+sect_len
            for (sfb=k; sfb<k+sect_len; sfb++)
                sfb_cb[g][sfb] = sect_cb[g][i];
            k += sect_len
            i++
        }
        num_sec[g] = i
    }

}
```

”

3) In Table 6.15 in subclause 6.3, replace:
No. of bits “4/6“ with “{4;6}”
and
No. of bits “3/5“ with “{3;5}”.

4) Replace Table 6.16 in subclause 6.3 with the following:

“

```
Syntax                                                              No. of bits   Mnemonic
spectral_data()
{
    for( g=0; g<num_window_groups; g++ ) {
        for (i=0; i<num_sec[g]; i++) {
            if (sect_cb[g][i] != ZERO_HCB &&
                sect_cb[g][i] <= ESC_HCB) {
                for (k=sect_sfb_offset[g][sect_start[g][i]];
                        k< sect_sfb_offset[g][sect_end[g][i]]; ) {
                    if (sect_cb[g][i]<FIRST_PAIR_HCB) {
                        hcod[sect_cb[g][i]][w][x][y][z]                   1..16   bslbf
                        if( unsigned_cb[sect_cb[g][i]] )
                            quad_sign_bits                                 0..4   bslbf
                        k += QUAD_LEN
                    }
                    else {
                        hcod[sect_cb[g][i]][y][z]                         1..15   bslbf
                        if( unsigned_cb[sect_cb[g][i]] )
                            pair_sign_bits                                 0..2   bslbf
                        k += PAIR_LEN
                    if (sect_cb[g][i]==ESC_HCB) {
                        if (y==ESC_FLAG)
                            hcod_esc_y                                5..21   bslbf
                        if (z==ESC_FLAG)
                            hcod_esc_z                                5..21   bslbf
                    }
                }
            }
        }
    }
}
```
"

5) *Replace Table 6.18 in subclause 6.3 with the following:*
"

```
Syntax                                                          No. of bits   Mnemonic
coupling_channel_element()
{
    element_instance_tag                                        4             uimsbf
    ind_sw_cce_flag                                             1             uimsbf
    num_coupled_elements                                        3             uimsbf
    num_gain_element_lists = 0
    for (c=0; c<num_coupled_elements+1; c++) {
        num_gain_element_lists++
        cc_target_is_cpe[c]                                     1             uimsbf
        cc_target_tag_select[c]                                 4             uimsbf
        if ( cc_target_is_cpe[c] ) {
            cc_l[c]                                             1             uimsbf
            cc_r[c]                                             1             uimsbf
            if (cc_l[c] && cc_r[c] )
                num_gain_element_lists++
        }
    }
    cc_domain                                                   1             uimsbf
    gain_element_sign                                           1             uimsbf
    gain_element_scale                                          2             uimsbf

    individual_channel_stream(0)

    for ( c=1; c<num_gain_element_lists; c++ ) {
        if ( ind_sw_cce_flag ) {
            cge = 1
        } else {
            common_gain_element_present[c]                      1             uimsbf
            cge = common_gain_element_present[c]
        }
        if ( cge )
            hcod_sf[common_gain_element[c]]                     1..19         bslbf
        else {
            for (g=0; g<num_window_groups; g++) {
                for (sfb=0; sfb<max_sfb; sfb++) {
                    if ( sfb_cb[g][sfb] != ZERO_HCB )
                        hcod_sf[dpcm_gain_element[c][g][sfb]]   1..19         bslbf
                }
            }
        }
    }
}
```
"

6) Replace Table 6.22 in subclause 6.3 with the following:

"

Syntax	No. of bits	Mnemonic
fill_element()		
{		
cnt = **count**	**4**	**uimsbf**
if (cnt == 15)		
cnt += **esc_count** - 1;	**8**	**uimsbf**
while (cnt > 0) {		
cnt -= extension_payload(cnt)		
}		
}		

"

and add the following tables, Table 6.24, Table 6.25 and Table 6.26, at the end of subclause 6.3:

"

Table 6.24 — Syntax of extension_payload()

extension_payload(cnt)		
{		
extension_type	**4**	**uimsbf**
switch(extension_type) {		
case EXT_DYNAMIC_RANGE:		
n = dynamic_range_info();		
return n;		
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 default:		
for (i=0; i<8*(cnt-1)+4; i++)		
other_bits[i]	**1**	**uimsbf**
return cnt		
}		
}		

Table 6.25 — Syntax of dynamic_range_info()

Syntax	No. of bits	Mnemonic
dynamic_range_info()		
{		
n = 1		
drc_num_bands = 1		
pce_tag_present	**1**	**uimsbf**
if (pce_tag_present == 1) {		
pce_ instance_tag	**4**	**uimsbf**
drc_tag_reserved_bits	**4**	
n++		
}		
excluded_chns_present	**1**	**uimsbf**
if (excluded_chns_present == 1) {		
n += excluded_channels()		
}		
drc_bands_present	**1**	**uimsbf**
if (drc_bands_present == 1) {		
drc_band_incr	**4**	**uimsbf**
drc_bands_reserved_bits	**4**	**uimsbf**
n++		

drc_num_bands = drc_num_bands + drc_band_incr		
for (i=0; i<drc_num_bands; i++) {		
drc_band_top[i]	**8**	**uimsbf**
n++		
}		
}		
prog_ref_level_present	**1**	**uimsbf**
if (prog_ref_level_present == 1) {		
prog_ref_level	**7**	**uimsbf**
prog_ref_level_reserved_bits	**1**	**uimsbf**
n++		
}		
for (i=0; i<drc_num_bands; i++) {		
dyn_rng_sgn[i]	**1**	**uimsbf**
dyn_rng_ctl[i]	**7**	**uimsbf**
n++		
}		
return n		
}		

Table 6.26 — Syntax of excluded_channels()

Syntax	No. Of bits	Mnemonic
excluded_channels( )		
{		
n = 0		
num_excl_chan = 7		
for (i=0; i<7; i++)		
exclude_mask[i]	**1**	**uimsbf**
n++		
while (**additional_excluded_chns[n-1]** == 1) {	**1**	**uimsbf**
for (i= num_excl_chan; i< num_excl_chan+7; i++)		
exclude_mask[i]	**1**	**uimsbf**
n++		
num_excl_chan += 7		
}		
return n		
}		

”

7) In subclause 7.1, replace “ATDS0” *with* “ADTS”.

8) Replace definition of **num_coupled_channels** *in subclause 7.3.2 with the following:*
“

num_coupled_elements	number of coupled target elements

”

9) Replace definition of **home** *in subclause 8.1.1 with the following:*
“
see ISO/IEC 11172-3, subclause 2.4.2.3 (Table 6.2) definition for **original_copy**
”

10) Replace definition of **original_copy** *in subclause 8.1.1 with the following:*
“
see ISO/IEC 11172-3, subclause 2.4.2.3 (table 6.2) definition for **copyright**
”

11) Replace the last sentence in the second paragraph of subclause 8.1.2 with the following:
"

However, one non-normative transport stream, called Audio_Data_Transport_Stream (ADTS), is described. It may be used for applications in which the decoder can parse this stream.
"

12) Replace the first two paragraphs of subclause 8.2.3 with the following:
"

Assuming that the start of a raw_data_block is known, it can be decoded without any additional «transport-level» information and produces 1024 audio samples per output channel. The sampling rate of the audio signal, as specified by the **sampling_frequency_index,** may be specified in a program_config_element or it may be implied in the specific application domain. In the latter case, the **sampling_frequency_index** must be deduced in order for the bitstream to be parsed. Since a given **sampling_frequency_index** is associated with only one sampling frequency, 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_index**. It is used as follows: identify the frequency in the table that is the highest frequency that is less than or equal to the implied frequency, and use the index that is in that same row.

Frequency	sampling_frequency_index
92017	0x0
75132	0x1
55426	0x2
46009	0x3
37566	0x4
27713	0x5
23004	0x6
18783	0x7
13856	0x8
11502	0x9
9391	0xa
0	0xb

Assuming that the start of the first raw_data_block in a raw_data_stream is known, the sequence can be decoded without any additional "transport-level" information and produces 1024 audio samples per raw_data_block per output channel.
"

13) Replace the second item of the second level bulleted list in subclause 8.3.5 with the following:
"

If there is only one group with length eight (num_window_group = 1, window_group_length[0]=8), the results is that spectral data of all eight SHORT_WINDOWs is interleaved by scalefactor window bands.
"

14) Replace definition of **num_valid_cce_elements** *in subclause 8.5 with the following:*
"

num_valid_cc_elements The number of CCE's that can add to the audio data for this program (Table 6.21)
"

15) Replace definition of **valid_cce_element_tag_select** *in subclause 8. 5 with the following:*
"

valid_cc_element_tag_select instance_tag of the CCE addressed (Table 6.21)
"

16) Replace subclause 8.7 with the following:
"

8.7 Fill element (FIL) including Dynamic Range Control (DRC)

Bitstream elements:

count	Initial value for length of fill data (Table 6.22)
esc_count	Incremental value of length of fill data (Table 6.22)
extension_type	Four bit field indicating the type of fill element content (Table 6.22)
fill_nibble	Four bit field for fill (Table 6.24)
fill_byte	Byte to be discarded by the decoder (Table 6.24)
other_bits	Bits to be discarded by the decoder (Table 6.24)
pce_tag_present	One bit indicating that program element tag is present (table 6.25).
pce_instance_tag	Tag field that indicates with which program the dynamic range information is associated (table 6.25)
drc_tag_reserved_bits	Reserved (table 6.25)
excluded_chns_present	One bit indicating that excluded channels are present (table 6.25)
drc_bands_present	One bit indicating that DRC multi-band information is present (table 6.25)
drc_band_incr	Number of DRC bands greater than 1 having DRC information (table 6.25)
drc_bands_reserved_bits	Reserved (table 6.25)
drc_band_top[i]	Indicates top of i-th DRC band in units of 4 spectral lines (table 6.25). If drc_band_top[i]=k, then the index (w.r.t zero) of the highest spectral coefficient that is in the i-th DRC band is = k*4+3. In case of an EIGHT_SHORT_SEQUENCE window_sequence the index is interpreted as pointing into the concatenated array of 8*128 (de-interleaved) frequency points corresponding to the 8 short transforms.
prog_ref_level_present	One bit indicating that reference level is present (table 6.25).
prog_ref_level	Reference level. A measure of long-term program audio level for all channels combined (table 6.25).
prog_ref_level_reserved_bits	Reserved (table 6.25)
dyn_rng_sgn[i]	Dynamic range control sign information. One bit indicating the sign of dyn_rng_ctl (0 if positive, 1 if negative, table 6.25)
dyn_rng_ctl[i]	Dynamic range control magnitude information (table 6.25)
exclude_mask[i]	Boolean array indicating the audio channels of a program that are excluded from DRC processing using this DRC information.
additional_excluded_chns[i]	One bit indicating that additional excluded channels are present (table 6.26)

Fill elements have to be added to the bitstream if the total bits for all audio data together with all additional data is lower than the minimum allowed number of bits in this frame necessary to reach the target bitrate. Dynamic Range Control (DRC) bits must be added to the fill element whenever the encoder wishes to include DRC information. Under normal conditions fill bits are avoided and free bits are used to fill up the bit reservoir. Fill bits are written only if the bit reservoir is full. Any number of fill elements are allowed.

Decoding process:

The syntactic element **count** gives the initial value of the length of the fill data. In the same way as for the data element this value is incremented with the value of **esc_count** if **count** equals 15. The resulting number gives the number of bytes to be read.

DRC Decoding process:

Fill elements containing an extension_payload with a extension_type of EXT_DYNAMIC_RANGE (see below) are reserved for dynamic range information. In this case the fill_element **count** field must be set equal to the total length, in bytes, of all dynamic range information plus the extension_type field.

prog_ref_level_present indicates that **prog_ref_level** is being transmitted. This permits **prog_ref_level** to be sent as infrequently as desired (e.g. once), although periodic transmission would permit break-in.

prog_ref_level is quantized in 0.25 dB steps using 7 bits, and therefore has a range of approximately 32 dB. It indicates program level relative to full scale (i.e. dB below full scale), and is reconstructed as:

$$level = 32767 \cdot 2^{-prog\_ref\_level/24}$$

where "full scale level" is 32767 (prog_ref_level equal to 0).

pce_tag_present indicates that **pce_instance_tag** is being transmitted. This permits **pce_instance_tag** to be sent as infrequently as desired (e.g. once), although periodic transmission would permit break-in.

pce_instance_tag indicates with which program the dynamic range information is associated. If this is not present then the default program is indicated. Since each AAC bitstream typically has just one program, this would be the most common mode. Each program in a multi-program bitstream would send its dynamic range information in a distinct extension_payload() of the fill_element(). In the multiple program case, the **pce_instance_tag** would always have to be signaled.

The **drc_tag_reserved_bits** fill out the optional fields to an integral number of bytes in length.

The **excluded_chns_present** bit indicates that channels that are to be *excluded* from dynamic range processing will be signaled immediately following this bit. The excluded channel mask information must be transmitted in each frame where channels are excluded. The following ordering principles are used to assign the exclude_mask to channel outputs:

- If a PCE is present (explicit speaker mapping), the **exclude_mask** bits correspond to the audio channels in the SCE, CPE, CCE and LFE syntax elements in the order of their appearance in the PCE. In the case of a CPE, the first transmitted mask bit corresponds to the first channel in the CPE, the second transmitted mask bit to the second channel. In the case of a CCE, a mask bit is transmitted only if the coupling channel is specified to be an independently switched coupling channel.
- For the case of an implicit speaker mapping (no PCE present), the **exclude_mask** bits correspond to the audio channels in the SCE, CPE and LFE syntax elements in the order of their appearance in the bitstream, followed by the audio channels in the CCE syntax elements in the order of their appearance in the bitstream. In the case of a CPE, the first transmitted mask bit corresponds to the first channel in the CPE, the second transmitted mask bit to the second channel. In the case of CCE, a mask bit is transmitted only if the coupling channel is specified to be an independently switched coupling channel.

drc_band_incr is the number of bands greater than one if there is multi-band DRC information.

dyn_rng_ctl is quantized in 0.25 dB steps using a 7-bit unsigned integer, and therefore, in association with **dyn_rng_sgn,** has a range of +/-31.75 dB. It is interpreted as a gain value that shall be applied to the decoded audio output samples of the current frame.

The range supported by the dynamic range information is summarized in the following table:

Field	bits	steps	stepsize, dB	range, dB
prog_ref_level	7	128	0.25	31.75
dyn_rng_sgn and **dyn_rng_ctl**	1 and 7	+/- 127	0.25	+/- 31.75

The following symbolic abbreviations for values of the extension_type field are defined currently:

Symbol	**Value of extension_type**	**Purpose**
EXT_FILL	'0000'	Bitstream filler
EXT_FILL_DATA	'0001'	Bitstream data as filler
EXT_DYNAMIC_RANGE	'1011'	Dynamic range control
-	all other values	reserved

The 'reserved' values can be used for further extension of the syntax in a compatible way.

Note that fill_nibble is normatively defined to be '0000' and fill_byte is normatively defined to be '10100101' (to ensure that self-clocked data streams, such as radio modems, can perform reliable clock recovery).

The dynamic range control process is applied to the spectral data `spec[i]` of one frame immediately before the synthesis filterbank. In case of an EIGHT_SHORT_SEQUENCE window_sequence the index i is interpreted as pointing into the concatenated array of 8*128 (de-interleaved) frequency points corresponding to the 8 short transforms.

This following pseudo code is for illustrative purposes only, showing one method for applying one set of dynamic control information to a frame of a target audio channel. The constants `ctrl1` and `ctrl2` are compression constants (typically between 0 and 1, zero meaning no compression) that may optionally be used to scale the dynamic range compression characteristics for levels greater than or less than the program reference level, respectively. The constant `target_level` describes the output level desired by the user, expressed in the same scaling as `prog_ref_level`.

```
bottom = 0;
drc_num_bands = 1;
if (drc_bands_present)
   drc_num_bands += drc_band_incr;
if (drc_num_bands == 1)
   drc_band_top[0] = 1024/4 - 1;
for (bd=0; bd < drc_num_bands; bd++)  {
   top = 4 * (drc_band_top[bd] + 1);

   /* Decode DRC gain factor */
   if (dyn_rng_sgn[bd])
      factor = 2^(-ctrl1*dyn_rng_ctl[bd]/24);   /* compress */
   else
      factor = 2^(ctrl2*dyn_rng_ctl[bd]/24);    /* boost */

   /* If program reference normalization is done in the digital domain, modify
    * factor to perform normalization.
    * prog_ref_level can alternatively be passed to the system for modification
    * of the level in the analog domain. Analog level modification avoids problems
    * with reduced DAC SNR (if signal is attenuated) or clipping (if signal is boosted)
    */
   factor *= 0.5^((target_level-prog_ref_level)/24);

   /* Apply gain factor */
   for (i=bottom; i<top; i++)
      spec[i] *= factor;
   bottom = top;
}
```

Note the relation between dynamic range control and coupling channels:

- Dependently switched coupling channels are always coupled onto their target channels as spectral coefficients prior to the DRC processing and synthesis filtering of these channels. Therefore a dependently switched coupling channel's signal that couples onto to a specific target channel will undergo the DRC processing of that target channel.
- Since independently switched coupling channels couple to their target channels in the time domain, each independently switched coupling channel will undergo DRC processing and subsequent synthesis filtering separate from its target channels. This permits the independently switched coupling channel to have distinct DRC processing if desired.

Persistence of DRC information:

At the beginning of a stream, all DRC information for all channels is assumed to be set to its default value: program reference level equal to the decoder's target reference level, one DRC band, with no DRC gain modification for that band. Unless this data is specifically overwritten, this remains in effect.

There are two cases for the persistence of DRC information that has been transmitted:

- The program reference level is per audio program, and persists until a new value is transmitted, at which point the new data overwrites the old and takes effect that frame. (It may be appropriate to send this value periodically to allow bitstream break-in.)
- Other DRC information persists on a per-channel basis. Note that if a channel is excluded via the appropriate **exclude_mask[]** bit, then effectively no information is transmitted for that channel in that call to dynamic_range_info(). The excluded channel mask information must be transmitted in each frame where channels are excluded.

The rules for retaining per-channel DRC information are as follows:

- If there is no DRC information in a given frame for a given channel, use the information that was used in the previous frame. (This means that one adjustment can hold for a long time, although it may be appropriate to transmit the DRC information periodically to permit break-in.)

- If any DRC information for this channel appears in the current frame, the following sequence occurs: first, overwrite all per-channel DRC information for that channel with the default values (one DRC band, with no DRC gain modification for that band), then overwrite any per-channel DRC information with the transmitted values.

"

17) In the fifth paragraph of subclause 9.3, replace "less than 24 bits" *with* "less than 22 bits".

18) In subclause 10.3, replace the third line in the inverse quantization with the following:

"

```
        width = (swb_offset [sfb+1] - swb_offset [sfb]);
```

"

19) In subclause 11.3.2, replace "`/* see clause 4 */`" *with* "`/* see clause 9 */`".

20) In the first paragraph of subclause 11.3.2, replace "(but is initialized to zero to have an valid in the array)" *with* "(but is initialized to zero to have a valid entry in the array)".

21) Add the following sentence at the end of subclause 11.3.2:

"

Note that scalefactors, sf[g][sfb], must be within the range of zero to 256, both inclusive.

"

22) In subclause 12.1.3, replace the pseudo code used for computing the inverse M/S matrix

"

```
        tmp = l_spec[g][b][sfb][i] +
                r_spec[g][b][sfb][i];
        l_spec[g][b][sfb][i] =    l_spec[g][b][sfb][i] -
                r_spec[g][b][sfb][i];
        r_spec[g][b][sfb][i] = tmp;
```

"

with

"

```
        tmp = l_spec[g][b][sfb][i] -
                r_spec[g][b][sfb][i];
        l_spec[g][b][sfb][i] =    l_spec[g][b][sfb][i] +
                r_spec[g][b][sfb][i];
        r_spec[g][b][sfb][i] = tmp;
```

"

23) In the second paragraph of subclause 13.3.2.1, replace:

"

$$x_{est,m}(n) = b \cdot k_m(n) \cdot a \cdot r_{q,m-1}(n-1),$$

where

$$r_{q,m}(n) = r_{q,m-1}(n-1) - b \cdot k_m(n) \cdot e_{q,m-1}(n)$$

"

with

"

$$x_{est,m}(n) = b \cdot k_m(n) \cdot r_{q,m-1}(n-1),$$

where

$$r_{q,0}(n) = ax_{rec}(n),$$

$$r_{q,1}(n) = a(r_{q,0}(n-1) - b \cdot k_1(n) \cdot e_{q,0}(n))$$

"

24) In subclause 13.3.2.3, replace:
"
```
static void
flt_round_inf(float *pf)
{
    int flg;
    ulong tmp, tmp1;
    float *pt = (float *)&tmp;
    *pt = *pf;                       /* write float to memory */
    tmp1 = tmp;                            /* save in tmp1 */
    flg = tmp & (ulong)0x00008000;   /* rounding position */
    tmp &= (ulong)0xffff0000;              /* truncated float */
    *pf = *pt;
    /* round 1/2 lsb toward infinity */
    if (flg) {
      tmp = tmp1 & (ulong)0xff810000;/* 1.0 * 2^e + 1 lsb */
      *pf += *pt;                    /* add 1.0 * 2^e+ 1 lsb */
      tmp &= (ulong)0xff800000;            /* 1.0 * 2^e */
      *pf -= *pt;                    /* subtract 1.0 * 2^e */
   }
}
```
"

with
"
```
static void
flt_round_inf(float *pf)
{
     int flg;
     ulong tmp;
     float *pt = (float *)&tmp; /* note: this presumes 32 bit ulong */
     *pt = *pf;
     flg = tmp & (ulong)0x00008000;
     tmp &= (ulong)0xffff0000;
     *pf = *pt;
     /* round 1/2 lsb toward infinity */
     if (flg) {
         tmp &= (ulong)0xff800000;       /* extract exponent and sign */
         tmp |= (ulong)0x00010000;       /* insert 1 lsb */
         *pf += *pt;                     /* add 1 lsb and elided one */
         tmp &= (ulong)0xff800000;       /* extract exponent and sign */
         *pf -= *pt;                     /* subtract elided one */
     }
}
```
"

25) In subclause 13.3.2.4, replace:
"
```
static void
flt_round_even(float *pf)
{
   float f1, f2;

   f1 = 1.0;
   f2 = f1 + (*pf / (1<<15));
   f2 = f2 - f1;
   f2 = f2 * (1<<15);
   *pf = f2;
}
```
"

with
"
```
static void
flt_round_even(float *pf)
{
   int exp;
   double mnt;
   float offset;
   mnt = frexp((double)*pf, &exp);
   offset = (float)ldexp(1.0, exp+15);
   *pf += offset;   /* WARNING:  This shifts out LSB's. Do not remove this pair of
operations! */
   *pf -= offset;
}
```
"

26) In subclause 13.3.2.4, replace:

"

```
   /* exponent table */
   for (i=0; i<256; i++) {
      tmp = tmp1 + i<<23;       /* float 1.0 * 2^exp */
      ftmp = 1.0 / *pf;
      exp_table[i] = ftmp;
   }
```
"

with

"

```
   /* exponent table */
   for (i=0; i<256; i++) {
      tmp = i<<23;              /* float 1.0 * 2^exp */
      if (*pf > 1.0) {
         ftmp = 1.0 / *pf;
      }
      else {
         ftmp = 0;
      }
      exp_table[i] = ftmp;
   }
```
"

27) In the sixth paragraph of subclause 13.3.3, replace the first sentence with:

"

An encoder is required to signal the reset of a group at least once every 8 frames when prediction has been switched on as indicated by the **predictor_data_present** flag.

"

28) In subclause 14.3, replace:

"

```
/* Conversion to LPC coefficients */
a[0] = 1;
for (m=1; m<=order; m++)  {
      b[0] = 1;
      b[m+1] = 0;
      for (i=1; i<=m; i++) {
            b[i] = a[i] + tmp2[m] * a[m-i];
      }
      for (i=0; i<=m; i++)  {
            a[i] = b[i];
      }
}
```
"

with

"

```
/* Conversion to LPC coefficients */
a[0] = 1;
for (m=1; m<=order; m++) {
      for (i=1; i<m; i++) {          /* loop only while i<m */
            b[i] = a[i] + tmp2[m-1] * a[m-i];
      }
```

```
    for (i=1; i<m; i++) {          /* loop only while i<m */
        a[i] = b[i];
    }
    a[m] = tmp2[m-1];              /* changed */

}
```
"

29) After the pseudo code for TNS decoding in subclause 14.3, add the following:
"
Please note that this pseudo code uses a "C"-style interpretation of arrays and vectors, i.e. if coef[w][filt][i] describes the coefficients for all windows and filters, coef[w][filt] is a pointer to the coefficients of one particular window and filter. Also, the identifier coef is used as a formal parameter in function tns_decode_coef().
"

30) In subclause 15.3.2, replace the formula:
"

$$W_{KBD\_RIGHT,N}(n) = \sqrt{\frac{\sum_{p=0}^{N-n}[W'(n,\alpha)]}{\sum_{p=0}^{N/2}[W'(p,\alpha)]}} \quad \text{for} \quad \frac{N}{2} \le n < N$$

"
with
"

$$W_{KBD\_RIGHT,N}(n) = \sqrt{\frac{\sum_{p=0}^{N-n-1}[W'(n,\alpha)]}{\sum_{p=0}^{N/2}[W'(p,\alpha)]}} \quad \text{for} \quad \frac{N}{2} \le n < N$$

"

31) In subclause 16.2, replace definition of **max_band** *with the following:*
"
2-bit field indicating the number of IPQF bands in which their signal gain have been controlled.
The meanings of this value are shown below (see 6.3, Table 6.23).

0: no bands have activated gain control.
1: signal gain on 2nd IPQF band has been controlled.
2: signal gain on 2nd and 3rd IPQF bands have been controlled.
3: signal gain on 2nd, 3rd and 4th IPQF bands have been controlled.
"

32) Add clause 0 in Annex B:
"

B.0 Information on Unused Codebooks

As specified by the normative part of this standard, the AAC decoder does not make use of codebooks #12 and #13. However, if desired, a decoder may use these codebooks to extend its functionality in a way that is consistent with other MPEG standards like ISO/IEC 14496-3 which use these particular codebooks to indicate coding by extended coding methods.
As an example, the syntax in Table 6.14 in subclause 6.3 would change to

```
Syntax                                                     No. Of bits   Mnemonic
scale_factor_data()
{
    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         bslbf
                Else if ( sfb_cb[g][sfb] == 13 )
                    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         bslbf
                Else
                    hcod_sf[dpcm_sf[g][sfb]]               1..19         bslbf
            }
        }
    }
}
```
”

33) Replace Table B.2.1.4.a with the following:
“

Table B.2.1.4.a — Psychoacoustic parameters for 16 KHz long FFT

index	w_low	w_high	width	bval	qsthr
0	0	4	5	0,20	43,30
1	5	9	5	0,59	43,10
2	10	14	5	0,99	38,30
3	15	19	5	1,38	38,10
4	20	24	5	1,77	38,00
5	25	29	5	2,16	35,10
6	30	34	5	2,54	35,30
7	35	39	5	2,92	30,00
8	40	44	5	3,29	30,00
9	45	49	5	3,66	28,30
10	50	54	5	4,03	28,30
11	55	59	5	4,39	28,30
12	60	64	5	4,74	28,30
13	65	69	5	5,09	28,30
14	70	74	5	5,43	28,30
15	75	80	6	5,79	28,30
16	81	86	6	6,18	28,30
17	87	92	6	6,56	28,00
18	93	98	6	6,92	29,27
19	99	104	6	7,28	29,27
20	105	110	6	7,63	29,27
21	111	116	6	7,96	29,27
22	117	123	7	8,31	29,27
23	124	130	7	8,68	29,06
24	131	137	7	9,03	30,06
25	138	144	7	9,37	30,06
26	145	152	8	9,71	30,06
27	153	160	8	10,07	30,73
28	161	168	8	10,41	30,73

29	169	177	9	10,75	30,73
30	178	186	9	11,10	31,31
31	187	196	10	11,45	31,31
32	197	206	10	11,80	31,82
33	207	217	11	12,14	31,82
34	218	228	11	12,48	32,28
35	229	240	12	12,82	32,28
36	241	253	13	13,16	32,69
37	254	267	14	13,51	32,69
38	268	282	15	13,86	33,07
39	283	298	16	14,21	33,46
40	299	315	17	14,56	33,82
41	316	333	18	14,90	34,12
42	334	352	19	15,24	34,42
43	353	373	21	15,58	34,68
44	374	395	22	15,91	35,15
45	396	419	24	16,25	35,32
46	420	445	26	16,58	35,73
47	446	473	28	16,92	35,91
48	474	503	30	17,25	36,42
49	504	536	33	17,59	36,75
50	537	571	35	17,93	37,11
51	572	609	38	18,26	37,34
52	610	650	41	18,60	37,63
53	651	694	44	18,94	38,12
54	695	741	47	19,27	38,17
55	742	791	50	19,60	41,52
56	792	845	54	19,94	41,84
57	846	903	58	20,27	42,13
58	904	965	62	20,61	44,41
59	966	1023	58	20,92	44,87

"

34) Add the following at the end of subclause B 2.5:
"

Optionally, the use of the coef_compress field allows saving 1 bit per transmitted reflection coefficient if none of the reflection coefficients use more than half of their full range. Specifically, if the two most significant bits of each quantized reflection coefficient are either '00' or '11', coeff_compress may be set to a value of one and the size of the transmitted quantized reflection coefficients decreased by one.
"

35) Replace subclauses B.2.7.1 through B.2.7.5 with the following:
"

B.2.7.1 Introduction

The description of the AAC quantization module is subdivided into three levels. The top level is called "loops frame program". The loops frame program calls a subroutine named "outer iteration loop" which calls the subroutine "inner iteration loop". For each level a corresponding flow diagram is shown.

The loops module quantizes an input vector of spectral data in an iterative process according to several demands. The inner loop quantizes the input vector and increases the quantizer step size until the output vector can be coded with the available number of bits. After completion of the inner loop an outer loop checks the distortion of each scalefactor band and, if the allowed distortion is exceeded, amplifies the scalefactor band and calls the inner loop again.

AAC loops module input:

1. vector of the magnitudes of the spectral values mdct_line(0..1023).
2. xmin(sb) (see B 2.1.4. „Steps in threshold calculation“, Step 12)

3. mean_bits (average number of bits available for encoding the bitstream).
4. more_bits, the number of bits in addition to the average number of bits, calculated by the psychoacoustic module out of the perceptual entropy (PE).
5. the number and width of the scalefactor bands (see table 3.5 normative part)
6. for short block grouping the spectral values have to be interleaved so that spectral lines that belong to the same scalefactor band but to different block types which shall be quantized with the same scalefactors are put together in one (bigger) scalefactor band (for a full description of grouping see clause 3.3.4 normative part)

AAC loops module output:
1. vector of quantized values x_quant(0..1023).
2. a scalefactor for each scalefactor band (sb)
3. common_scalefac (quantizer step size information for all scalefactor bands)
4. number of unused bits available for later use.

B.2.7.2 Preparatory steps

B.2.7.2.1 Reset of all iteration variables

1. The start value of common_scalefac for the quantizer is calculated so that all quantized MDCT values can be encoded in the bitstream :.

$start\_common\_scalefac = ceiling(16/3*(\log_2((max\_mdct\_line \wedge (3/4))/MAX\_QUANT))$

max_mdct_line is the largest absolute MDCT coefficient value, and ceiling() is the function which rounds to the nearest integer in the direction of positive infinity. MAX_QUANT is the maximum quantized value which can be encoded in the bitstream, defined to be 8191. During the iteration process, the common_scalefac must not become less than start_common_scalefac.

2. All scalefactors are set to zero.

B.2.7.3 Bit reservoir control

Bits are saved to the reservoir when fewer than the average bits are used to code one frame.

*average_bits = bit_rate * 1024 / sampling_rate.*

The number of bits which can be saved in the bit reservoir at maximum is called 'maximum_bitreservoir_size' which is calculated using the procedure outlined in normative clause 3.2.2. If the reservoir is full, unused bits have to be encoded in the bitstream as fillbits.

The maximum amount of bits available for a frame is the sum of mean_bits and bits saved in the bit reservoir.

The number of bits that should be used for encoding a frame depends on the more_bits value which is calculated by the psychoacoustic model and the maximum available bits. The simplest way to control bit reservoir is :

```
if more_bits > 0 :
        available_bits = average_bits + min ( more_bits, bitres_bits)
if more_bits < 0 :
        available_bits = average_bits + max ( more_bits, bitres_bits - maximum_bitreservoir_size)
```

B.2.7.4 Quantization of MDCT coefficients

The formula for the quantization in the encoder is the inverse of the decoder dequantization formula (see also the decoder description) :

*x_quant = int ((abs(mdct_line) * (2^(¼ * (sf_decoder - SF_OFFSET))))^(3/4) + MAGIC_NUMBER)*

MAGIC_NUMBER is defined to 0.4054, SF_OFFSET is defined as 100 and mdct_line is one of spectral values, which is calculated from the MDCT. These values are also called 'coefficients'

For use in the iteration loops, the scalefactor ‘sf_decoder’ is split in two variables:

sf_decoder = common_scalefac - scalefactor + SF_OFFSET

It follows from this, that the formula used in the distortion control loop is:

*x_quant = int ([abs(mdct_line) * (2^(¼ * (scalefactor - common_scalefac)))]^(3/4) + MAGIC_NUMBER)*

The signs of scalefactor is such that that a *positive* change *increases* the magnitude of x_quant, and so *decreases* the distortion and *increases* the number of bits used.

The sign of the *mdct_line* is saved separately and added again only for counting the bits and encoding the bitstream.

B.2.7.4.1 Outer iteration loop (distortion control loop)

The outer iteration loop controls the quantization noise which is produced by the quantization of the frequency domain lines within the inner iteration loop. The coloring of the noise is done by multiplication of the lines within scalefactor bands with the actual scalefactors before doing the quantization. The following pseudo-code illustrates the multiplication.

```
do for each scalefactor band sb:
    do from lower index to upper index i of scalefactor band
        mdct_scaled(i) = abs(mdct_line(i))^(3/4) * 2^(3/16  * scalefactor(sb))
    end do
end do
```

B.2.7.4.2 Call of inner iteration loop

For each outer iteration loop (distortion control loop) the inner iteration loop (rate control loop) is called. The parameters are the frequency domain values with the scalefactors applied to the values within the scalefactor bands (mdct_scaled(0..1023)), a start value for common_scalefac, and the number of bits which are available to the rate control loop. The result is the number of bits actually used and the quantized frequency lines x_quant(i), and a new common_scalefac.
The formula to calculate the quantized MDCT coefficients is:

*x_quant(i) = int ((mdct_scaled (i) * 2^(-3/16 * common_scalefac)) + MAGIC_NUMBER)*

The bits, that would be needed to encode the quantizes values and the side information (scalefactors etc.) are counted according to the bitstream syntax, described in [A 2.8 Noiseless Coding].

B.2.7.4.3 Amplification of scalefactor bands which violate the masking threshold

The calculation of the distortion (*error_energy(sb)*)of the scalefactor band is done as follows:

```
do for each scalefactor band sb:
    error_energy(sb)=0
    do from lower index to upper index i of scalefactor band
        error_energy(sb) = error_energy(sb) + (abs( mdct_line(i))
                    - (x_quant(i) ^(4/3) * 2^( -¼ * (scalefactor(sb) -common_scalefac ))))^2
    end do
end do
```

All spectral values of the scalefactor bands which have a distortion that exceeds the allowed distortion (*xmin(sb)*)are amplified according to formula in B 2.7.4.1 (“Outer Iteration Loop“), the new scalefactors can be calculated according to this pseudocode:

```
    do for each scalefactor band sb
        if ( error_energy(sb) > xmin(sb) ) then
            scalefactor(sb) = scalefactor(sb) + 1
        end if
    end do
```