INTERNATIONAL STANDARD ISO/IEC 14496-3:2009
TECHNICAL CORRIGENDUM 3

Published 2012-09-01

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

Information technology — Coding of audio-visual objects —

Part 3: **Audio**

TECHNICAL CORRIGENDUM 3

Technologies de l'information — Codage des objets audiovisuels —

Partie 3: Audio

RECTIFICATIF TECHNIQUE 3

Technical Corrigendum 3 to ISO/IEC 14496-3:2009 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

In 12.3 Payloads for the audio object, replace Table 12.2:

Syntax	No. of bits	Mnemonics

lle_element()
{
for (ch=0;ch<channel_number;) {
if (is_channel_pair(ch)) {
lle_channel_pair_element();
ch += 2;
} else {
lle_single channel_element();
ch++;
}
}
}		

ICS 35.040 **Ref. No. ISO/IEC 14496-3:2009/Cor.3:2012(E)**

Published in Switzerland

with:

```
Syntax                                                          No. of bits   Mnemonics
lle_element()
{
        for (el=0;el<getNrOfElements(channelConfiguration);) {
            switch(getNextElement(el, channelConfiguration)){
            case LLE_SCE:
                lle_single channel_element();
                break;
            case LLE_LFE:
                lle_lfe channel_element();
                break;
            case LLE_CPE:
                lle_channel_pair_element();
                break;
            }
            el++;
        }
}
```

In 12.3 Payloads for the audio object, add at the end:

Table 12.9 — Syntax of lle_lfe_channel_element

```
Syntax                                                          No. of bits   Mnemonics
lle_lfe_channel_element()
{
    lle_individual_channel_stream(1);
}
```

In 12.3 Payloads for the audio object, add at the end:

The functions getNrOfElements(channelConfiguration) and getNextElement(el, channelConfiguration) are defined as follows:

```
UINT32 getNrOfElements(channelConfiguration)

{

  UINT32 NrOfFixChannelConfigElements[7] = { 1, 1, 2, 3, 3, 4, 5 };



  if(channelConfiguration == 0){

    return num_front_channel_elements + num_side_channel_elements  +

           num_back_channel_elements  + num_lfe_channel_elements;

  } else {

    return NrOfFixChannelConfigElements[channelConfiguration - 1];
```

```
  }
}
typedef enum {
  LLE_SCE = 0,
  LLE_CPE,
  LLE_LFE,
  LLE_INV      /*dummy element for array */
} LLE_Element;
LLE_Element getNextElement(el, channelConfiguration)
{
  LLE_Element FixChannelConfigElements[7][5] =
  { {LLE_SCE, LLE_INV, LLE_INV, LLE_INV, LLE_INV},
    {LLE_CPE, LLE_INV, LLE_INV, LLE_INV, LLE_INV},
    {LLE_SCE, LLE_CPE, LLE_INV, LLE_INV, LLE_INV},
    {LLE_SCE, LLE_CPE, LLE_SCE, LLE_INV, LLE_INV},
    {LLE_SCE, LLE_CPE, LLE_CPE, LLE_INV, LLE_INV},
    {LLE_SCE, LLE_CPE, LLE_CPE, LLE_LFE, LLE_INV},
    {LLE_SCE, LLE_CPE, LLE_CPE, LLE_CPE, LLE_LFE} };

  if(channelConfiguration == 0){
    UINT el_cnt = 0, el_front = 0, el_side = 0, el_back = 0, el_lfe = 0;

    for(;el_front < num_front_channel_elements; el_cnt++, el_front++){
      if(el_cnt == el){
        if(front_element_is_cpe[el_front] == 1){
          return LLE_CPE;
        } else {
          return LLE_SCE;
        }
    }
```

```
    for(;el_side < num_side_channel_elements; el_cnt++, el_side++){
      if(el_cnt == el){
        if(side_element_is_cpe[el_side] == 1){
          return LLE_CPE;
        } else {
          return LLE_SCE;
        }
    }

    for(;el_back < num_back_channel_elements; el_cnt++, el_back++){
      if(el_cnt == el){
        if(back_element_is_cpe[el_back] == 1){
          return LLE_CPE;
        } else {
          return LLE_SCE;
        }
    }

    for(;el_lfe < num_lfe_channel_elements; el_cnt++, el_lfe++){
      if(el_cnt == el){
        return LLE_LFE;
      }
    }
  } else {
    return FixChannelConfigElements [channelConfiguration - 1][el];
  }

  return LLE_INV;
}
```

The elements of array `NrOfFixChannelConfigElements` define the number of syntactic elements of Channel Configuration 1-7 (corresponding to Table 1.19 – Channel Configuration).

The elements of array `FixChannelConfigElements` define the syntactic elements of Channel Configuration 1-7 in order received (corresponding to Table 1.19 – Channel Configuration).

The values `num_front_channel_elements`, `num_side_channel_elements`, `num_back_channel_elements`, `num_lfe_channel_elements` and the arrays

`front_element_is_cpe`, `side_element_is_cpe` and `back_element_is_cpe` are defined in the program_conig_element() inside the SLSSpecificConfig(), transmitted in case of channelConfiguration 0.

In 12.4 Semantics, replace:

Data elements:

with:

12.4.1 SLSSpecificConfig

In 12.4 Semantics, after Table 12.10 Length of the IntMDCT frame, add:

12.4.2 Program_config_element (PCE)

See 4.5.1.2 "Program Config Element (PCE) within the GaSpecificConfig").

The following changes apply in the context of MPEG-4 SLS:

element_id Shall be set to 0x0. All other values reserved for future use.

object_type Shall be set to 0x1. All other values reserved for future use.

sampling_frequency_index Indicates the sampling frequency of the program according to the table defined in 1.6.3.4 (samplingFrequencyIndex). The escape value is not permitted. If the sampling rate is not listed in Table 1.18 in 1.6.3.4., then table 4.82 in 4.5.1.1 shall be used.

In 12.5 SLS decoder tool, replace:

12.5.4 Decoding of lle_single_channel_element (LLE_SCE) and lle_channel_pair_element (LLE_CPE)

with:

12.5.4 Decoding of lle_single_channel_element (LLE_SCE), lle_channel_pair_element (LLE_CPE) and lle_lfe_channel_element(LLE_LFE)

In 12.5 SLS decoder tool, replace:

12.5.4.2.1 LLE_SCE and LLE_CPE

An LLE_SCE is composed of an lle_individual_channel_stream (LLE_ICS) while an LLE_CPE has two lle_individual_channel_streams (LLE_ICS).

with:

12.5.4.2.1 LLE_SCE, LLE_CPE and LLE_LFE

An LLE_SCE and LLE_LFE is composed of a lle_individual_channel_stream (LLE_ICS) while an LLE_CPE has two lle_individual_channel_streams (LLE_ICS).

In 12.5.4.2.3 Recovering BPGC/CBAC side information, add after "For scalefactor bands coded with IS or PNS the values of inv_quant(x) is set to 0.":

For scalefactor bands coded with IS the non M/S core_scaling_factor must be used.

In 12.5.8 Integer Mid/Side process, add after "…, the inverse integer M/S processing has to be applied to the scale factor bands where the M/S flag is switched on":

Scalefactor bands coded with IS always must be treated as if the M/S flag is switched off.

In 12.5.8 Integer Mid/Side process, add after "Hence M/S has to be applied to the scale factor bands where the M/S flag is switched off":

Scalefactor bands coded with IS always must be treated as if the M/S flag is switched off.

In 12.5 SLS decoder tool, add at the end:

12.5.12 Decoding of PNS and IS scalefactor bands

If PNS or Intensity Stereo is used in the core layer, the M/S flag can be used to signal PNS / IS specific information. Therefore IS scalefactor bands must be treated in the SLS layer as if the M/S flag is switched off during the whole decoding process. For PNS the M/S flag indicates a correlation between the two channels of a channel pair element and therefore must be evaluated.

In 12.A.4 Integer Mid/Side, add at the end:

Scalefactor bands coded with IS always must be treated as if the M/S flag is switched off.