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;
}
```