INTERNATIONAL STANDARD

ISO/IEC 23003-4

Second edition
2020-06

AMENDMENT 1
2022-07

Information technology — MPEG audio technologies —

Part 4: Dynamic range control

AMENDMENT 1: Side chain normalization

Partie 4: Contrôle de gamme dynamique

AMENDEMENT 1: Normalisation de l'entrée latérale

Reference number
ISO/IEC 23003-4:2020/Amd. 1:2022(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see https://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia, and hypermedia*.

A list of all parts in the ISO/IEC 23003 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Information technology — MPEG audio technologies —

Part 4:
Dynamic range control

AMENDMENT 1: Side chain normalization

6.1.1

In the second last paragraph replace "UNIDRCCONFEXT_V1" with "UNIDRCCONFEXT_V1 or UNIDRCCONFEXT_V2".

6.1.2.3

Add, after the fourth paragraph, the following new paragraph:

The drcCoefficientsUniDrc() payload for ISO/IEC 14496-12 (see Table 69) for *version*=2 and *characteristicV1Override*=1 carries essentially the same information as the extension UNIDRCCONFEXT_V2. The corresponding bitstream fields are coded the same way as specified in Table A.10.

6.4.6

Add, after the fourth paragraph, the following new paragraph:

If the encoder-side characteristic is provided in the bitstream, it is recommended to use linear gain interpolation. ISO/IEC 23091-3 (CICP) encoder characteristics in the range of 65 to 70 are only supported by drcCoefficientsUniDrcV1() as part of a UNIDRCCONFEXT_V2 extension. These characteristics shall not be used otherwise. When a CICP characteristic in this range is inverted in the decoder, loudness normalization shall be applied after the inversion based on the available loudness metadata and encoder normalization gain, if applicable. This is shown in the pseudo code of Table E.3, where the output of the inverse characteristic is computed with an offset depending on the value of *sourceLoudness* and *encDrcNormGainDb*. *sourceLoudness* is the integrated DRC input loudness at the encoder before any normalization. The value of *sourceLoudness* is obtained from the loudness metadata for the DRC. *encDrcNormGainDb* is the signal gain in dB applied at the encoder DRC input. This gain value is available in a UNIDRCCONFEXT_V2 extension payload to support legacy devices when *characteristicV1Override*==1 (see also E.4). The value of *drcInputLoudnessTarget* is the target input loudness of the DRC characteristic applied to generate the DRC gains in the decoder. The drcCoefficientsUniDrc() payload of the Base Media File Format ISO/IEC 14496-12 supports CICP characteristics 65 to 70 only if *version*>=2 (see also Table 69).

6.4.6

Replace Table 17 with the following table:

Table 17 — Conversion of a DRC gain sample and associated slope from dB to linear domain (*slopeIsNegative*==1 if the source DRC characteristic has a negative slope)

```
toLinear (gainDb, slopeDb) {
    SLOPE_FACTOR_DB_TO_LINEAR = 0.1151f;        /* ln(10) / 20 */
    EFFECT_BIT_CLIPPING = 0x0100;               /* drcSetEffect 9 (Clip.Prev.) */
    EFFECT_BIT_FADE   = 0x0200;                 /* drcSetEffect 10 (Fade) */
    EFFECT_BITS_DUCKING = 0x0400 | 0x0800;      /* drcSetEffect 11 or 12 (Ducking) */
    gainRatio = 1.0;
    gainDbMod = gainDb;
    if (((drcSetEffect & EFFECT_BITS_DUCKING) == 0) &&
                (drcSetEffect != EFFECT_BIT_FADE) &&
                (drcSetEffect != EFFECT_BIT_CLIPPING)) {
        if (drcCharacteristicTarget > 0) {
            gainDbMod = mapGain(gainDb); /* target characteristic from host */
        }
        else if (drcCoefficientsUniDrcV1Present == 1) {
            if (((gainDb >= 0.0) && (slopeIsNegative == 1)) || ((gainDb <= 0.0) && (slopeIsNegative == 0))) {
                if (targetCharacteristicLeftPresent == 1) {
                    gainDbMod = mapGain(gainDb); /* target characteristic in payload */
                }
            }
            else if (((gainDb <= 0.0) && (slopeIsNegative == 1)) || ((gainDb >= 0.0) && (slopeIsNegative == 0))) {
                if (targetCharacteristicRightPresent == 1) {
                    gainDbMod = mapGain(gainDb); /* target characteristic in payload */
                }
            }
        }
        if (gainDbMod < 0.0) {
            gainRatio *= compress;
        }
        else {
            gainRatio *= boost;
        }
    }
    if (gainScalingPresent) {
        if (gainDbMod < 0.0) {
            gainRatio *= attenuationScaling;
        }
        else {
            gainRatio *= amplificationScaling;
        }
    }
    if (duckingScalingPresent && (drcSetEffect & EFFECT_BITS_DUCKING)) {
        gainRatio *= duckingScaling;
    }
    gainLin = pow(2.0, gainRatio * gainDbMod / 6.0);
    slopeLin = SLOPE_FACTOR_DB_TO_LINEAR * gainRatio * gainLin * slopeDb;
    if (gainOffsetPresent) {
```

```
            gainLin *= pow(2, gainOffset/6.0);
        }
        /* The only drcSetEffect is "clipping prevention" */
        if (limiterPeakTargetPresent && (drcSetEffect == EFFECT_BIT_CLIPPING)) {
            gainLin *= pow(2, max(0.0, -limiterPeakTarget-loudnessNormalizationGainDb
                        -loudnessNormalizationGainModificationDb)/6.0);
            if (gainLin >= 1.0) {
                gainLin = 1.0;
                slopeLin = 0.0;
            }
        }
        return (gainLin, slopeLin);
}
```

7.3

Replace Table 69 with the following table:

Table 69 — Syntax of drcCoefficientsUniDrc() payload for ISO/IEC 14496-12

```
aligned(8) class DRCCoefficientsUniDrc extends FullBox('udc2', version, flags=0) {
   // N copies of this box, one of these per DRC_location, plus boxes with characteristicV1Override ==
1
   characteristicV1Override = 0;
  if (version >= 2) {
    bit(1) characteristicV1Override;
    if (characteristicV1Override == 1) {
      bit(4) reserved = 0;
      signed int(5)     DRC_location;
      unsigned int(6) gainSetCount;
      for (j=1; j<=gainSetCount; j++) {
        bit(3) reserved = 0;
        bit(1) characteristicOverridePresent;
        unsigned int(4) bandCount;
        if (characteristicOverridePresent == 1) {
            for (k=1; k<=bandCount; k++) {
                bit(3) reserved = 0;
                unsigned int(7) overrideCicpCharacteristic;
                unsigned int(6) bsEncDrcNormGain;
              }
            }
         }
       }
  }
  if (characteristicV1Override == 0) {
       if (version < 2) {
```

```
    bit(1) reserved = 0;
}
bit(1)              reserved = 0;
signed int(5)     DRC_location;
bit(1)              drc_frame_size_present;
if (drc_frame_size_present == 1) {
    bit(1)                  reserved = 0;
    unsigned int(15)    bs_drc_frame_size;
}
if (version >= 1) {
    bit(5) reserved = 0;
    bit(1) drc_characteristic_left_present;
    bit(1) drc_characteristic_right_present;
    bit(1) shape_filters_present;
    if (drc_characteristic_left_present == 1) {
        bit(4)                  reserved = 0;
        unsigned int(4)    characteristic_left_count;
        for (k=1; k<=characteristic_left_count; k++) {
            bit(7)              reserved = 0;
            unsigned int(1)    characteristic_format;
            if (characteristic_format==0) {
                bit(1)                  reserved = 0;
                unsigned int(6)    bs_gain_left;
                unsigned int(4)    bs_io_ratio_left;
                unsigned int(4)    bs_exp_left;
                bit(1)                 flip_sign_left;
            } else {
                bit(6)                  reserved = 0;
                unsigned int(2)     bs_char_node_count;
                for (n=1; n<=bs_char_node_count+1; n++) {
                bit(3)                  reserved = 0;
                unsigned int(5)     bs_node_level_delta;
                unsigned int(8)     bs_node_gain;
              }
            }
        }
    }
    if (drc_characteristic_right_present == 1) {
        bit(4)                  reserved = 0;
        unsigned int(4)    characteristic_right_count;
        for (k=1; k<=characteristic_right_count; k++) {
            bit(7)                  reserved = 0;
            unsigned int(1)     characteristic_format;
            if (characteristic_format==0) {
```

```
            bit(1)                reserved = 0;
            unsigned int(6)    bs_gain_right;
            unsigned int(4)    bs_io_ratio_right;
            unsigned int(4)    bs_exp_right;
            bit(1)                flip_sign_right;
          } else {
            bit(6)                reserved = 0;
            unsigned int(2)    bs_char_node_count;
            for (n=1; n<=bs_char_node_count+1; n++) {
              bit(3)                reserved = 0;
              unsigned int(5)    bs_node_level_delta;
              unsigned int(8)    bs_node_gain;
            }
          }
        }
      }
      if (shape_filters_present==1) {
        bit(4)        reserved = 0;
        unsigned int(4)    shape_filter_count;
        for (k=1; k<=shape_filter_count; k++) {
          bit(4)  reserved = 0;
          bit(1)  LF_cut_filter_present;
          bit(1)  LF_boost_filter_present;
          bit(1)  HF_cut_filter_present;
          bit(1)  HF_boost_filter_present;
          if (LF_cut_filter_present) {
            bit(3)                reserved = 0;
            unsigned int(3)    LF_corner_freq_index;
            unsigned int(2)    LF_filter_strength_index;
          }
          if (LF_boost_filter_present) {
            bit(3)                reserved = 0;
            unsigned int(3)    LF_corner_freq_index;
            unsigned int(2)    LF_filter_strength_index;
          }
          if (HF_cut_filter_present) {
            bit(3)                reserved = 0;
            unsigned int(3)    HF_corner_freq_index;
            unsigned int(2)    HF_filter_strength_index;
          }
          if (HF_boost_filter_present) {
            bit(3)                reserved = 0;
            unsigned int(3)    HF_corner_freq_index;
            unsigned int(2)    HF_filter_strength_index;
```

```
            }
        }
    }
}
bit(1)                  reserved = 0;
unsigned int(1)    delayMode;
if (version >= 1} {
    bit(2)                  reserved = 0;
    unsigned int(6)    gain_sequence_count;
}
unsigned int(6)    gain_set_count;
for (i=1; i<=gain_set_count; i++) {
    bit(2)                  reserved = 0;
    unsigned int(2)    gain_coding_profile;
    unsigned int(1)    gain_interpolation_type;
    unsigned int(1)    full_frame;
    unsigned int(1)    time_alignment;
    bit(1)                  time_delta_min_present;
    if (time_delta_min_present == 1) {
        bit(5)                  reserved = 0;
        unsigned int(11)    bs_time_delta_min;
    }
    if (gain_coding_profile!=3) {
        bit(3)                  reserved = 0;
        unsigned int(4)    band_count;          // shall be >= 1
        unsigned int(1)    drc_band_type;
        for (j = 1; j <= band_count; j++) {
            if (version>=1) {
                unsigned int(6)    bs_index;
                bit(1)                  drc_characteristic_present
                bit(1)                  drc_characteristic_format_is_CICP
                if (drc_characteristic_present==1) {
                    if (drc_characteristic_format_is_CICP==1) {
                        bit(1)                  reserved;
                        unsigned int(7)    drc_characteristic;
                    } else {
                        unsigned int(4)    drc_characteristic_left_index;
                        unsigned int(4)    drc_characteristic_right_index;
                    }
                }
            } else {
                bit(1)                  reserved = 0;
                unsigned int(7)    drc_characteristic;
            }
```

```
            }
            for (j = 2; j <= band_count; j++){
                if (drc_band_type == 1) {
                bit(4)                  reserved = 0;
                unsigned int(4)    crossover_freq_index;
                } else {
                bit(6)                  reserved = 0;
                unsigned int(10)    start_sub_band_index;
                }
            }
        }
    }
  }
}
```

7.3

Replace Table 75 with the following table:

Table 75 — Syntax of uniDrcConfigExtension() payload

Syntax	No. of bits	Mnemonic
uniDrcConfigExtension()		
{		
while (**uniDrcConfigExtType !**= UNIDRCCONFEXT_TERM) {	**4**	**uimsbf**
extSizeBits = **bitSizeLen** + 4;	**4**	**uimsbf**
extBitSize = **bitSize** + 1;	**extSizeBits**	**uimsbf**
switch (uniDrcConfigExtType) {		
case UNIDRCCONFEXT_PARAM_DRC:		
drcCoefficientsParametricDrc();		
parametricDrcInstructionsCount;	**4**	**uimsbf**
for (i=0; i<parametricDrcInstructionsCount; i++) {		
parametricDrcInstructions ();		
}		
break;		
case UNIDRCCONFEXT_V2:		
if (**characteristicV1Override**==1) {	**1**	**bslbf**
drcCoefficientsOverrideCount;	**3**	**uimsbf**
for (i=0; i<drcCoefficientsOverrideCount; i++) {		
drcLocation;	**4**	**uimsbf**
gainSetCount;	**6**	**uimsbf**
for (j=0; j<gainSetCount; j++) {		
if (**characteristicOverridePresent**==1) {	**1**	**bslbf**
bandCount;	**4**	**uimsbf**
for (k=0; k<bandCount; k++) {		
overrideCicpCharacteristic;	**7**	**uimsbf**