INTERNATIONAL STANDARD

ISO/IEC 23003-4

First edition
2015-11-15

Information technology — MPEG audio technologies —

Part 4: Dynamic Range Control

Technologies de l'information — Technologies audio MPEG —

Partie 4: Contrôle de gamme dynamique

Reference number
ISO/IEC 23003-4:2015(E)

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Contents

Page

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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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).

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).

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

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT), see the following URL: Foreword — Supplementary information.

The committee responsible for this document is ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia, and hypermedia*.

ISO/IEC 23003 consists of the following parts, under the general title *Information technology — MPEG audio technologies*:

— *Part 1: MPEG Surround*

— *Part 2: Spatial Audio Object Coding*

— *Part 3: Unified speech and audio coding*

— *Part 4: Dynamic Range Control*

Introduction

Consumer audio systems and devices are used in a large variety of configurations and acoustical environments. For many of these scenarios, the audio reproduction quality can be improved by appropriate control of content dynamics and loudness.

This part of ISO/IEC 23003 provides a universal dynamic range control tool that supports loudness normalization. The DRC tool offers a bitrate efficient representation of dynamically compressed versions of an audio signal. This is achieved by adding a low-bitrate DRC metadata stream to the audio signal. The DRC tool includes dedicated sections for clipping prevention, ducking, and for generating a fade-in and fade-out to supplement the main dynamic range compression functionality. The DRC effects available at the DRC decoder are generated at the DRC encoder side. At the DRC decoder side, the audio signal may be played back without applying the DRC tool, or an appropriate DRC tool effect is selected and applied based on the given playback scenario.

Information technology — MPEG audio technologies —

Part 4:
Dynamic Range Control

1 Scope

This part of ISO/IEC 23003 specifies technology for loudness and dynamic range control. This International Standard is applicable to most MPEG audio technologies. It offers flexible solutions to efficiently support the widespread demand for technologies such as loudness normalization and dynamic range compression for various playback scenarios.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 14496-12, *Information technology — Coding of audio-visual objects — Part 12: ISO base media file format*

ISO/IEC 23001-8, *Information technology — MPEG systems technologies — Part 8: Coding-independent code points*

3 Terms, definitions and mnemonics

For the purposes of this document, the terms and definitions given in ISO/IEC 14496-12 and the following apply.

3.1 Terms

3.1.1
DRC sequence
series of DRC gain values that can be applied to one or more audio channels

3.1.2
DRC set
defined set of DRC sequences that produce a desired effect if applied to the audio signal

3.1.3
album
collection of audio recordings that are mastered in a consistent way. Traditionally, a collection of songs released on a Compact Disk belongs into this category, for example

3.2 Mnemonics

bslbf	bit string, left bit first, where “left” is the order in which bit strings are written in ISO/IEC 14496. Bit strings are written as a string of 1s and 0s within single quote marks, for example ‘1000 0001’. Blanks within a bit string are for ease of reading and have no significance
uimsbf	unsigned integer, most significant bit first
vlclbf	variable length code, left bit first, where “left” refers to the order in which the variable length codes are written
bit(n)	a bit string with n bits in the same format as bslbf
unsigned int(n)	an unsigned integer with n bits in the same format as uimsbf
signed int(n)	a signed integer with n bits, most significant bit first

4 Symbols (and abbreviated terms)

a_i	Filter coefficient
b	Band index of DRC filter bank (starting at 0)
b_i	Filter coefficient
deltaTmin	Smallest permitted DRC gain sample interval in units of the audio sample interval.
f_c	Cross-over frequency in Hz
$f_{c,norm}$	Cross-over frequency expressed as fraction of the audio sample rate.
$f_{c,norm,SB}(s)$	Cross-over frequency of audio decoder sub-band s expressed as fraction of the audio sample rate. The cross-over frequency is the upper band edge frequency of the sub-band.
f_s	Audio sample rate in Hz. If an audio decoder is present, it is the sample rate of the decoded time-domain audio signal.
N_{DRC}	Maximum permitted number of DRC samples per DRC frame. Identical to the number of intervals with a duration of *deltaTmin* per DRC frame.
N_{Codec}	Codec frame size in units of the audio sample interval $1/f_s$
M_{DRC}	DRC frame size in units of the audio sample interval $1/f_s$
π	Ratio of a circle’s circumference to its diameter
s	Audio decoder sub-band index (starting at 0)
TRUE/FALSE	Values of Boolean data type, which correspond to numerical 1 and 0, respectively.
z	Complex variable of the z-transform

5 Technical overview

The technology described in this part of ISO/IEC 23003 is called DRC tool. It provides efficient control of dynamic range, loudness, and clipping based on metadata generated at the encoder. The decoder can choose to selectively apply the metadata to the audio signal to achieve a desired result. Metadata for dynamic range compression consists of encoded time-varying gain values that can be applied to the audio signal. Hence, the main blocks of the DRC tool include a DRC gain encoder, a DRC gain decoder, a DRC gain modification block, and a DRC gain application block. These blocks are exercised on a frame-by-frame basis during audio processing. Various DRC configurations can be conveyed in a separate bitstream element, such as configurations for a downmix or combined DRCs. The DRC set selection block decides based on the playback scenario and the applicable DRC configurations which DRC gains to apply to the audio signal. Moreover, the DRC tool supports loudness normalization based on loudness metadata.

A typical system for loudness and dynamic range control in the time domain is shown in Figure 1. A more complex system including downmixer and peak limiter is shown in Figure 2. The decoder part of the DRC tool is driven by metadata that efficiently represents the DRC gain samples and parameters for interpolation. The gain samples can be updated as fast as necessary to accurately represent gain changes down to at least 1 ms update intervals. In the following the decoder part of the DRC tool is referred to as "DRC decoder", which includes everything except the audio decoder and associated bitstream de-multiplexing.

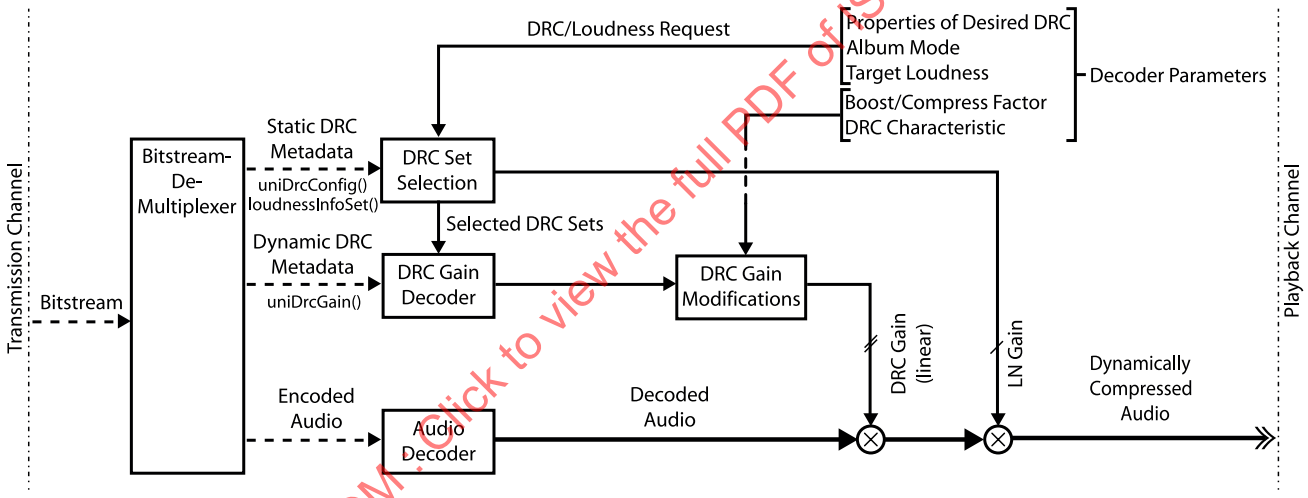

Figure 1 — Block diagram of a typical system with audio decoder and DRC tool modules to achieve loudness normalization (LN) and dynamic range control

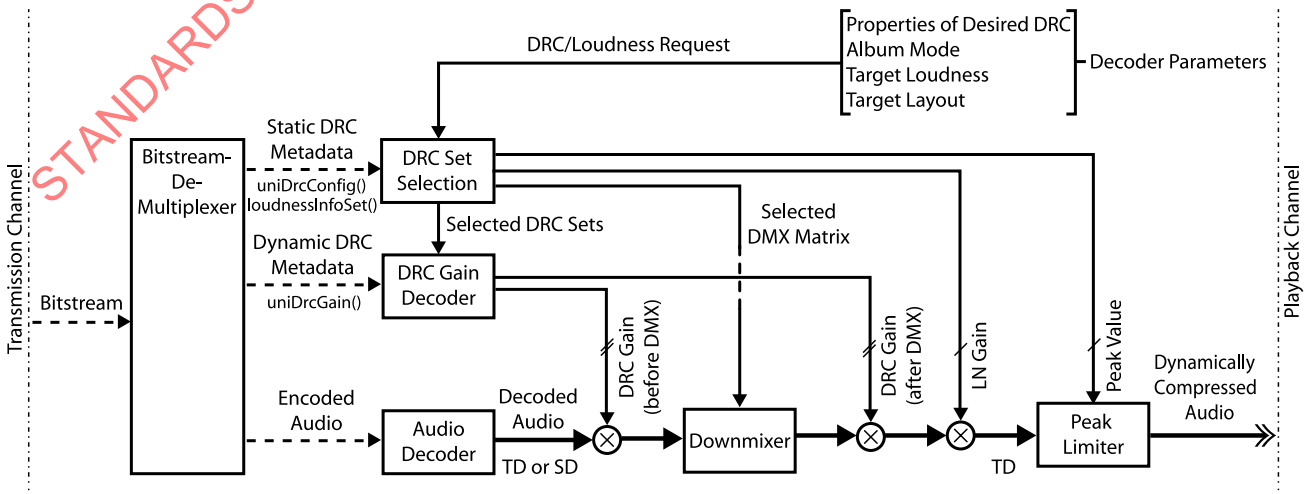

Figure 2 — Block diagram of a more complex system including downmixer and peak limiter (TD = time-domain, SD = subband-domain)

6 DRC decoder

6.1 DRC decoder configuration

6.1.1 Overview

The DRC configuration information can be received in-stream using the static payloads uniDrcConfig() and loudnessInfoSet() described below, or it can be delivered by a higher layer, such as 14496-12 (see Table 1). The basic decoding process of the static information is virtually the same. The difference consists mainly in a few syntax changes and reduced field sizes to increase the bit rate efficiency of the in-stream configuration. The syntax of the in-stream static payload is given in 7.3. The associated metadata encoding is given inA.6. The static DRC payload is evaluated once at the beginning of the decoding process and it is monitored subsequently. For static DRC payload changes during playback see 6.8.

Table 1 — Overview of configuration (setup) and separate metadata track in ISO/IEC 14496-12

	Sample Entry Code	*Setup (in sample entry)*	*Track reference*	*Sample format*
Audio Track	As specified for the audio codec in use (unchanged)	DRCInstructions box using negative values for drcLocation	'adrc' referring to the metadata tracks carrying gain values	As specified for the audio codec in use (unchanged)
Metadata Track	'unid'	(none)	(none)	Each sample is a uniDrcGain() payload

The static payload is divided into five logical blocks:

— channelLayout();

— downmixInstructions();

— drcCoefficientsBasic(), drcCoefficientsUniDrc();

— drcInstructionsBasic(), drcInstructionUniDrc();

— loudnessInfo().

Except for the channelLayout(), multiple instances of a logical block can appear. The DRC decoder combines the information of the matching instances of up to five logical blocks for a given playback scenario. Matching instances are found by matching several identifiers (labels) contained in the blocks.

From the static payload the decoder can also extract information about the effect of a particular DRC and various associated loudness information, if present. If multiple DRCs are available, this information can be used to select a particular DRC based on target criteria for dynamics and loudness (see 6.3)

uniDrcConfig() contains all blocks except for the loudnessInfo() blocks which are bundled in loudnessInfoSet(). The last part of the uniDrcConfig() payload can include future extension payloads. In the event that a *uniDrcConfigExtType* value is received that is not equal to UNIDRCCONFEXT_TERM, the DRC tool parser must read and discard the bits (otherBit) of the extension payload. Similarly, the last part of the loudnessInfoSet() payload can include future extension payloads. In the event that a *loudnessInfoSetExtType* value is received that is not equal to UNIDRCLOUDEXT_TERM, the DRC tool parser must read and discard the bits (otherBit) of the extension payload.

The top level fields of uniDrcConfig() include the audio sample rate, which is a fundamental parameter for the decoding process (if not present, the audio sample rate is inherited from the employed audio codec). Moreover, the top level fields of uniDrcConfig() include the number of instances of each of the logical blocks, except for the channelLayout() block which appears only once. The top level fields of loudnessInfoSet() only include the number of loudnessInfo() blocks. The five logical blocks are described in the following.

6.1.2 Description of logical blocks

6.1.2.1 channelLayout()

The channelLayout() block includes the channel count of the audio signal in the base layout. It may also include the base layout unless it is specified elsewhere. For use cases where the base audio signal represents objects or other audio content, the channel count represents the total number of base content channels.

6.1.2.2 downmixInstructions()

This block includes a unique non-zero downmix identifier (downmixId) that can be used externally to refer to this downmix. The targetChannelCount specifies the number of channels after downmixing to the target layout. It may also contain downmix coefficients, unless they are specified elsewhere. For use cases where the base audio signal represents objects or other audio content, the downmixId can be used to refer to a specific target channel configuration of a present rendering engine.

6.1.2.3 drcCoefficientsBasic(), drcCoefficientsUniDrc()

A drcCoefficients block describes all available DRC gain sequences in one location. The block can have the basic format or the uniDrc format. The basic format, drcCoefficientsBasic(), contains a subset of information included in drcCoefficientsUniDrc() that can be used to describe DRCs other than the ones specified in this standard. drcCoefficientsUniDrc() contains for each sequence several indicators on how it is encoded, the time resolution, time alignment, the number of DRC sub-bands and corresponding crossover frequencies and DRC characteristics. The crossover frequencies must increase with increasing band index. Alternatively, explicit indices in a decoder sub-band domain can be specified for the assignment of DRC sub-bands. The sub-band indices must also increase with increasing band index. If the DRC gains are applied in the time-domain by using the multi-band DRC filter bank specified in 6.4.11, explicit index signalling is not allowed. The index of the DRC characteristic indicates which compression characteristic was used to produce the gain sequence. The DRC location describes where these gain sequences can be found in the bitstream. The DRC gain sequences in that location are inherently enumerated according to their order of appearance starting with 1.

The DRC location field encoding depends on the audio codec. A codec specification may include this specification, and use values 1 – 4 to refer to codec-specific locations as indicated in Table 1. For example, for AAC (ISO/IEC 14496-3), the codec-specific values of the DRC location field are encoded as shown in Table 3.

Table 2 — Encoding of drcLocation for in-stream payload

drcLocation n	Payload
0	*Reserved*
1	Location 1 (Codec-specific use)
2	Location 2 (Codec-specific use)
3	Location 3 (Codec-specific use)
4	Location 4 (Codec-specific use)
$n > 4$	*reserved*

Table 3 — Codec-specific encoding of drcLocation for MPEG-4 Audio

drcLocation n	Payload
1	uniDrc() (defined in Clause 7)
2	dyn_rng_sgn[i] / dyn_rng_ctl[i] in dynamic_range_info() (defined in ISO/IEC 14496-3:2009 subpart 4)
3	compression_value in MPEG4_ancillary_data() (defined in ISO/IEC 14496-3:2009/AMD 4:2013)
4	*reserved*

The DRC frame size can optionally be specified. It must be provided if the DRC frame size deviates from the default size specified in 6.4.2. If not specified, the default frame size is used.

The in-stream drcCoefficient syntax is given in Table 42 and Table 44. The syntax for the corresponding block for ISO/IEC 14496-12 (ISO base media file format) is shown in Table 43 and Table 45. The corresponding blocks carry essentially the same information. Values that are identically included in both blocks are coded the same way except for drcLocation.

In ISO base media file format (see ISO/IEC 14496-12), for each codec that can be carried in MP4 files and that also carries DRC information, there is a specific definition of how the location is coded, using the DRC_location field (see Table 4). A negative value of DRC_location indicates that a DRC payload is in an associated meta-data track. That track is the n-th linked via a track reference of type 'adrc' (audio DRC) from the audio track, where *n* = abs(DRC_location), and the sample-entry type in the meta-data track indicates in which format the coefficients are stored. Table 3 defines the specific entries of the drcLocation field for AAC. Some example use cases are discussed in C.10.

If the uniDrc() payload is stored in a separate track in the ISO base media file format (ISO/IEC 14496-12), then the track is a metadata track with the sample entry identifier 'unid' (uniDrc), with no required boxes added to the sample entry. The time synchronization with the linked audio track is the same as if the payload was in-stream.

Table 4 — Encoding of drcLocation for ISO/IEC 14496-12

drcLocation n	Payload
$n < 0$	*DRC payload located in |n|-th linked meta-data track*
0	*reserved*
1	Location 1 (Codec-specific use)
2	Location 2 (Codec-specific use)
3	Location 3 (Codec-specific use)
4	Location 4 (Codec-specific use)
$n > 4$	*reserved*

6.1.2.4 drcInstructionsBasic(), drcInstructionsUniDrc()

A drcInstructions block includes information about one specific DRC set that can be applied to achieve a desired effect. This block can have the basic format or the uniDrc format. The basic format, drcInstructionsBasic(), contains a subset of information included in drcInstructionsUniDrc() that can be used to describe DRCs other than the ones specified in this standard. The information included in drcInstructionsUniDrc() consists mainly of pre-defined description elements such as the DRC set effect and the DRC gain sequences that are applied. The drcSetEffect field contains several effect bits as listed in Table A.32. Multiple bits can be set unless otherwise noted. Note that if no effect bit is set at all, the DRC set is ignored in the DRC set selection (see 6.3). Each drcInstructions block carries a unique non-zero identifier drcSetId. A downmixId is included to indicate if this DRC set applies to a certain downmix with this identifier. A downmixId of zero indicates that the DRC set is applied to the base layout. A downmixId of 0x7F indicates that the DRC set can be applied before or after the downmix.

Since such a DRC can be applied to any downmix, it has only one channel group including all channels. If a "Ducking" bit is set in the drcSetEffect field, the DRC set is applied before any downmix specified by the downmix ID, i.e. the DRC set is always applied to the base layout and the downmix is generated thereafter. The downmixId 0x7F is not permitted for a ducking DRC set. In all other cases, the DRC set is applied to the channel configuration indicated by the downmix ID.

A second DRC set may be specified for certain configurations. These configurations include cases where, e.g. one DRC set is used for dynamic range compression and the other for clipping prevention ("Clipping" bit is set); or, e.g. one DRC set is applied before and the other after the downmix. In those cases, the second DRC set contains a non-zero field *dependsOnDrcSet* that has the value of the *drcSetId* of the first DRC set it depends on. The declared DRC set effects of the second DRC set do not take into account the effects of the first DRC set. If the first DRC set is not designed to be used without combining it with another DRC set, the *noIndependentUse* flag must be set to 1. In that case, the DRC set can only be used in combination with another DRC set as indicated by the *dependsOnDrcSet* field of the other set that is combined with it.

Usually, each audio channel is assigned to a DRC gain sequence. A collection of channels assigned to the same DRC gain sequence is called "channel group". The assignment of a DRC gain sequence to a channel group is done in the order of first appearance of the sequence index when iterating through all channels (see also Table 14). A DRC gain sequence index *bsSequenceIndex* == 0 indicates that the assigned channel will be passed through by the DRC tool without processing unless otherwise noted. Note that therefore *bsSequenceIndex* is effectively 1-based, wereas the corresponding indices (*sequenceIndex*) for processing are zero-based.

If subsequent channels are assigned the same sequence index, the field *repeatSequenceCount* indicates how many channels will have the same sequence not including the first.

The *drcLocation* field is used in the same way as the *drcLocation* field in the drcCoefficients (see 6.1.2.3). Certain entries of the drcLocation field allow adding drcInstructions information to gain sequences defined elsewhere. Some use cases are discussed in C.10.

The field *limiterPeakTarget* declares the peak target level used by the encoder-side DRC, if applicable. For example, if a limiter is used to generate the DRC gain sequence, it is configured to control the audio sample magnitude to not exceed this peak target level. *limiterPeakTarget* is represented in dBFS and encoded according to Table A.27.

If *limiterPeakTarget* is present, and the only drcSetEffect is "clipping prevention", the gain sequence is to be shifted by the negative sum of *loudnessNormalizationGainDb* and *limiterPeakTarget* if the negative sum is greater than 0. Afterwards, the gain sequence is saturated at the threshold of 0 dB so that only negative gains (dB) occur. With this mechanism it is possible to send gains for clipping prevention in expectation of a high *loudnessNormalizationGainDb*. If *loudnessNormalizationGainDb* is lower than expected, the gains are applied only as far as needed, and the dynamic range can be kept as high as possible.

If *gainScalingPresent* == 1, the gain scaling coefficients must be applied to the channel group. If *gainOffsetPresent* == 1, the gain offset value must be applied to the channel group as shown in Table 16. Similarly, if *duckingScalingPresent* == 1, the scaling factor must be applied to the associated ducking gain sequence for that channel group.

The in-stream drcInstructions syntax is given in Table 46 and Table 48. The syntax for the corresponding block for ISO/IEC 14496-12 is shown in Table 47 and Table 49. The corresponding blocks carry essentially the same information. Values that are identically included in both blocks are coded the same way except for drcLocation. Further information on the coding of drcLocation is defined in 6.1.2.3.

6.1.2.5 loudnessInfo()

A loudnessInfo() block includes loudness and peak information. A downmix identifier and DRC set identifier indicate which configuration the information applies to. Hence, this block can be associated with the audio signal without DRC and without downmix, or with any specific DRC and/or downmix applied. If a DRC with a dependent DRC set is applied, the loudness information describes the output of

the combined DRCs. A loudnessInfo() block can either represent an individual content item or the entire album. Typically, all content items of an album include identical album loudnessInfo() blocks.

If *downmixId* is zero, then loudnessInfo() applies to the base layout. If the *drcSetId* is zero, then loudnessInfo() applies to the audio signal without DRC processing.

The fields *samplePeakLevel* and *truePeakLevel* represent the level of the maximum sample magnitude in dBFS and the true peak in dBTP, respectively, of the associated audio content before or after audio encoding as defined in Reference [4]. The *measurementSystem* field includes standardized systems and others (see Table A.37). System 3 is defined as ITU-R BS.1770-3 with pre-processing. The pre-processing is a high-pass filter that models the typical limited frequency response of portable device loudspeakers. System 4 is defined as "User". It means that the corresponding *methodValue* reflects a (subjective) user preference. System 5 is defined as "Expert/Panel". It means that the corresponding *methodValue* represents a (subjective) expert or panel preference.

The *methodDefinition* field according to Table A.36 specifies how the *methodValue* is derived. The mixing level is compatible with "mixlevel" in ATSC A/52.[1] It indicates the absolute acoustic sound pressure level of an individual channel during the final audio mixing session. The peak mixing level is the acoustic level of a sine wave in a single channel whose peaks reach 100 percent in the PCM representation. The absolute SPL value is typically measured by means of pink noise with an RMS value of −20 or −30 dB with respect to the peak RMS sine wave level. The value of mixing level is not typically used within the DRC tool, but may be used by other parts of the audio reproduction system.

The room type field is compatible with "roomtyp" in ATSC A/52.[1] It indicates the type and calibration of the mixing room used for the final audio mixing session. The value of *roomtyp* is not typically used by the DRC tool, but may be used by other parts of the audio reproduction system.

The loudnessInfoSet() payload contains all loudnessInfo() blocks. The in-stream syntax of loudnessInfoSet() is given in Table 37. For the ISO base media file format the slightly different syntax of "LoudnessBox" is used as defined in ISO/IEC 14496-12.

6.1.3 Derivation of peak and loudness values

The loudnessInfo() blocks provide optional values that describe loudness and peak. Several DRC decoder processes depend on these values, hence, when the loudness information is partially or entirely absent, fallback values are used as shown in Table 5. For peak values, a default value is to be used. Some other values can be drawn from the loudnessInfo() block of the base layout.

Table 5 — Default and fallback values of loudnessInfo

Value	Default	1st fallback: use value from loudnessInfo() of base layout with same DRCsetId	2nd fallback: use value from loudnessInfo() of the base layout without DRC
truePeakLevel	0.0	No	No
samplePeakLevel	0.0	No	No
programLoudness	Undefined	Yes	Yes
anchorLoudness	Undefined	Yes	Yes
loudnessRange	Undefined	No	No
Maximum loudness range	Undefined	No	No
Maximum momentary loudness	Undefined	No	No
Maximum short-term loudness	Undefined	No	No
Short-term loudness	Undefined	No	No
Mixing level	Undefined	Yes	Yes
Room type	Undefined	Yes	Yes

The *signalPeakLevel* of a DRC set is determined as specified in Table 6, where peak related metadata entries are selected dependent on their availability and dependent on the drcSetId, and the requested downmixId. If no explicit peak information is available, *signalPeakLevel* is estimated from downmix coefficients and others. The estimates based on downmix coefficients hold for passive downmixers and might hold for specific active downmixers.

Table 6 — Determination of signalPeakLevel for a specific DRC set

```
getSignalPeakLevelForDrcSet (drcSetId, downmixIdRequested) {
 dmxId = downmixIdRequested;
 if truePeakLevelIsPresent(drcSetId, dmxId) {
  signalPeakLevel = getTruePeakLevel(drcSetId, dmxId);
 } else if samplePeakLevelIsPresent(drcSetId, dmxId) {
  signalPeakLevel = getSamplePeakLevel(drcSetId, dmxId);
 } else if limiterPeakTargetIsPresent(drcSetId, dmxId) {
  signalPeakLevel = getLimiterPeakTarget(drcSetId, dmxId);
 } else if (dmxId != 0) {
  signalPeakLevelTmp = 0.0;
  downmixPeakLevelLinear = 0.0;
  if downmixCoefficientsArePresent(dmxId) {
   for (i=0; i<targetChannelCount; i++) {
    downmixPeakLevelLinearTmp = 0.0;
   for (j=0; j<baseChannelCount; j++) {
      downmixPeakLevelLinearTmp +=
         pow(10.0, getDownmixCoefficient(dmxId, i, j)/20.0);
     }
     if (downmixPeakLevelLinear < downmixPeakLevelLinearTmp) {
      downmixPeakLevelLinear = downmixPeakLevelLinearTmp;
     }
    }
  }
  if truePeakLevelIsPresent(drcSetId, 0) {
   signalPeakLevelTmp = getTruePeakLevel(drcSetId, 0);
  } else if samplePeakLevelIsPresent(drcSetId, 0) {
   signalPeakLevelTmp = getSamplePeakLevel(drcSetId, 0);
  } else if limiterPeakTargetIsPresent(drcSetId, 0) {
   signalPeakLevelTmp = getLimiterPeakTarget(drcSetId, 0);
  }
  signalPeakLevel = signalPeakLevelTmp + 20.0*log10(downmixPeakLevelLinear);
 } else {
  signalPeakLevel = 0.0;    /* worst case estimate */
 }
 return signalPeakLevel
}
```

Table 6 includes functions to check the availability and to retrieve peak-related information from loudnessInfo() and a drcInstructions block which can have the basic or uniDrc format. Table 7 shows pseudo code for some of the functions for the truePeakLevel and limiterPeakTarget. The functions for samplePeakLevel can be implemented by replacing truePeakLevel with samplePeakLevel.

Table 7 — Pseudo code for functions referenced in Table 6

```
truePeakLevelIsPresent(drcSetId, downmixId) {
 if (useAlbumMode == 1) count = loudnessInfoAlbumCount;
 else count = loudnessInfoCount;
 for (i=0; i<count; i++) {
  if (loudnessInfo[i]->drcSetId == drcSetId) &&
    (loudnessInfo[i]->downmixId == downmixId) {
   if (loudnessInfo[i]->truePeakLevelPresent) return TRUE;
  }
 }
 return FALSE;
}
getTruePeakLevel(drcSetId, downmixId) {
 if (useAlbumMode == 1) count = loudnessInfoAlbumCount;
 else count = loudnessInfoCount;
 for (i=0; i<count; i++) {
  if (loudnessInfo[i]->drcSetId == drcSetId) &&
    (loudnessInfo[i]->downmixId == downmixId) {
   if (loudnessInfo[i]->truePeakLevelPresent) {
    return (loudnessInfo[i]->truePeakLevel);
   }
  }
 }
 return error;
}
limiterPeakTargetIsPresent(drcSetId, downmixId) {
 for (i=0; i<drcInstructionsCount; i++) {
  if  (drcInstructions[i]->drcSetId == drcSetId) &&
    ((drcInstructions[i]->downmixId == downmixId) ||
     (drcInstructions[i]->downmixId == 0x7F)) {
   if (drcInstructions[i]->limiterPeakTargetPresent) return TRUE;
  }
 }
 return FALSE;
}
getlimiterPeakTarget(drcSetId, downmixId) {
 for (i=0; i<drcInstructionsCount; i++) {
  if  (drcInstructions[i]->drcSetId == drcSetId) &&
    ((drcInstructions[i]->downmixId == downmixId) ||
     (drcInstructions[i]->downmixId == 0x7F)) {
   if (drcInstructions[i]->limiterPeakTargetPresent)) {
    return (drcInstructions[i]->limiterPeakTarget);
   }
  }
 }
 return error;
}

downmixCoefficientsArePresent(downmixId) {
 for (i=0; i<downmixInstructionsCount; i++) {
  if (downmixInstructions[i]->downmixId == downmixId) {
   if (downmixInstructions[i]->downmixCoefficientsPresent) return TRUE;
  }
 }
 return FALSE;
}
getDownmixCoefficient(downmixId, outChan, inChan) {
 for (i=0; i<downmixInstructionsCount; i++) {
  if (downmixInstructions[i]->downmixId == downmixId) {
   if (downmixInstructions[i]->downmixCoefficientsPresent) {
    return (downmixInstructions[i]->downmixCoefficient[outChan][inChan]);
   }
  }
 }
 return error;
}
```


6.2 Dynamic DRC gain payload

The dynamic gain sequences for all DRCs are received in-stream or via a metadata track using the uniDrcGain() syntax given in Table 34. Each access unit contains gain sequences for the duration of *drcFrameSize* samples that are decoded according to 6.4.5.

The last part of the uniDrcGain() can include future extension payloads. If a uniDrcGainExtType is received that is different from UNIDRCGAINEXT_TERM, the extension payload (otherBit) must be read and discarded.

When gain values are stored in a separate meta-data track under the sample entry code 'unid' as described in 6.1.2.3, each sample is a uniDrcGain() padded with 0 to 7 trailing zero bits to the next byte boundary.

6.3 DRC set selection

6.3.1 Overview

A bitstream can carry multiple DRCs for various purposes. At the decoder, the DRC set that best matches the requirements for the given playback scenario is selected. The selection process is performed in three stages:

1. A pre-selection that discards all DRC sets that are not applicable because they do not match a target channel configuration, don't support the decoder target loudness, or have more clipping than requested (see 6.3.2).
2. A main selection process based on requested DRC set features (see 6.3.3).
3. A final selection that picks a single DRC set if multiple DRC sets are applicable (see 6.3.4).

The most relevant metadata for the selection process is summarized in Table 8. The bit fields of the *drcSetEffect* field are described in detail in Table A.32. All parameters that can be supplied by the host to control loudness normalization and dynamic range compression are summarized in Table A.40 and Table A.41, respectively.

DRC sets with only a "Fade" or "Ducking" effect are automatically selected by the decoder without using the three-stage selection process. DRC sets with other features can be requested by using DRC decoder settings as described below.

The pool of DRC sets that is subject to the three-stage selection process comprises not only the DRC sets defined in the bitstream (except for "Fade" and "Ducking") but also virtual DRC sets generated in the DRC tool. The virtual DRC sets are placeholders for the cases where no compression is applied to the audio signal, hence their drcSetEffect bits are zero and they correspond to the DRC effect request "None".

Conceptually it is possible to integrate other DRC sets specified by the drcInstructionsBasic() and drcCoefficientsBasic() syntax in the selection process. However, this is out of scope and not part of this specification.

Table 8 — Most relevant metadata for DRC selection at the decoder

Field	Type
drcInstructionsUniDrc->downmixId	Identifier
drcInstructionsUniDrc->limiterPeakTarget	Value
drcInstructionsUniDrc->drcSetEffect	Bit field
drcInstructionsUniDrc->noIndependentUse	Flag
drcInstructionsUniDrc->drcSetTargetLoudnessValueUpper/-Lower	Value
loudnessInfo->truePeakLevel	Value
loudnessInfo->samplePeakLevel	Value
loudnessInfo->program loudness	Value
loudnessInfo->anchor loudness	Value
loudnessInfo->maximum short-term loudness	Value
loudnessInfo->maximum momentary loudness	Value
loudnessInfo->maximum loudness range	Value
drcCoefficientsUniDrc->drcCharacteristic[sequence][band]	Index
drcCoefficientsUniDrc->bandCount[sequence]	Value

6.3.2 Pre-selection based on Signal Properties and Decoder Configuration

6.3.2.1 Overview

The pre-selection selects all DRC sets that fulfill all requirements listed in Table 9. All available DRC sets are analysed in the given order of steps. If no DRC set is selected, no DRC can be applied except for fading or ducking.

Table 9 — Requirements for DRC pre-selection

#	Requirement	Applicability	Comment
1	DownmixId of DRC set matches the requested downmixId.	If a downmixId is requested	See 6.3.2.2
2	Output channel layout of DRC set matches the requested layout.	If a target channel layout is requested	See 6.3.2.2
3	Channel count of DRC set matches the requested channel count.	If a target channel count is requested	See 6.3.2.2
4	The DRC set is not a "Fade-" or "Ducking-" only DRC set.	Always	DRC sets with "Fade" or "Ducking" effect are selected automatically. They are not subject to this selection process.
5	The number of DRC bands is supported.	Always	DRC sets that exceed the number of supported DRC bands are discarded. For time-domain DRC, the maximum is four bands.
6	Independent use of DRC set is permitted.	If the DRC set is not used in combination with another DRC set.	DRC sets with a noIndependentUse flag value of 1 can only be used in combination with a second DRC set.
7	The range of the target loudness specified for a DRC set has to include the requested decoder target loudness.	If drcSetTargetLoudnessPresent = = 1 and no explicit peak information is available for that DRC set.	See 6.3.2.2.2
8	Clipping is minimized.	Except for DRC sets which were already selected in pre-selection step #7.	See 6.3.2.2.3

6.3.2.2 Detailed description of pre-selection steps

6.3.2.2.1 Pre-selection based on downmixId, channel layout, or channel count (#1,2,3)

Requests for downmixIds, target channel layout, or target channel count are supported. Only one of these requests will be used based on the following priority:

1. downmixId(s)
2. target channel layout
3. target channel count

If no downmixId is requested, the request(s) will be mapped to one or more matching downmixIds that are used in the pre-selection process as defined in the following.

If a target channel layout is requested but a downmixId is not requested, the channel layout is mapped to downmixIds with a matching layout. If only a target channel count is requested, it is mapped to downmixIds with a matching target channel count. If no matching downmixIds can be found, no DRC set can be applied except for fading or ducking.

If no request is present, a downmixId of 0x0 (base layout) will be used for the pre-selection.

A DRC set is selected if the requested downmixId (or the downmixId, which was generated from a different kind of request) matches one of the defined downmixIds of the DRC set. Note that DRC sets, which define a downmixId of 0x0 or 0x7F automatically pass the pre-selection #1,2,3.

A downmix ID list contains all downmixIds to be used for pre-selection as described in the previous steps. The pre-selection for these requests is done by selecting all DRC sets that have one of the downmixIds of the list or a downmixId of 0x0 or 0x7F. Note that a requested downmixId of 0x7F is meaningless and therefore not permitted.

6.3.2.2.2 Pre-selection based on drcSetTargetLoudness (#7)

This pre-selection step addresses only DRC sets for which drcSetTargetLoudnessPresent equals one and for which no explicit peak information is available. From the DRC sets which match this criterion, only those are selected whose range defined by drcTargetLoudnessValueUpper/-Lower includes the requested decoder target loudness (targetLoudness). Note that pre-selection step #8 is omitted for DRC sets selected in this step. Explicit peak information for a specific DRC set is available if at least one of the following results is TRUE:

— truePeakLevelIsPresent(drcSetId, downmixIdRequested)

— samplePeakLevelIsPresent(drcSetId, downmixIdRequested)

— limiterPeakTargetIsPresent(drcSetId, downmixIdOfDrcSet)

6.3.2.2.3 Pre-selection based on output peak level (#8)

This pre-selection step addresses the problem that can arise if due to loudness normalization or downmixing the maximum peak value exceeds full scale. Two mechanisms can be used to avoid clipping, a DRC set that reduces or eliminates clipping and/or a peak limiter. DRC sets resulting in too high peak values at the output possibly need to be discarded. The peak value of the output signal is computed as

$$outputPeakLevel = signalPeakLevel + loudnessNormalizationGainDb$$

where

signalPeakLevel is the peak value of the signal in dBFS according to Table 6;

loudnessNormalizationGainDb is the scaling factor for the loudness normalization in dB.

NOTE The *loudnessNormalizationGainDb* can be specific for each DRC set if individual loudnessInfo() blocks are present for each DRC set.

All DRC sets are discarded whose *outputPeakLevel* exceeds *outputPeakLevelMax.* The recommended value of *outputPeakLevelMax* is 0 dBFS when no peak limiter is applied after the DRC tool. If a peak limiter is subsequently applied, it is recommended to set *outputPeakLevelMax* to the maximum peak value the peak limiter can handle without introducing severe and audible distortions. By default, the DRC tool assumes that no peak limiter will be applied. The default value for *outputPeakLevelMax* is 0 dBFS. If a peak limiter is applied (peakLimiterPresent==1), the default value for *outputPeakLevelMax* is 6 dBFS.

If no DRC set was selected so far (including the pre-selection in step #7), select the DRC sets for which drcSetTargetLoudnessPresent equals one and whose range defined by drcTargetLoudnessValueUpper/-Lower includes the requested decoder target loudness. If after that no DRC set was selected, the requirement of clipping prevention needs to be relaxed until at least one DRC set is selected. Therefore, the DRC sets with the lowest *outputPeakLevel* are selected. In addition, the DRC sets are selected whose *outputPeakLevel* does not exceed the lowest *outputPeakLevel* by more than 1.0 dB.

At this point, the output signal can have audible distortions, either due to clipping or because a peak limiter is applied at too high levels. This is mitigated by lowering the output level resulting in an intentional deviation from the target loudness. The parameter *loudnessDeviationMax* (which is 63 dB by default) limits the gain reduction that is applied to reduce the distortions (e.g. *loudnessDeviationMax* of 3 dB permits a gain adjustment of –3 dB).

6.3.3 Selection based on requests

6.3.3.1 Overview

In the following, the selected DRC sets are matched with requested features that can describe certain aspects of a DRC set. The features are organized in an ordered list that is given to the DRC decoder. One or more features can be requested. A feature can be requested multiple times. The decoder will search the available DRC sets to find the best match according to the following rules. Table 10 lists DRC features that are supported for the DRC search.

Table 10 — Requestable features of DRC sets

Index	DRC feature	Comment
0	"Effect type"	Specifies the type of DRC effect
1	"Dynamic range"	Specifies the dynamic range, see Table 12
2	"DRC characteristic"	Specifies the DRC characteristic at the encoder

The decoder works through the feature list item-by-item starting from the first. Conceptually, it takes for each item the selected DRC sets and selects only those that match the requested feature. The following specifies the selection rules for each requested feature in detail. If no features are requested, the selection process is performed as if the effect type feature with effect type "None" has been requested as described in the following section. If that request does not succeed, another feature request attempt is issued using the "General" effect type with the fallback effect types "Night", "Noisy", "Limited", "LowLevel".

6.3.3.2 Sub-selection by requesting an effect type feature

The effect type as specified by the field *drcSetEffect* in Table A.32 describes the result of applying the DRC set. For DRC sets that carry a non-zero entry in the dependsOnDrcSet field, the valid effect types for the combination of both sets are obtained from the effect types of both sets (the depending DRC set and the one with a corresponding dependsOnDRCSet value).

Table 11 lists all effect types that can be requested. One effect type can be requested at a time. Requests for effect types can be repeated multiple times with different effect types. From the DRC sets selected so far, all DRC sets that match the requested effect type are selected.

An effect type request is ignored if none of the selected DRC sets match. Consequently, the sub-selection process proceeds with the results of the previous request, if applicable; otherwise the results of the pre-selection are used.

The list of requested effect types contains the desired effect types. Additionally, fallback effect types can follow the desired effect types. The fallback effect types are specified to allow selecting a DRC set even if DRC sets with the desired effect types are not available.

After a request sequence of desired effect types, the sub-selection ends if one or more DRC sets are selected and any fallback requests are ignored. Otherwise, the sub-selection continues with the fallback effects. In that case, the sub-selection ends if one or more matching DRC sets have been selected during the processing of fallback requests and the remaining fallback requests are ignored.

The requests can be specified as a list of requested effect types together with an index of the first fallback request in the list. If for example only one single desired effect type is specified, the fallback requests start at position 2. More detailed explanations are given in Annex E.

Table 11 — Requestable DRC effect types and short names for reference

Index	drcSetEffect	Short name	Description
0	None	"None"	A DRC set that has no dynamic compression, for instance "Clipping", or applying no DRC set for compression.
1	Late night	"Night"	For quiet environment, listening at low level, avoiding to disturb others.
2	Noisy environment	"Noisy"	Optimized to get the best experience in noisy environments, for instance by amplifying soft sections.
3	Limited playback range	"Limited"	Reduced dynamic range to improve quality on playback devices with limited dynamic range.
4	Low playback level	"LowLevel"	Listening at a low playback level.
5	Dialog enhancement	"Dialog"	The main effect is a more prominent dialogue within the content.
6	General compression	"General"	A DRC effect that reduces the dynamic range and is applicable to multiple playback scenarios.
7	Dynamic expansion	"Expand"	Dynamics enhancement.
8	Artistic effect	"Artistic"	To create an artistic sound effect.

6.3.3.3 Sub-selection by requesting a "Dynamic Range" value

The decoder can receive a request for a single Dynamic Range measurement value or value range. The measurement is based on one of those provided in Table 12. If a value is specified, all DRC sets are selected with the closest matching value of the feature. If a range is specified, all DRC sets are selected whose feature value is within that range.

Each of the measurement values for "short-term loudness", "momentary loudness", and "top of loudness range" can be given for multiple measurement systems. Permitted measurement systems include EBU R128 and BS.1771-1. The first system shall be selected when searching the available values in order of increasing measurement system index (see Table A.37) including all reserved indices for future updates, i.e. if a reserved index is found, it shall be interpreted as a valid measurement system.

The program loudness values used in Table 12 shall be based on the following measurement systems (see Table A.37):

1. RMS_C
2. RMS_B
3. RMS_A
4. BS.1770-3

If multiple values are given, choose the first available using the order of the list.

Table 12 — Requestable Dynamic Range Measurement Values

Index	Label	Dynamic range measurement	Comment
0	StA	"Short-term loudness peak to average"	Max. short-term loudness minus program loudness (see 6.6.2)
1	MtA	"Momentary loudness peak to average"	Max. momentary loudness minus program loudness (see 6.6.2)
2	TtA	"Top of loudness range to average"	Top of loudness range minus program loudness (see 6.6.2)

6.3.3.4 Sub-selection by requesting a “DRC characteristic”

From the selected DRC sets all sets are selected with the closest DRC characteristic index according to a matching order. If different DRC characteristics are used within the same DRC set, only the DRC characteristic indices are taken that are in the matching order list. The remaining ones are ignored. If no DRC set is found for the matching order of the requested DRC characteristic in Table 13, this selection step is ignored.

Table 13 — Matching order for DRC characteristic

		Requested DRC characteristic											
		1	2	3	4	5	6	7	8	9	10	11	c; c > 11
Matching order	first	1	2	3	4	5	6	7	8	9	10	11	c
	second	2	3	4	5	6	5	9	10	7	8	10	-
	third	-	1	2	3	4	-	-	-	-	-	9	-

6.3.4 Final selection

This clause uses the term “multiple DRC sets” for DRC sets that are independent of each other and do not include ducking or fading effects.

If there are still multiple DRC sets selected, select the ones that result in an output peak value of 0.0 or less. If no such DRC set is found, from the ones selected, select the DRC sets that exceed the threshold by the minimum amount. Note that DRC sets selected in pre-selection step #7 are assumed to have an output peak value of 0.0 or less.

If there are still multiple DRC sets selected, select the ones that exactly match the requested downmixId.

If there are still multiple DRC sets selected, select the ones with the minimum number of effect types where the effect bit for “General” is ignored. Please note that the reason for ignoring the effect bit “General” is that otherwise DRC sets without “General” would be unjustifiably preferred.

If there are still multiple DRC sets selected, discard any DRC set that was selected in the pre-selection step dealing with drcSetTargetLoudness. If all selected DRC sets are discarded in this step, select the DRC set with the smallest drcSetTargetLoudnessValueUpper instead.

If there are still multiple DRC sets selected, select the DRC sets with the largest peak value. Note that the output peak value also depends on the modifications of the gains as described in 6.4.6.

If there are still multiple DRC sets selected, choose the one with the largest DrcSetId.

6.3.5 Applying multiple DRC sets

In the following cases multiple DRC sets are applied simultaneously. First, if the DRC set selected in 6.3.4 carries a non-zero entry in the dependsOnDrcSet field, the depending DRC set is applied together with the selected one. Second, if a DRC set with “Fade” or “Ducking” effect was automatically selected, it is applied simultaneously with the DRC set selected in 6.3.4. Thus, if the DRC set selected in 6.3.4 has a non-zero dependsOnDrcSet value, a total of three DRC sets are applied, which is the maximum number permitted. If all three DRC sets are applied to the same layout (downmixId), the DRC set with “Fade” or “Ducking” effect shall be applied first, the DRC set referenced in the dependsOnDrcSet field shall be applied thereafter, and the DRC set selected in 6.3.4 shall be applied last. If only two DRC sets are applied to the same layout (downmixId), the same order applies. Note that if a DRC set with “Fade” effect and another DRC set with “Ducking” effect were both automatically selected, the DRC set with “Fade” effect is ignored.

6.3.6 Album mode

When the playback system is in album mode, i.e. successive songs of an album are played back, the DRC selection should be applied first using all DRCs in the loudnessInfo blocks for albums. These blocks may contain drcSetIds that point to matching DRCs. Only if there is no match with any DRC in a loudnessInfo block for albums, the selection should proceed to apply the logic described in 6.3.2 to all available DRCs.

In album mode, any "Fade" DRC is not applied. If not in album mode, if an applicable "Fade" DRC exists, it must be applied. The "Fade" DRC can be applied simultaneously with any other DRC except "Ducking".

6.3.7 Ducking

The base layout and each specific downmix with a unique downmix ID can have a maximum of one DRC set with a ducking effect. During configuration, the decoder scans all available DRC sets for the active downmix to identify the applicable DRC set for ducking if present. If ducking DRC sets are both defined for the base layout and the active downmix, select the one that exactly matches the active downmix. If a ducking DRC set is identified and the associated overlaid audio signal is active, the ducking gain sequence is automatically applied to all channels except those that are members of the channel group associated with the ducking DRC set (drcSetEffect=="Duck other") or alternatively to all channels that are members of the channel group associated with the ducking DRC set (drcSetEffect=="Duck self"). The overlaid audio is defined to be active if at least one nonzero downmix coefficient is applied to it.

Ducking is always applied before any downmix, i.e to the base layout. Hence, the DRC channel groups for the ducking process refer to the base layout. The downmixId of the corresponding drcInstructionsUniDrc() indicates how to generate the downmix after the ducking was applied.

Note that a ducking DRC set with downmix ID 0x0 (baseLayout) is automatically applied independent of the requested downmix ID. It is therefore recommended to define ducking DRC sets with downmix ID 0x0 only for specific use cases, where the ducking DRC set should be always applied when DRC processing is enabled.

6.3.8 Precedence

If a bitstream contains the described DRC metadata and other DRC metadata such as MPEG light or heavy compression, the described metadata will take precedence unless the decoder is instructed to apply the other DRC metadata.

6.4 Time domain DRC application

6.4.1 Overview

The DRC gain can be applied to the time-domain audio signal or to the sub-band signals of the audio decoder. The following text first includes a full description of the time-domain DRC. Subsequently, the necessary modifications of the time-domain DRC are described to achieve DRC in sub-bands.

Since the encoder provides only sparsely sampled gain values, the decoder applies interpolation to achieve a smooth gain transition between the samples. The sample rate of the interpolated gain is the audio sample rate. Each gain sequence can be configured to use either linear interpolation or spline interpolation. For linear interpolation, the interpolated values of one segment between two subsequent gain samples (nodes) are derived from the two gain samples at both ends of the segment. For spline interpolation, they are additionally derived from their slope (derivative). Hence, when transitioning from one segment to the next, the first derivative is continuous as both segments have the same slope at the transition point.

For applications of the DRC tool in tandem with an audio codec, the following parameters are provided to adjust the DRC frame size and time resolution so that codec and DRC processing can be done most efficiently in terms of complexity and delay. The parameters are:

— DRC frame size in units of the audio sample interval

— *deltaTmin* in units of the audio sample interval

— delay mode.

These parameters have default values, but an audio codec specification may override the defaults.

6.4.2 Framing

The DRC gain information is organized in DRC frames. Each DRC frame contains DRC data to generate the DRC gain for the duration of a DRC frame. The DRC frame duration is constant for a given audio item and it is a multiple of the audio sample interval. DRC frames do not overlap.

In practice, whenever suitable, the DRC frame size M_{DRC} is recommended to correspond to the same duration as the codec's frame size to minimize delay and complexity. This is the default setting for frame sizes of 1 ms and more. For audio formats with a frame size below 1 ms, as is typically the case for PCM, the default DRC frame size is 32 *deltaTmin* as defined in Formula (1). The default frame size is used, unless the frame size is specified by *drcFrameSize* in the bitstream.

6.4.3 Time resolution

The DRC tool uses a uniform time grid to generate a sparse representation of the DRC gain. The spacing of this grid defines the highest available time resolution *deltaTmin*. The unit of *deltaTmin* is one sample interval at the audio sample rate. For efficiency, *deltaTmin* is chosen to be an integer multiple of the audio sample interval with a corresponding duration between [0.5..1.0] ms. Preferably, *deltaTmin* is an integer power of 2, so that sample rates can be efficiently converted between audio and DRC. The default values are computed based on the following formulas:

$$f_s 0.0005 \text{ s} < deltaTmin \leq f_s 0.001 \text{ s} \quad (1)$$

and

$$deltaTmin = 2^M \quad (2)$$

with the audio sample rate f_s in Hz and the non-negative integer exponent M. Audio sample rates smaller than 1 kHz are not permitted.

Alternatively, the value of deltaTmin can be transmitted in the DRC bitstream field *timeDeltaMin*. If it is present, it overrides the default value.

6.4.4 Time alignment

The time alignment specifies the temporal location of each gain sample within a block of *deltaTmin* audio sample intervals. The alignment is conveyed by the *timeAlignment* field in drcCoefficientsUniDrc(). A value of 0 indicates that the gain sample is located at the last audio sample interval of the block, i.e. the sample index at the audio rate is *deltaTmin*-1 assuming that index 0 is the first audio sample interval in the block. A value of 1 indicates that the gain value is located in the centre of the block, i.e. at the sample index of floor((*deltaTmin*-1)/2).

In low-delay mode, a *timeAlignment* value of 1 is not supported. Hence, *timeAlignment* is set to 0 in low-delay mode. The *timeAlignment* value for the gain sequences of all channelGroups within a DRC set shall be identical.

6.4.5 Decoding

The DRC gains are derived from linear interpolation or spline interpolation, which is determined by the node coordinates and, in case of spline interpolation, by their associated slopes. The decoding process of the node coordinates consists of the following sequence of tasks:

1) Gather the DRC configuration information (once);

2) Select a DRC set (once);

3) Parse the DRC gain bitstream (per DRC frame);

4) Apply the code tables including Huffman decoding to decode the quantized values (per DRC frame);

5) Undo the differential encoding (per DRC frame).

The basic in-stream DRC configuration information is fully specified in this document including some parts of ISO base media file format that are not included in ISO/IEC 14496-12. The configuration of the core DRC gain decoding is essentially the same for both cases. The following configuration parameters are most relevant for decoding:

— The number of gain sequences: *nDrcGainSequences*;

— The profile for DRC gain coding: *drcGainCodingProfile*;

— The type of DRC gain interpolation: *gainInterpolationType*;

— The assignment of a gain sequence to each channel. Channels using the same sequence are referred to as channel groups. The total number of groups is *nDrcChannelGroups* (See Table 14);

— The number of DRC bands in a group: *nDrcBands*.

Given these parameters, the part of the DRC bitstream containing the DRC gains (uniDrcGain()) can be parsed and decoded using the algorithms of Table 15 up to Table 19 and the codes of Table A.1 and the following ones based on *gainCodingProfile*.

In the ISO base media file format, the audio content may be carried in multiple tracks where a base track contains the DRC metadata for all tracks. The additional tracks are referenced by the base track using a track reference of type ‘adda’ (additional audio). The channels are all the channels in the base track, plus all the channels in track(s) referenced, in the order of the references. The channel groups apply to all those channels (even if they are channels in a track that is disabled or not currently being played).

Table 14 — Derivation of drcChannelGroups from sequenceIndexes

```
uniqueIndex = {-10, -10, ...};
k=0;
if ((drcSetEffect & (3<<10)) != 0) { /* Ducking */
 uniqueScaling = {-10, -10, ...};
 for (i=0; i<channelCount; i++) {
  match = FALSE;
  idx = sequenceIndex[i];
  factor = duckingScaling[i];
  if (idx<0) channelGroupForChannel[i] = -1;  // means no DRC for this channel
  else {
   for (n=0; n<k; n++) {
    if ((uniqueIndex[n] == idx) && (uniqueScaling[n] == factor)) {
     match = TRUE;
     channelGroupForChannel[i] = n;
     break;
    }
   }
   if (match == FALSE) {
    uniqueIndex[k] = idx;
    uniqueScaling[k] = factor;
    channelGroupForChannel[i] = k;
    k++;
   }
  }
 }
}
else {
 for (i=0; i<channelCount; i++) {
  match = FALSE;
  idx = sequenceIndex[i];
  if (idx<0) channelGroupForChannel[i] = -1;
  else {
   for (n=0; n<k; n++) {
    if (uniqueIndex[n] == idx) {
     match = TRUE;
     channelGroupForChannel[i] = n;
     break;
    }
   }
   if (match == FALSE) {
    uniqueIndex[k] = idx;
    channelGroupForChannel[i] = k;
    k++;
   }
  }
 }
}
nDrcChannelGroups = k;
```


The application of codes is expressed in Table 15 by the pseudo-functions decodeInitialGain(), decodeDeltaGain(), decodeTimeDelta(), and decodeSlope(). Differentially encoded values are then converted into absolute values according to Table 15. The decoded result is represented by the gain values *gDRC*[*g*][b][*k*], the time values *tDRC*[*g*][b][*k*], and the slope values *sDRC*[*g*][b][*k*] where *g* is the channel group index, *b* is the band index, and *k* is the node index. For linear interpolation, the slope values are set to 0. They will be neglected during interpolation. Time values are integer numbers relative to the beginning of the DRC frame in units of *deltaTmin*. The audio sample that coincides with the beginning of the DRC frame has a time value of *tDRC* = 0.

For *gainCodingProfile* == 3, no dynamic gain sequence is transmitted. Hence, the decoded values are constant. This is useful for generating a constant DRC gain based on the *gainOffset* value.

Table 15 — Decoding of DRC gain sample coordinates and slopes in the dB domain

```
if (timeAlignment==0) {
 timeOffset = -1;
}
else {
 timeOffset = - deltaTmin + floor((deltaTmin-1)/2);
}
for(g=0; g<nDrcChannelGroups; g++) {
 if (gainCodingProfile[g]==3) {
  nDrcBands[g] = 1;
  nNodes[g][b] = 1;
  gDRC[g][b][0] = 0.0;
  tDRC[g][b][0] = drcFrameSize + timeOffset;
  sDRC[g][b][0] = 0.0;
 }
 else {
  for(b=0; b<nDrcBands[g]; b++) {
   gDRC[g][b][0] = decodeInitialGain(gainInitialCode[g][b]);
   if (drcGainCodingMode[g][b] == 0) {
    /* "simple" mode */
    tDRC[g][b][0] = drcFrameSize + timeOffset;
    sDRC[g][b][0] = 0.0;
   }
   else {
    for (k=1; k<nNodes[g][b]; k++) {
     gDRC[g][b][k] = gDRC[g][b][k-1]+decodeDeltaGain(gainDeltaCode[g][b][k-1]);
    }
    tmp_timeOffset = timeOffset;
    if (frameEndFlag[g][b] == 1) {
     nodeResFlag = 0;
     for (k=0; k<nNodes[g][b]-1; k++) {
      tDRC_Buf = tmp_timeOffset +
            deltaTmin * decodeTimeDelta(timeDeltaCode[g][b][k]);
      if (tDRC_Buf > drcFrameSize + timeOffset) {/* nodes from node reservoir */
       if (nodeResFlag == 0) {
        tDRC[g][b][k] = drcFrameSize + timeOffset;
        nodeResFlag = 1;
       }
       tDRC[g][b][k+1] = tDRC_Buf;
      }
      else {
       tDRC[g][b][k] = tDRC_Buf;
      }
      tmp_timeOffset = tDRC_Buf;
     }
     if (nodeResFlag == 0) {
      tDRC[g][b][k] = drcFrameSize + timeOffset;
     }
    }
    else {
     for (k=0; k<nNodes[g][b]; k++) {
      tDRC[g][b][k] = tmp_timeOffset +
            deltaTmin * decodeTimeDelta(timeDeltaCode[g][b][k]);
      tmp_timeOffset = tDrc[g][b][k];
     }
    }
    for (k=0; k<nNodes[g][b]; k++) {
     if (gainInterpolationType == 0) {
      sDRC[g][b][k] = decodeSlope (slopeCode[g][b][k]);
     }
     else {
      sDrc[g][b][k] = 0.0; /* slope will be neglected */
     }
    }
   }
  }
 }
}
```

6.4.6 Gain modifications and interpolation

Before the gain can be applied to the audio signal, it must be converted to the linear domain and gain values between gain samples must be interpolated. To achieve lower complexity, the dB to linear conversion is done before the interpolation. Hence, the interpolation process is entirely done in the linear domain. Both, the gain modification and conversion to the linear domain are done using the pseudo code of Table 16. The input variables are the gain sample and slope in the dB domain. The output consists of the gain sample and slope in the linear domain.

As described in E.2.4 and E.4, there are several ways to adapt the DRC characteristics in the DRC tool decoder. These adjustments are applied to the decoded gain samples in the dB domain. The function toLinear() includes all necessary steps to generate a linear gain sample from the logarithmic value in dB (see Table 16). It contains an optional mapping function mapGain() (see Table 17) that supports modifications of the DRC gain values with the purpose of achieving a different compression characteristic than the one used in the encoder. The mapping is controlled by the index *drcCharacteristicTarget* that will select one of the custom decoder DRC characteristics if it is larger than 0. Otherwise, the encoder characteristic will not be replaced. A modified characteristic can be generated based on the encoder compression characteristic that is conveyed in the DRC configuration. Moreover, a compression and boost factor is supported to scale negative and positive gains, respectively. These factors have a value of 1.0, unless values in the range [0,1] are supplied by the user. Similarly, an encoder-controlled scaling is applied when *gainScalingPresent* == 1 using the scale factors *attenuationScaling* and *amplificationScaling*. When ducking is active, the ducking gains in dB are scaled by the factor *duckingScaling,* if present. Note that the *duckingScaling* factors are conveyed in the drcInstructionsUniDrc() payload for the channel they are applied to, which is in contrast to the *bsSequenceIndex* channel assignment for the "Duck other" effect. User supplied compression and boost factors shall be applied to all DRC sets except DRC sets with ducking, fading, or clipping effect.

If the only drcSetEffect is "clipping prevention", the clipping prevention gains are shifted depending on the target loudness to provide just enough signal attenuation to avoid clipping. The scaling factor is derived based on the *limiterPeakTarget*. See 6.6 for the calculation of *loudnessNormalizationGainDb*. Note that the shifting mechanism won't be enabled if there are additionally defined effect bits apart from "clipping prevention".

Table 16 — Conversion of a DRC gain sample and associated slope from dB to linear domain

```
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;
  if (((drcSetEffect & EFFECT_BITS_DUCKING) == 0) &&
     (drcSetEffect != EFFECT_BIT_FADE) &&
     (drcSetEffect != EFFECT_BIT_CLIPPING)) {
   if ((drcCharacteristicTarget > 0) && (gainDb != 0.0)){
      gainRatio = mapGain(gainDb) / gainDb;
   }
   if (gainDb < 0.0) {
    gainRatio *= compress;
   }
   else {
      gainRatio *= boost;
   }
  }
  if (gainScalingPresent) {
   if (gainDb < 0.0) {
    gainRatio *= attenuationScaling;
   }
   else {
      gainRatio *= amplificationScaling;
   }
  }
  if (duckingScalingPresent && (drcSetEffect & EFFECT_BITS_DUCKING)) {
   gainRatio *= duckingScaling;
  }
  gainLin = pow(2.0, gainRatio * gainDb / 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);
}
```

Table 17 — DRC gain mapping according to a target DRC characteristic

```
mapGain(gainQuant) {
   inLevel = inverseCompressorIO(gainQuant);
   outgain = targetCompressorIO(inLevel);
   return outgain;
}
```

The gain interpolation is implemented by the pseudo code in Table 18. Dependent on the configuration variable gainInterpolationType, either spline interpolation or linear interpolation is performed. The input variables are:

— the time difference between the two gain samples in units of the target sample rate interval *tGainStep*

— a pair of subsequent gain samples *gain0* and *gain1* in dB

— a pair of corresponding slope steepness values *slope0* and *slope1* in the dB domain. These are neglected for linear interpolation.

This function uses toLinear() to convert the variables to the linear domain. The result is a smooth sequence of gain values at the target sample rate located between the pair of gain samples. The target sample rate is the sample rate of the compressed audio signal.

Table 18 — Interpolation of the DRC gain for one spline or linear segment

```
interpolateDrcGain(tGainStep, gain0, gain1, slope0, slope1)
{
  int n;
  float k1, k2, a, b, c, d;
  float slopeLeft;
  float slopeRight;
  float gainLeft;
  float gainRight;
  (gainLeft, slopeLeft) = toLinear (gain0, slope0);
  (gainRight, slopeRight) = toLinear (gain1, slope1);

  if (gainInterpolationType == 0) {
    slopeLeft = slopeLeft / deltaTmin;
    slopeRight = slopeRight / deltaTmin;
    bool useCubicInterpolation = TRUE;
    int tConnect;
    float tConnectFloat;
    if (abs(slopeLeft) > abs(slopeRight)) {
      tConnectFloat = 2.0 * (gainRight - gainLeft - slopeRight * tGainStep) / (slopeLeft - slopeRight);
      tConnect = (floor) (0.5 + tConnectFloat);
      if ((tConnect >= 0) && (tConnect < tGainStep)) {
        useCubicInterpolation = FALSE;
        result[0] = gainLeft;
        result[tGainStep] = gainRight;
        a = (slopeRight - slopeLeft) / (tConnectFloat + tConnectFloat);
        b = slopeLeft;
        c = gainLeft;
```

Table 18 *(continued)*

```
        for (n=1; n<tConnect; n++) {
          float t = (float) n;
          result[n] = (a * t + b ) * t + c;
          result[n] = max (0.0, result[n]);
        }
        a = slopeRight;
        b = gainRight;
        for (   ; n<tGainStep; n++) {
          float t = (float) (n - tGainStep);
          result[n] = a * t + b;
        }
      }
    }
    else if (abs(slopeLeft) < abs(slopeRight)) {
      tConnectFloat = 2.0 * (gainLeft - gainRight + slopeLeft * tGainStep) / (slopeLeft - slopeRight);
      tConnectFloat = tGainStep - tConnectFloat;
      tConnect = (floor) (0.5 + tConnectFloat);
      if ((tConnect >= 0) && (tConnect < tGainStep)) {
        useCubicInterpolation = FALSE;
        result[0] = gainLeft;
        result[tGainStep] = gainRight;
        a = slopeLeft;
        b = gainLeft;
        for (n=1; n<tConnect; n++) {
          float t = (float) n;
          result[n] = a * t + b;
        }
        a = (slopeRight - slopeLeft) / (2.0 * (tGainStep - tConnectFloat));
        b = - slopeRight;
        c = gainRight;
        for (   ; n<tGainStep; n++) {
          float t = (float) (tGainStep-n);
          result[n] = (a * t + b ) * t + c;
          result[n] = max (0.0, result[n]);
        }
      }
    }

    if (useCubicInterpolation == TRUE) {
      float tGainStepInv = 1.0 / (float)tGainStep;
      float tGainStepInv2 = tGainStepInv * tGainStepInv;
      k1 = (gainRight - gainLeft) * tGainStepInv2;
      k2 = slopeRight + slopeLeft;
      a = tGainStepInv * (tGainStepInv * k2 - 2.0 * k1);
      b = 3.0 * k1 - tGainStepInv * (k2 + slopeLeft);
      c = slopeLeft;
      d = gainLeft;
      result[0] = gainLeft;
      result[tGainStep] = gainRight;
      for (n=1; n<tGainStep; n++) {
        float t = (float) n;
        result[n] = (((a * t + b ) * t + c ) * t ) + d;
        result[n] = max (0.0, result[n]);
      }
    }
    else {
      a = (gainRight – gainLeft) / tGainStep;
      b = gainLeft;
      result[0] = gainLeft;
      result[tGainStep] = gainRight;
      for (n=1; n<tGainStep; n++) {
        float t = (float)n;
        result[n] = a * t + b;
      }
    }
    return result;
}
```

6.4.7 Spline interpolation

Interpolation of the DRC gain in the decoder is based on pairs of gain samples. For spline interpolation, in addition to the node coordinates (time and value in dB), also the slope information is given for each pair. The decoder will choose one of three available types of interpolation as illustrated in Figure 3. In most cases cubic interpolation is chosen. However, under certain conditions, a hybrid interpolation combining linear and quadratic interpolation is applied instead. For the hybrid interpolation, a node is inserted between the two given nodes (shown as a square). At one side of that node, linear interpolation is applied and quadratic interpolation is applied at the other. The method is fully specified in 6.4.6.

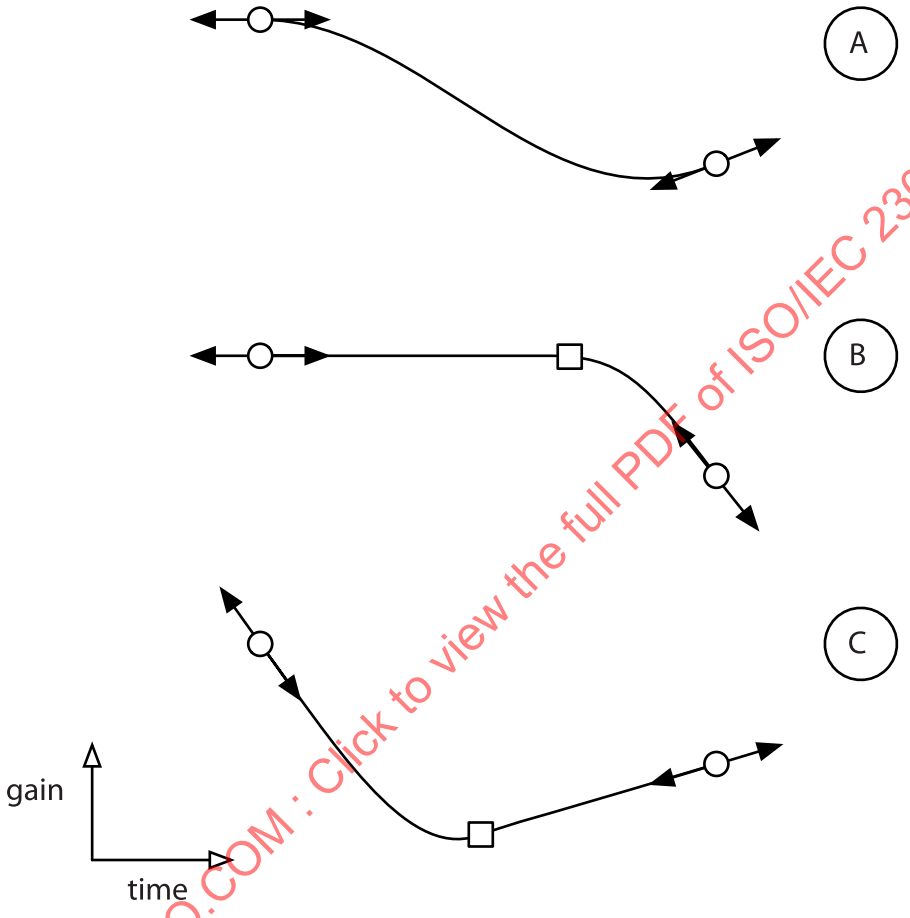

Figure 3 — Spline segment types: cubic interpolation (A). Hybrid using linear and quadratic interpolation (B, C). A square indicates the connection point of the linear and quadratic sub-segment

6.4.8 Look-ahead in decoder

The DRC tool decoder can be operated in one of two delay modes. The low-delay mode immediately applies the decoded DRC gain while the default mode applies the DRC gain with a delay of one DRC frame. The low-delay mode requires that the flag *fullFrame* is 1, which signals that a gain value sample is located at the end of each DRC frame. For the default mode, the flag *fullFrame* can also be 0. Then it supports gain sample interpolation from any position of the current DRC frame to any position of the next DRC frame.

Figure 4 illustrates the two delay modes. The upper diagram shows that each DRC frame has a spline node at the end of the frame, so that the entire DRC gain curve for that frame can immediately be generated by interpolation. The lower diagram shows that the interpolated gain curve is applied with a delay of one DRC frame, since the interpolation for frame *n*-1 can only be completed after the first node of frame *n* is received.

For common perceptual codecs the default delay mode does not require additional decoder delay. The delay is already required due to the overlap-add operation. The decoder appends audio samples with zeros at the end to reconstruct the last frame. If the DRC gains for the last frame don't cover the entire duration, the last gain shall be repeated until the end of the frame. If DRC gains are required for the first frame in default delay mode, they should be generated based on a node at the beginning of the first frame with 0 dB gain and, when in spline interpolation mode, with a slope of 0.

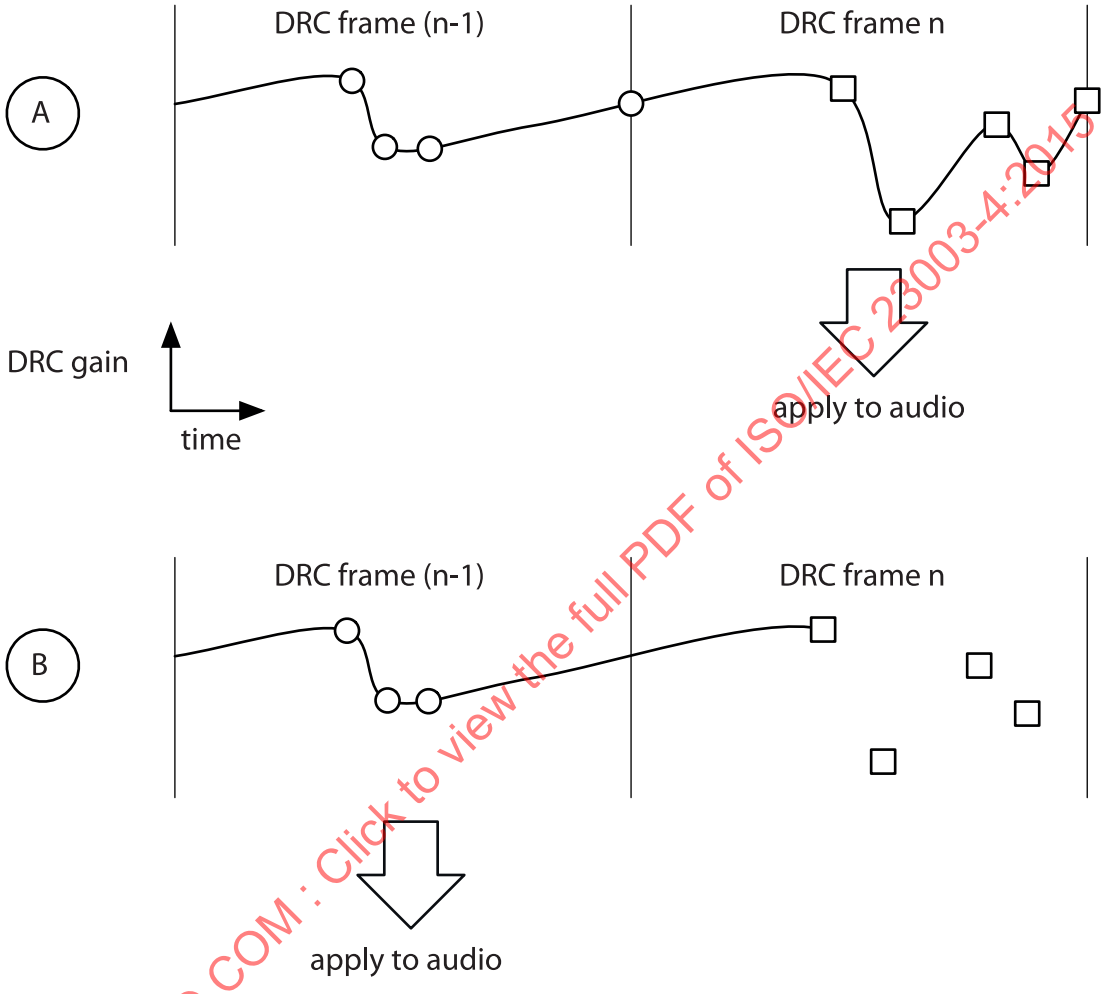

Figure 4 — Delay modes: low-delay and *fullFrame* = 1 (A), default delay and *fullFrame* = 0 (B). Spline nodes received in frame *n*-1 (circles) and frame *n* (squares). The solid line illustrates the interpolated DRC gain up to DRC frame *n*

The low-delay mode is suitable for decoders that don't have inherent delay such as a delay due to overlap-add. For instance this is the case for some lossless codecs. For any PCM audio format, low-delay mode shall be used.

For ISO/IEC 14496 (ISO base media file format) the delay mode is explicitly conveyed in the drcCoefficientsUniDrc() payload in the delayMode field (see Table 45). The encoding of delayMode is given in Table A.20.

6.4.9 Node reservoir

To reduce bitrate peaks, a so-called node reservoir can be employed. The node reservoir can be only applied in the default delay mode, where the DRC gain is applied with a delay of one DRC frame (see 6.4.8). The node reservoir mechanism shifts nodes from the current frame into the subsequent frame (see Figure 5). In order to keep the decoding delay constant, it is not allowed to shift the first node

of a frame since in default delay mode, the first node is needed to fully decode the previous frame. Thus, the maximum number of nodes that can be shifted to the subsequent frame is limited to *nNodes-1* nodes.

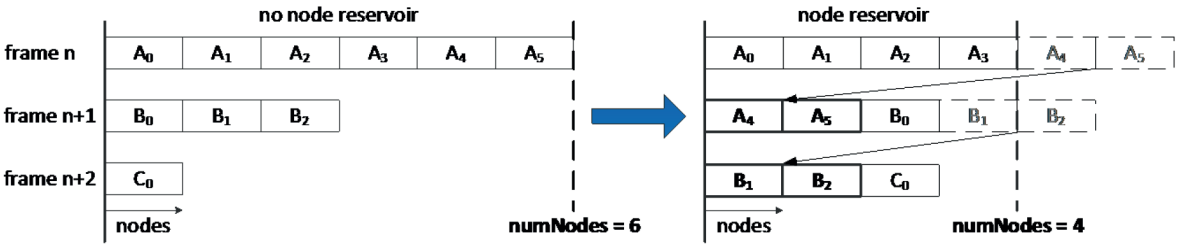

Figure 5 — Illustration of node reservoir mechanism: A_X, B_X, and C_X are nodes regularly transmitted in DRC frame n, n+1, and n+2, respectively (left). When the node reservoir is used, some nodes (bold) are transmitted with a delay of one frame (right)

Each node is defined by time, gain, and slope data (gainInterpolationType = 0). The gain and slope data have chronological order in the bitstream, therefore the data from the node reservoir precedes the regular nodes of a particular frame in the bitstream. During decoding, the nodes of the node reservoir can be identified because their time values are encoded with an offset of *drcFrameSize*. Therefore, the use of the node reservoir is not explicitly signalled. The offset is subtracted in the decoding process to obtain the correct time value.

The number of nodes shifted to the node reservoir in each frame is not transmitted explicitly, but results implicitly from the number of time values larger than *drcFrameSize*. The decoding of time values precedes the decoding of gain and slope values. Therefore, the assignment of gain and slope values to time values is known after decoding the time values. The decoding process is fully specified in 6.4.5 and 6.4.10.

6.4.10 Applying the compression

The interpolated gain values of each interpolation segment are concatenated to generate a complete gain vector *gain*[*g*][*b*][*t*] for the entire DRC frame. Finally, the gain vector is applied as shown in Table 19. The function *channelInDrcGroup*() returns TRUE if the current channel *c* belongs to the current DRC channel group. Please note that the scheduling of the interpolation segments depends on the delay mode as indicated in Table 19.

Table 19 — Concatenation of interpolation segments to a gain vector and application of the DRC gain vector to the audio channels

```
for(g=0; g<nDrcChannelGroups; g++) {
 for(b=0; b<nDrcBands[g]; b++) {
  if (delayMode == DELAY_MODE_DEFAULT) {
   nNodesNodeReservoir = 0;
   for (k=0; k<nNodes[g][b]; k++){
    if (tDRC[g][b][k] >= drcFrameSize){
     nNodesNodeReservoir++;
    }
   }
   for (k=0; k<nNodesNodeReservoir; k++){
    tDRCprev[g][b][nNodesPrev[g][b]+k]=tDRC[g][b][nNodes[g][b]-
                        nNodesNodeReservoir+k]-drcFrameSize;
    gDRCprev[g][b][nNodesPrev[g][b]+k]=gDRC[g][b][k];
    sDRCprev[g][b][nNodesPrev[g][b]+k]=sDRC[g][b][k];
   }
   memmove(&gDRC[g][b][0],&gDRC[g][b][nNodesNodeReservoir],
        nNodes[g][b]-nNodesNodeReservoir);
   memmove(&sDRC[g][b][0],&sDRC[g][b][nNodesNodeReservoir],
        nNodes[g][b]-nNodesNodeReservoir);
   nNodesPrev[g][b] = nNodesPrev[g][b] + nNodesNodeReservoir;
   nNodes[g][b] = nNodes[g][b] – nNodesNodeReservoir;
   for (k=0; k<nNodesPrev[g][b]-1; k++) {
    duration = tDRCprev[g][b][k+1] – tDRCprev[g][b][k];
    interpolationSegment= interpolateDrcGain(duration, gDRCprev[g][b][k],
              gDRCprev[g][b][k+1], sDRCprev[g][b][k], sDRCprev[g][b][k+1]);
    for (t=0; t<duration; t++)
    {
     gain[g][b][t+tDRCprev[g][b][k]] = interpolationSegment[t];
    }
   }
   k = nNodesPrev[g][b]-1;
   duration = drcFrameSize + tDRC[g][b][0] – tDRCprev[g][b][k];
   interpolationSegment= interpolateDrcGain(duration, gDRCprev[g][b][k],
                    gDRC[g][b][0], sDRCprev[g][b][k], sDRC[g][b][0]]);
   for (t=0; t<duration; t++) {
    gain[g][b][t+tDRCprev[g][b][k]] = interpolationSegment[t];
   }
  }
```

Table 19 *(continued)*

```
  else
  {
   k = nNodesPrev[g][b]-1;
   duration = tDRC[g][b][0]+1;
   interpolationSegment= interpolateDrcGain(duration, gDRCprev[g][b][k],
                    gDRC[g][b][0], sDRCprev[g][b][k], sDRC[g][b][0]]);
   for (t=1; t<=duration; t++)
   {
    gain[g][b][t-1] = interpolationSegment [t];
   }
   for (k=0; k<nNodes[g][b]-1; k++) {
    duration = tDRC[g][b][k+1] – tDRC[g][b][k];
    interpolationSegment= interpolateDrcGain(duration, gDRC[g][b][k],
                    gDRC[g][b][k+1], sDRC[g][b][k], sDRC[g][b][k+1]);
    for (t=1; t<=duration; t++)
    {
     gain[g][b][t+tDRC[g][b][k]] = interpolationSegment [t];
    }
   }
  }
  /* Apply gain to DRC bands of audio in each channel */
  for (c=0; c<nChannels; c++) {
   if (channelInDrcGroup(c, g)) {
    for (t=0; t<drcFrameSize; t++) {
     audioBandOut[c][b][t] = audioBandIn[c][b][t] * gain[g][b][t];
    }
   }
  }

  if (delayMode == DELAY_MODE_DEFAULT) {
   for (k=0; k<nNodes; k++) {
    gDRCprev[g][b][k] = gDRC[g][b][k];
    sDRCprev[g][b][k] = sDRC[g][b][k];
    tDRCprev[g][b][k] = tDRC[g][b][k];
   }
   nNodesPrev[g][b] = nNodes[g][b];
   for (t=0; t<drcFrameSize; t++) {
    gain[g][b][t] = gain[g][b][t + drcFrameSize];
   }
  }
  else {
   k=nNodes[g][b]-1;
   gDRCprev[g][b][k] = gDRC[g][b][k];
   sDRCprev[g][b][k] = sDRC[g][b][k];
   nNodesPrev[g][b] = nNodes[g][b];
  }
 }
}
for(g=0; g<nDrcChannelGroups; g++) {
 for(c=0; c<nChannels; c++) {
  if (channelInDrcGroup(c, g)) {
   for (t=0; t<drcFrameSize; t++) {
    sum = 0.0;
    for(b=0; b<nDrcBands[g]; b++) {
     sum = sum + audioBandOut[c][b][t];
    }
    audioSampleOut[c][t] = sum;
   }
  }
 }
}
```

Table 19 is based on the following assumption:

— *interpolationSegment* is a vector that contains the gain values of one interpolation segment.

— *duration* is an integer number describing the duration of the interpolation segment in units of audio sample intervals.

— *nNodes* is the number of gain values in the current DRC frame.

— *drcFrameSize* is the number of audio sample intervals in a DRC frame.

— Initialize the following variables:

```
gDRCprev[g][b][0]=0.0,
sDRCprev[g][b][0]=0.0,
nNodesPrev[g][b]=1,
if (timeAlignment==0)
        tDRCprev[g][b][0]=drcFrameSize-1
else
        tDRCprev[g][b][0]=drcFrameSize-timeDeltaMin+floor((timeDeltaMin-1)/2).
```

6.4.11 Multi-band DRC filter bank

When the DRC gains are applied in the time domain and a multi-band DRC is used, the time-domain audio signal must be split into sub-bands before the DRC gains are applied to the bands. The filter configuration parameters are conveyed by the drcCoefficientsUniDrc() defined in Table 44 or Table 45. The tables provide the bitstream syntax for the number of bands and the crossover frequency indices between bands. The number of time-domain DRC bands cannot be larger than four.

The time-domain audio signal is split into the specified number of bands by Linkwitz-Riley (LR) filters with a topology shown in Figure 6. If there are more than two bands, all-pass filters are added to compensate for the delay of the different outputs, so that they are all in phase. The low-pass and high-pass filters are implemented as second order sections (biquads).

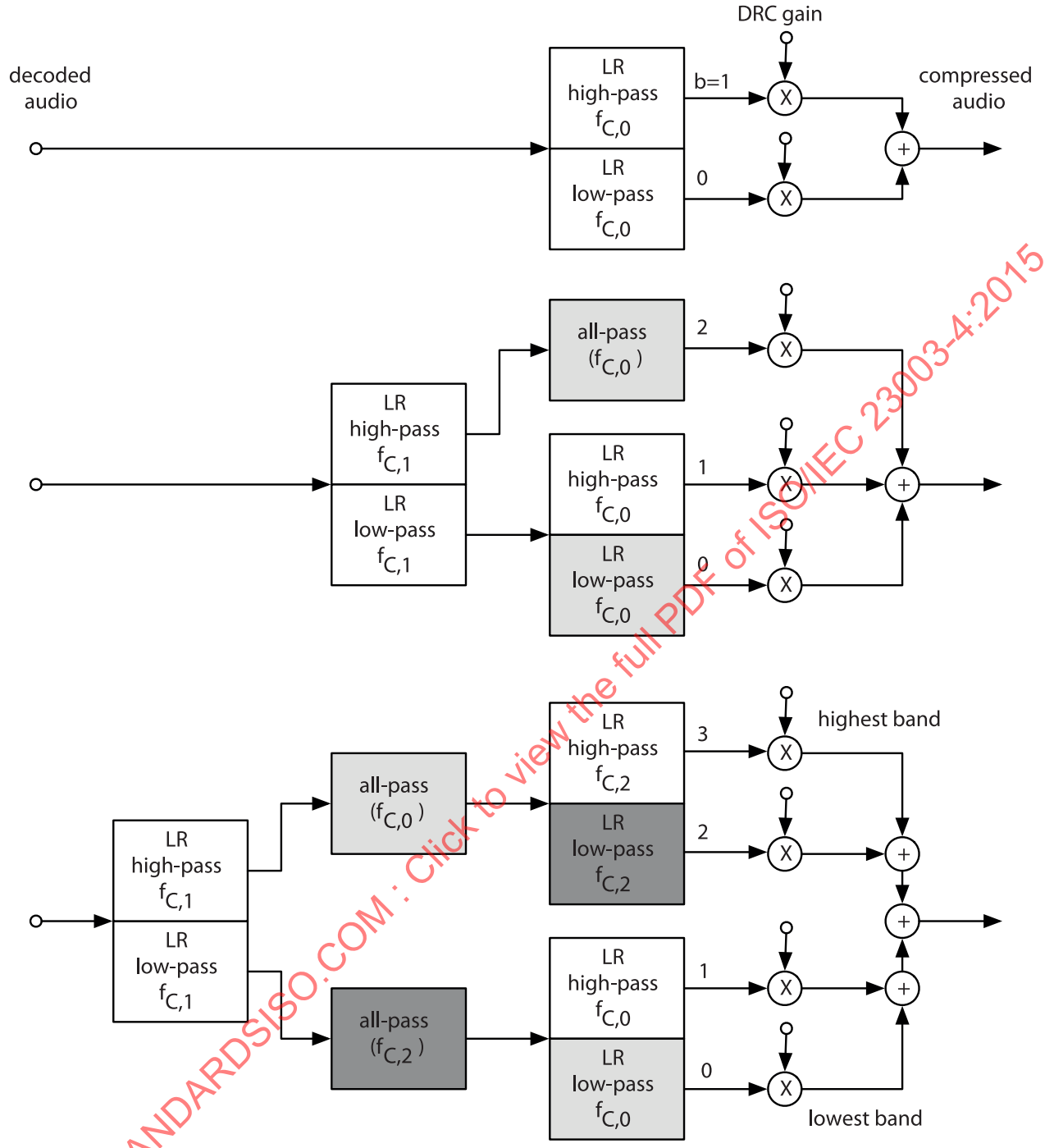

Figure 6 — Topology of Linkwitz-Riley crossover filters for 2, 3, and 4 bands. The band index b increases with the frequency of the band. The crossover frequencies $f_{C,b}$ increase with index b, i.e. $f_{C,b+1} > f_{C,b}$ Crossover frequencies in brackets of an all-pass filter specify the corresponding LR low-pass filter with the matching phase response

Each Linkwitz-Riley crossover filter is composed of a pair of a complementing low-pass and high-pass filter that results in a flat frequency response overall. Each LR low-pass filter is created by a cascade of two identical second-order Butterworth (BW) low-pass filters. Similarly, each LR high-pass filter is a cascade of two identical BW high-pass filters with the same order and cutoff frequency as the BW low-pass filters. Each all-pass filter is a second order IIR filter.

Each BW filter and each all-pass filter is implemented as second order section with the following transfer function.

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \tag{3}$$

Based on the crossover frequency indices in Table A.8, the decoder can look up the normalized crossover frequencies $f_{c,Norm}$ or the filter coefficient parameters γ and δ. The filter coefficients are then computed using Table 20 for the BW filters and Table 21 for the all-pass filters. The crossover frequencies f_c in Hz are computed by:

$$f_c = f_s \cdot f_{c,Norm} \tag{4}$$

Table 20 — Butterworth filter coefficient formulas

	BW low-pass	BW high-pass
Normalized cutoff frequency	$\omega_0 = \tan(\pi f_{c,Norm})$	
Intermediate parameters	$\delta = \frac{1}{1 + \sqrt{2\omega_0 + \omega_0^2}}$ $\gamma = \omega_0^2 \delta$	
Final BW filter coefficients	$a_{LP,0} = 1$ $a_{LP,1} = 2(\gamma - \delta)$ $a_{LP,2} = 2(\gamma + \delta) - 1$ $b_{LP,0} = \gamma$ $b_{LP,1} = 2\gamma$ $b_{LP,2} = \gamma$	$a_{HP,0} = 1$ $a_{HP,1} = 2(\gamma - \delta)$ $a_{HP,2} = 2(\gamma + \delta) - 1$ $b_{HP,0} = \delta$ $b_{HP,1} = -2\delta$ $b_{HP,2} = \delta$

The all-pass filters in Figure 6 are used to generate the same phase response as one of the LR low-pass filters (with matching gray level and matching f_c in Figure 6) so that the signals of all bands are in phase at the output of the filter bank. The all-pass filter coefficients are derived from the coefficients of the corresponding BW low-pass filter as shown in Table 21.

Table 21 — All-pass filter coefficient formulas

$a_{AP,0} = a_{LP,0}$ $a_{AP,1} = a_{LP,1}$ $a_{AP,2} = a_{LP,2}$ $b_{AP,0} = a_{LP,2}$ $b_{AP,1} = a_{LP,1}$ $b_{AP,2} = a_{LP,0}$

After the DRC gains are applied to the individual bands, the final audio signal is computed by adding all bands.

In configurations that apply different multiband filters to different channel groups, additional all-pass filters must be incorporated to achieve an in-phase output of all channel groups. The example in

Figure 7 shows three channel groups that are processed with a single band (0), two DRC bands (1), and four DRC bands (2). The overall phase response of the system is determined by the following steps:

— Construct a filter structure for each channel group according to the crossover frequencies and number of bands.

— Concatenate all filter structures to determine the target phase response.

— Add all-pass filters to each channel group filter structure to match the target phase response.

— Eliminate identical all-pass filters that appear in all channel groups to minimize the delay if possible while maintaining in-phase output of all channel groups.

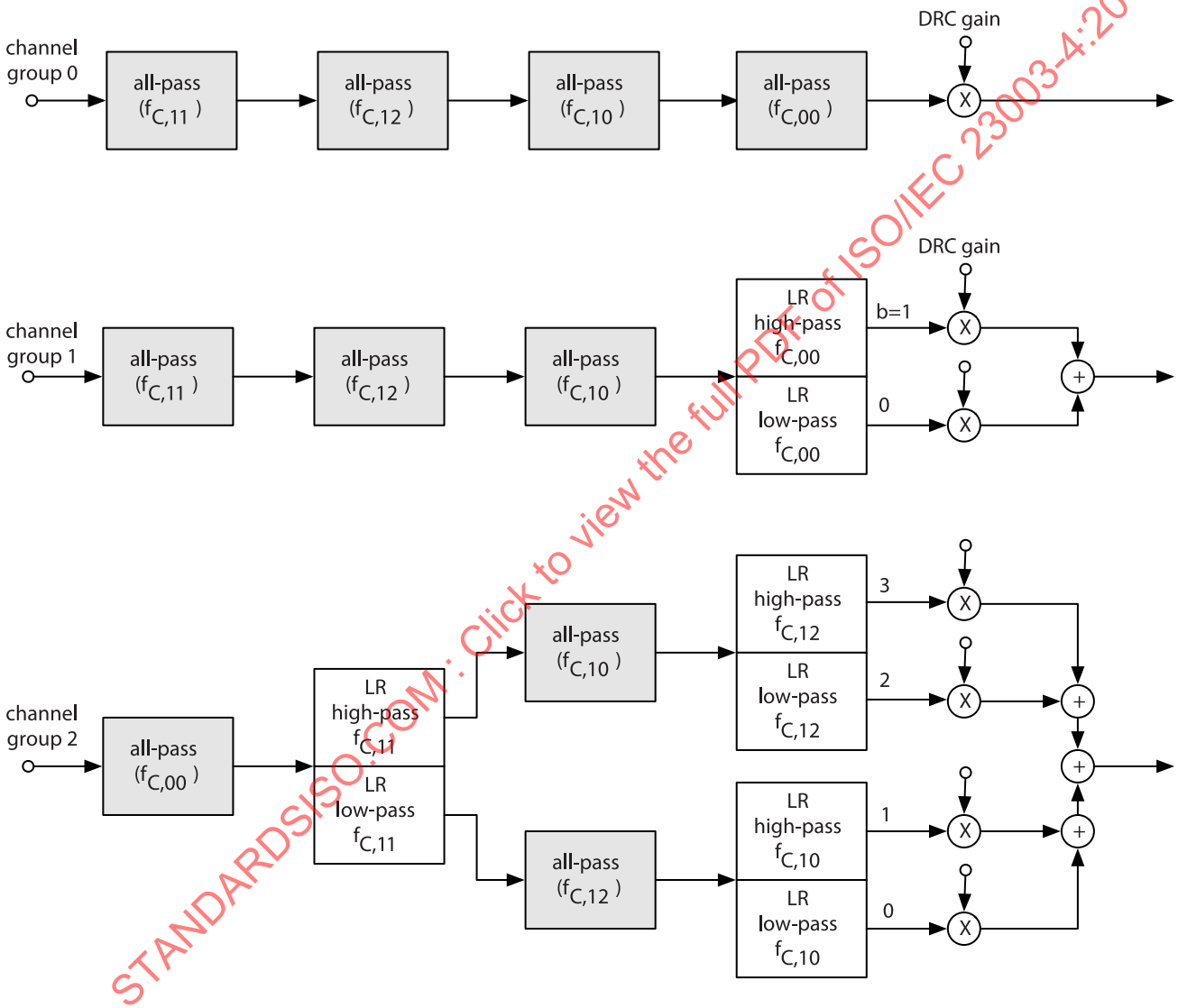

Figure 7 — Example of phase adjustments for time-domain multiband DRC with different DRC bands for each channel group

Support for up to three different time-domain multiband DRC filter banks with phase alignment of the corresponding channel groups is mandatory. This number includes full-band DRCs and counts each multiband filter bank with a different crossover frequency. If two DRC sets for time-domain application are combined using the dependsOnDrcSet field, not more than one of them can be a multiband DRC. In this case, the exact output waveform depends on the processing order of the DRC sets, which is defined in 6.3.5. However, if the multiband DRC set has a downmixId of 0x7F, the information whether it is processed before or after the downmix needs to be provided to fully define the output waveform, e.g. for conformance testing

Systems that process audio objects may not apply time-domain multi-band DRC to only a sub-set of objects since that will result in phase misalignment with other objects in the final mix.

6.5 Sub-band domain DRC

While the application of DRC gains in the time-domain is mandatory for MPEG AAC up to four DRC bands, other MPEG codecs might use sub-band domain DRC. The concept of sub-band domain DRC means that the existing sub-band signals of the audio decoder are subject to the DRC gain application. Therefore, it is not necessary to add time-domain band splitting for a multi-band DRC and it is possible to apply DRC gains before rendering and/or downmixing in the frequency domain. Table 22 contains a non-exhaustive list of codecs and the domain where the DRC gain is applied. Please note that the domain depends on the decoder configuration and not on the bitstream. For instance, if MPEG Surround is decoded with a plain AAC decoder, the DRC gains are applied in the time domain. Furthermore, the sub-band domain is not the MDCT domain of a core codec – it is usually the QMF domain. The applicable audio codec standard may specify the DRC application in more detail.

Table 22 — Domain of DRC gain applications for various MPEG decoders

Decoder	Time-domain DRC	Sub-band DRC
MPEG-4 AAC	✓	
MPEG-4 HE-AAC		✓
MPEG-D Surround		✓
MPEG-D SAOC		✓
MPEG-D USAC	✓	✓
MPEG-H	✓	✓

In the most general case, the DRC gains are decoded as described in 6.4.5. They are interpolated using the same technique as described in Table 18 and Table 19, however, the sampling rate of the interpolation result is lowered to match the sample rate of the sub-band signals. This can be achieved by sub-sampling the interpolated time-domain DRC gains by a factor of L or by directly interpolating using the sub-band sample rate as target.

In many cases of sub-band DRC, it is more efficient to override the default *deltaTmin* in the encoder by setting *timeDeltaMin* to a value that matches the sub-band sample interval. With that, unnecessary interpolation and downsampling can be avoided in the decoder and the DRC time resolution is not higher than necessary (see also D.1.2).

To reproduce the frequency response of the Linkwitz-Riley filters, the filter slopes are generated in the sub-band domain as illustrated by the pseudo code in Table 24 for *bandType*==1. The preliminary weighting coefficients reflect the filter responses including the slopes. The final weighting coefficients are obtained by normalizing the preliminary weighting coefficents, such that the sum of the weights of all DRC band gains at each decoder sub-band centre frequency is 1.0.

The down-sampling and delay operations can be expressed by the first part of pseudo code in Table 23. The remaining part shows how the weighting coefficients are applied to the DRC gains to generate the gains for the audio decoder sub-bands. The meaning of variables and functions of the pseudo code is explained in Table 25. The description assumes that the sample rates in all sub-bands are equal. If this is not the case, the down-sampling factor L needs to be adjusted for the different sub-band sample rates.

Table 23 — DRC gain down-sampling, overlap, and application in decoder sub-bands

```
/* resample DRC gain */
for (g=0; g<nDrcChannelGroups; g++) {
 for (b=0; b<nDrcBands[g]; b++) {
  for (m=0; m<drcFrameSizeSb; m++) {
   gainLr[g][b][m]=gain[g][b][(m*L+floor((L-1)/2)];
  }
 }
}
for (g=0; g<nDrcChannelGroups; g++) {
 if (nDrcBands[g] == 1) {
  for (s=0; s<nDecoderSubbands; s++) {
   overlapWeight[g][0][s] = 1.0;
  }
 } else {
  overlapWeight[g] = generateOverlapWeights(g)
 }
 for (b=0; b<nDrcBands[g]; b++) {
  for (s=0; s<nDecoderSubbands; s++) {
   for (m=0; m<drcFrameSizeSb; m++) {
    gainSb[g][s][m] += overlapWeight[g][b][s] * gainLr[g][b][m];
   }
  }
 }
}
/* apply DRC gain in sub-bands */
for (g=0; g<nDrcChannelGroups; g++) {
 for (c=0; c<nChannels; c++) {
  if (channelInDrcGroup(c, g)) {
   for (s=0; s<nDecoderSubbands; s++) {
    for (m=0; m<drcFrameSizeSb; m++) {
     audioSampleSbOut[c][s][m]=gainSb[g][s][m]*audioSampleSbIn[c][s][m];
    }
   }
  }
 }
}
```

Table 24 — Computation of overlap weigths

```
generateSlope(fCrossNormLo, fCrossNormHi) {
   float filterSlope = -24.0          /* filter slope in dB per octave /
   float log10_2_inv = 3.32192809;     /* 1.0 / log10(2) */
   float norm = 0.05 * filterSlope * log10_2_inv;
   for (s=0; s<nDecoderSubbands; s++) {
      if (fCenterNormSb[s]<fCrossNormLo) {
         response[s] = pow (10.0, norm * log10(fCrossNormLo / fCenterNormSb[s]));
      }
      else if (fCenterNormSb[s]<fCrossNormHi) {
         response[s] = 1.0;
      }
      else {
         response[s] = pow (10.0, norm * log10(fCenterNormSb[s] / fCrossNormHi));
      }
   }
   return (response)
}
generateOverlapWeights(g) {
   if (drcBandType[g] == 1) {
      fCrossNormLo = 0.0f;
      for (b=0; b<nDrcBands[g]; b++) {
         if (b<nDrcBands - 1) {
            fCrossNormHi = fCross[g][b+1];
         }
         else {
            fCrossNormHi = 0.5;
         }
         overlapWeight[b] = generateSlope (fCrossNormLo, fCrossNormHi);
         fCrossNormLo = fCrossNormHi;
      }
      for (s=0; s<nDecoderSubbands; s++) {
         wNorm[s] = overlapWeight[0][s];
         for (b=1; b<nDrcBands[g]; b++) {
            wNorm[s] += overlapWeight[b][s];
         }
      }
      for (s=0; s< nDecoderSubbands; s++) {
         for (b=0; b<nDrcBands[g]; b++) {
            overlapWeight[b][s] /= wNorm[s];
         }
      }
   } else {
      startSubBandIndex = 0;
      for (b=0; b<nDrcBands[g]; b++) {
         if (b < nDrcBands[g]-1) {
            stopSubBandIndex = startSubBandIndex[g][b+1]-1;
         } else {
            stopSubBandIndex = nDecoderSubbands-1;
         }
         for (s=0; s< nDecoderSubbands; s++) {
            if (s >= startSubBandIndex && s <= stopSubBandIndex) {
               overlapWeight[b][s] = 1.0;
            } else {
               overlapWeight[b][s] = 0.0;
            }
         }
         startSubBandIndex = stopSubBandIndex+1;
      }
   }
   return (overlapWeight);
}
```

Table 25 — Explanation of pseudo code items

Code item	Meaning
gainSb	DRC gain to be applied to decoder sub-bands
gainLr	Low-rate (resampled) DRC gain
fCross	Normalized crossover frequency
startSubBandIndex	Sub-band start index of multiband DRC band
drcFrameSizeSb	Number of sub-band samples per sub-band in one audio frame
nDecoderSubbands	Number of audio decoder sub-bands
fCenterNormSb	Center frequency of audio decoder sub-band
audioSampleSbIn	Decoded sub-band audio sample before dynamic compression
audioSampleSbOut	Decoded sub-band audio sample after dynamic compression

6.6 Loudness normalization

6.6.1 Overview

Loudness normalization is done by the DRC decoder if a target loudness level (targetLoudness) is supplied to the decoder. If no value is supplied, loudness normalization is disabled. If the loudness normalization would result in a too high level (that causes clipping or severe audible distortions introduced by a peak limiter), the output level can be decreased by reducing the gain as described in 6.3.2. The DRC decoder accepts an optional value *loudnessDeviationMax* to limit the applied gain reduction.

6.6.2 Loudness normalization based on target loudness

The target loudness is the desired output loudness level in LKFS (loudness, K-weighted[4]). If the target loudness value is supplied, a loudness normalization gain is applied to all channels of the time domain signal as the final step after all DRC related processing, if applicable.

The loudness normalization is based on the loudness metadata received in the applicable loudnessInfo() structure. The DRC decoder accepts an optional value *loudnessNormalizationGainDbMax* to limit the maximum loudness normalization gain. Second, it accepts an optional value *outputPeakLevelMax* to limit the peak level as described in 6.3.2. Third, it accepts an optional value *loudnessNormalizationGainModificationDb* to modifiy the default *loudnessNormalizationGainDb* based on external parameters or measurements available only at the decoder. *loudnessNormalizationGainModificationDb* can be different for each DRC frame.

The DRC decoder has an input for loudness normalization settings which accepts three indices to request a particular loudness measurement method, measurement system, and pre-processing as specified in Table 26, Table 27, and Table 28. If no settings are supplied, default values are used according to Table 29.

Table 26 — Permitted method definition indices for the loudness normalization settings

Index	Method definition
0	Default method
1	Program Loudness (as defined in loudnessInfo())
2	Anchor loudness (as defined in loudnessInfo())

Table 27 — Measurement system indices for the loudness normalization settings

Index	Measurement system
0	Default system
1	ITU-R BS.1770–3
2	User
3	Expert/Panel
4 *(reserved, not permitted)*	Reserved Measurement System A (RMS_A)
5 *(reserved, not permitted)*	Reserved Measurement System B (RMS_B)
6 *(reserved, not permitted)*	Reserved Measurement System C (RMS_C)
7 *(reserved, not permitted)*	Reserved Measurement System D (RMS_D)
8 *(reserved, not permitted)*	Reserved Measurement System E (RMS_E)

Table 28 — Permitted measurement system pre-processing indices for the loudness normalization settings

Index	Pre-processing
0	Default preprocessing
1	No preprocessing
2	High-pass filter with 500Hz cutoff frequency as specified in Table A.37

Table 29 — Default loudness normalization settings

Entry	Default value	Meaning of default
Measurement method	1	Program Loudness
Measurement system	1	ITU-R BS.1770–3
Pre-processing	1	No pre-processing

Table 30 — Matching order for measurement system

Order of selection	Requested measurement system as defined in Table 27							
	BS.1770–3	User	Expert/ Panel	RMS_A	RMS_B	RMS_C	RMS_D	RMS_E
	Selected measurement system as defined in Table A.37							
1	BS.1770–3	User	Expert/ Panel	RMS_A	RMS_B	RMS_C	RMS_D	RMS_E
2	RMS_C	RMS_E	RMS_D	RMS_C	RMS_C	RMS_B	Expert/ Panel	User
3	RMS_B	RMS_D	RMS_C	RMS_B	RMS_A	RMS_A	RMS_C	RMS_D
4	RMS_A	Expert/ Panel	RMS_B	BS.1770–3	BS.1770–3	BS.1770–3	RMS_B	Expert/ Panel
5	RMS_D	RMS_C	RMS_A	RMS_D	RMS_D	RMS_D	RMS_A	RMS_C
6	Expert/ Panel	RMS_B	BS.1770–3	Expert/ Panel	Expert/ Panel	Expert/ Panel	BS.1770–3	RMS_B
7	RMS_E	RMS_A	RMS_E	RMS_E	RMS_E	RMS_E	RMS_E	RMS_A
8	User	BS.1770–3	User	User	User	User	User	BS.1770–3

The request is then fulfilled by the decoder with best effort depending on which loudness metadata values are available. This is accomplished by the following steps:

1) From the applicable loudnessInfo() structure, select all loudness measurements that match the requested measurement method. The applicable loudnessInfo() structure is defined by the selected drcSetId and the requested downmixId as specified in 6.3. If no match is found, select all loudness measurements that match the other method for the applicable loudnessInfo() structure. If no match is found, fallback values are selected from a different loudnessInfo() structure as specified in 6.1.3, first for the requested method. Second, the other method is used if no match was found. Finally, if no match was found at all, disable loudness normalization and ignore the remaining steps. The same holds if no applicable loudnessInfo() structure is available.

2) Within all selected loudness measurements, find the one that matches the requested measurement system. If none is found, use the order given in Table 30 to find the best match. The value of contentLoudness is equal to the selected loudness value. Note that all column entries for the requested measurement system must be processed including the reserved measurement systems to ensure that measurement systems defined in the future can be found and applied. Although the reserved measurement systems are currently not permitted, the system shall be able to process requests of those systems.

3) If pre-processing is requested, the loudness value needs to be adjusted. The adjustment value ΔL_{pre} is the difference between the program loudness based on ITU-R BS.1770-3 with pre-processing and ITU-R BS.1770-3 without pre-processing. If the two loudness values are not available, the adjustment value is set to –2 dB. The value of contentLoudness is equal to the selected loudness value plus adjustment.

4. If pre-processing is requested and a value for deviceCutOffFrequency $f_{c,\text{pre}}$ is provided (see Table A.40), the value of contentLoudness is calculated as stated in step 3, but with a modified adjustment value ΔL_{pre}:

$$\Delta L_{\text{pre}} = \Delta L_{\text{pre}} \frac{\max[20, \min(500, f_{c,\text{pre}})] - 20}{500 - 20} \tag{5}$$

For each DRC frame, the loudness normalization is then applied according to Table 31.

Table 31 — Loudness normalization processing

```
if (targetLoudnessPresent) {
  loudnessNormalizationGainDb = targetLoudness – contentLoudness;
}
else {
  loudnessNormalizationGainDb = 0.0;
}
if (loudnessNormalizationGainDbMaxPresent) {
  loudnessNormalizationGainDb =
          min(loudnessNormalizationGainDb, loudnessNormalizationGainDbMax);
}
if (loudnessNormalizationGainModificationDbPresent) {
  gainNorm = pow(2.0, (loudnessNormalizationGainDb +
                       loudnessNormalizationGainModificationDb) / 6);
} else {
  gainNorm = pow(2.0, loudnessNormalizationGainDb / 6);
}
for (t=0; t<drcFrameSize; t++) {
  for(c=0; c<nChannels; c++) {
    audioSample[c][t] = gainNorm * audioSample[c][t];
  }
}
```

6.7 DRC in streaming scenarios

6.7.1 DRC configuration

The DRC configuration information is usually conveyed by the ISO Base Media File Format (ISO/IEC 14496-12) syntax. However, if a legacy streaming format such as ADTS is used to carry an MPEG Audio stream that does not support the ISO base media file format, the configuration information needs to be embedded in the audio stream. This can be achieved by sending uniDrcConfig() as part of the uniDrc() payload in-stream. Whenever uniDrcConfig() is transmitted, then a loudnessInfoSet() must be transmitted as well. Since the configuration information is only required at a lower rate than the DRC frame rate, presence flags are used to indicate when this information is available. Each received loudnessInfoSet() and uniDrcConfig() payload must be decoded and processed. DRC frames with configuration information should be synchronized with random access points of the employed streaming format.

Please note that the full DRC information can only be decoded after loudnessInfoSet() and uniDrcConfig() were received by the decoder. The repetition rate of uniDrcConfig() determines the maximum decoding delay for a decoder that starts decoding the content at an arbitrary position of the stream. In such a scenario when the decoder receives uniDrc() but no uniDrcConfig(), the decoder shall mute the audio output until uniDrcConfig() is available and the DRC information can be successfully decoded and applied to the audio signal. However, if the uniDrcConfig() is not received within 2.5 s, the decoder shall disable any DRC- and loudness-related processing and un-mute the audio.

If the DRC configuration information is both conveyed by the ISO base media file format syntax and by the uniDrcConfig() syntax, the in-stream syntax shall take precedence over the ISO base media file format syntax.

6.7.2 Error handling

If uniDrc() payloads are corrupted or lost on the receiver side during DRC gain application, the DRC decoder shall replace missing DRC frames by artificial uniDrc() payloads with drcGainCodingMode = 0 (“simple” mode). If the last valid DRC gain was an attenuation gain, gainInitialCode shall be set to the last valid DRC gain. If the last valid DRC gain was an amplification gain, gainInitialCode shall be set to 0dB. The same applies for all DRC channel groups and DRC bands. The described behaviour shall be repeated till the next valid uniDrc() payload is received. However, if a valid uniDrc() payload is not received within 2.5 s the decoder shall disable any DRC-and loudness-related processing until a valid uniDrc() payload is received.

6.8 DRC configuration changes during active processing

After the DRC tool decoder is initialized and has started to process audio frames, there are several events that can trigger a reconfiguration or an adjustment of parameters as summarized in Table 32 when certain values of the payload change with respect to the previous values.

Table 32 — Configuration changes for various events

Payload received	Condition	Result
loudnessInfoSet(), no uniDrcConfig()	If different values in loudnessInfoSet()	New selection of DRC set based on latest request. Reset of DRC tool if selected DRC set changes. Update loudness normalization gain
loudnessInfoSet(), uniDrcConfig()	If same values in all fields of uniDrcConfig() but different values in loudnessInfoSet()	New selection of DRC set based on latest request. Reset of DRC tool if selected DRC set changes. Update loudness normalization gain
	If different value in any field of uniDrcConfig()	Reset of DRC tool. New selection of DRC set based on latest request
uniDrcInterface()	If different value in any field ignoring compress, boost, and loudnessNormalizationGainModificationDb	New selection of DRC set based on latest request. Reset of DRC tool if selected DRC set changes.
	If different values in compress, boost, or loudnessNormalizationGainModificationDb fields but same values in all remaining fields	Update loudness normalization gain, compress, and boost values

A DRC tool decoder reset is associated with an initialization that discards the current state except for the latest request in form of a uniDrcInterface() payload (see Annex B) or equivalent and the last DRC gains and loudness normalization values that were applied. When the decoder is initialized after the reset, it uses the latest DRC request to select a DRC set. In order to prevent discontinuities in the output signal, linear interpolation is applied to smoothly transition from the previous gain sequence to the new one after reset. The interpolation starts at the minimum gain node of the received DRC frame when the reset occurs and ends at the minimum gain node of the next frame. Similarly, the loudness normalization gain is linearly interpolated during the first frame after reset. If the DRC set is applied in a location before or after downmix where no DRC set was applied before reset, the gain value to interpolate from is set to 0 dB. Vice versa, if a DRC set is not used anymore in a location where a DRC set has been used before, the gain value to interpolate to in the second frame is 0 dB.

If any of the active DRC sets during a DRC set change includes a time-domain multi-band DRC, no interpolation is applied. Instead, a fade-out shall be applied to the audio signal before the reset and a fade-in after the reset. The multi-band filter states are initialized with zero.

If the configuration change includes a target channel layout change or a target channel count change, no interpolation is applied from previous DRC gains nor from the previous normalization gain. The reset does not disrupt the output audio signal.

An update of the loudness normalization gain, loudnessNormalizationGainModificationDb, compress, or boost value does not involve a reset. The compress and boost values are smoothly adjusted to the new values by linear interpolation during one DRC frame. A loudness normalization gain adjustment in one DRC frame cannot be faster than ± 40 dB/s with linear interpolation. If more frames are needed to reach the new gain value, a rate of ± 40 dB/s is applied until the target value is reached. The loudness normalization gain is limited to a maximum value so that *outputPeakLevelMax* is not exceeded (see 6.3.2.2.3).

In the ISO base media file format framework (see ISO/IEC 14496-12), the same rules apply. For instance, if a LoudnessBox is received, the loudness normalization gain is updated and configuration changes can trigger a reset as indicated in Table 32.

7 Syntax

7.1 Syntax of DRC payload

Table 33 — Syntax of uniDrc() payload

Syntax	No. of bits	Mnemonic
uniDrc()		
{		
uniDrcLoudnessInfoSetPresent;	**1**	**bslbf**
if (uniDrcLoudnessInfoSetPresent==1) {		
uniDrcConfigPresent;	**1**	**bslbf**
if (uniDrcConfigPresent==1) {		
uniDrcConfig();		
}		
loudnessInfoSet();		
}		
uniDrcGain();		
}		

7.2 Syntax of DRC gain payload

Table 34 — Syntax of uniDrcGain() in-stream payload and for ISO/IEC 14496-12

Syntax	No. of bits	Mnemonic
uniDrcGain()		
{		
nDrcGainSequences = sequenceCount; /* from drcCoefficientsUniDrc */		
for (s=0; s<nDrcGainSequences; s++) {		
if (gainCodingProfile[s]<3) {		
drcGainSequence();		
}		
}		
uniDrcGainExtPresent;	**1**	**bslbf**
if (uniDrcGainExtPresent==1) {		
uniDrcGainExtension();		
}		
}		

Table 35 — Syntax of uniDrcGainExtension() payload

Syntax	No. of bits	Mnemonic
uniDrcGainExtension()		
{		
while (**uniDrcGainExtType** !=UNIDRCGAINEXT_TERM) {	**4**	**uimsbf**
extSizeBits = **bitSizeLen** + 4;	**3**	**uimsbf**
extBitSize = **bitSize** + 1;	**extSizeBits**	**uimsbf**
switch (uniDrcGainExtType) {		
/* add future extensions here */		
default:		
for (i=0; i<extBitSize; i++) {		
otherBit;	**1**	**bslbf**
}		
}		
}		
}		

7.3 Syntax of static DRC payload

Table 36 — Syntax of uniDrcConfig() payload

Syntax	No. of bits	Mnemonic
uniDrcConfig()		
{		
sampleRatePresent;	**1**	**bslbf**
if (sampleRatePresent==1) {		
bsSampleRate;	**18**	**uimsbf**
}		
downmixInstructionsCount;	**7**	**uimsbf**
drcDescriptionBasicPresent;	**1**	**bslbf**
if (drcDescriptionBasicPresent==1) {		
drcCoefficientsBasicCount;	**3**	**uimsbf**
drcInstructionsBasicCount;	**4**	**uimsbf**
} else {		
drcCoefficientsBasicCount = 0;		
drcInstructionsBasicCount = 0;		
}		
drcCoefficientsUniDrcCount;	**3**	**uimsbf**
drcInstructionsUniDrcCount;	**6**	**uimsbf**
channelLayout();		
for (i=0; i<downmixInstructionsCount; i++) {		
downmixInstructions();		
}		
for (i=0; i<drcCoefficientsBasicCount ; i++) {		
drcCoefficientsBasic();		
}		
for (i=0; i<drcInstructionsBasicCount ; i++) {		
drcInstructionsBasic();		
}		
for (i=0; i<drcCoefficientsUniDrcCount; i++) {		
drcCoefficientsUniDrc();		
}		
for (i=0; i<drcInstructionsUniDrcCount; i++) {		
drcInstructionsUniDrc();		
}		
uniDrcConfigExtPresent;	**1**	**bslbf**
if (uniDrcConfigExtPresent==1) {		
uniDrcConfigExtension();		
}		
}		

Table 37 — Syntax of loudnessInfoSet() payload

Syntax	No. of bits	Mnemonic
loudnessInfoSet()		
{		
loudnessInfoAlbumCount;	**6**	**uimsbf**
loudnessInfoCount;	**6**	**uimsbf**
for (i=0; i<loudnessInfoAlbumCount; i++) {		
loudnessInfo();		
}		
for (i=0; i<loudnessInfoCount; i++) {		
loudnessInfo();		
}		
loudnessInfoSetExtPresent;	**1**	**bslbf**
if (loudnessInfoSetExtPresent==1) {		
loudnessInfoSetExtension();		
}		
}		

Table 38 — Syntax of loudnessInfo() payload

Syntax	No. of bits	Mnemonic
loudnessInfo()		
{		
drcSetId;	**6**	**uimsbf**
downmixId;	**7**	**uimsbf**
samplePeakLevelPresent;	**1**	**bslbf**
if (samplePeakLevelPresent==1) {		
bsSamplePeakLevel;	**12**	**uimsbf**
}		
truePeakLevelPresent;	**1**	**bslbf**
if (truePeakLevelPresent==1) {		
bsTruePeakLevel;	**12**	**uimsbf**
measurementSystem;	**4**	**uimsbf**
reliability;	**2**	**uimsbf**
}		
measurementCount;	**4**	**uimsbf**
for (i=0; i<measurementCount; i++) {		
methodDefinition;	**4**	**uimsbf**
methodValue;	**2..8**	**vlclbf**
measurementSystem;	**4**	**uimsbf**
reliability;	**2**	**uimsbf**
}		
}		

Table 39 — Syntax of loudnessInfoSetExtension() payload

Syntax	No. of bits	Mnemonic
loudnessInfoSetExtension()		
{		
while (**loudnessInfoSetExtType** != UNIDRCLOUDEXT_TERM) {	**4**	**uimsbf**
extSizeBits = **bitSizeLen** + 4;	**4**	**uimsbf**
extBitSize = **bitSize** + 1;	**extSizeBits**	**uimsbf**
switch (loudnessInfoSetExtType) {		
/* add future extensions here */		
default:		
for (i=0; i<extBitSize; i++) {		
otherBit;	**1**	**bslbf**
}		
}		
}		
}		

Table 40 — Syntax of channelLayout() payload

Syntax	No. of bits	Mnemonic
channelLayout()		
{		
baseChannelCount;	**7**	**uimsbf**
layoutSignallingPresent;	**1**	**bslbf**
if (layoutSignallingPresent==1) {		
definedLayout;	**8**	**uimsbf**
if (definedLayout==0) {		
for (i=0; i<baseChannelCount; i++) {		
speakerPosition;	**7**	**uimsbf**
}		
}		
}		
}		

Table 41 — Syntax of downmixInstructions() payload

Syntax	No. of bits	Mnemonic
downmixInstructions()		
{		
downmixId;	**7**	**uimsbf**
targetChannelCount;	**7**	**uimsbf**
targetLayout;	**8**	**uimsbf**
downmixCoefficientsPresent;	**1**	**bslbf**
if (downmixCoefficientsPresent==1)		
for (i=0; i<targetChannelCount; i++) {		
for (j=0; j<baseChannelCount; j++) {		
bsDownmixCoefficient;	**4**	**bslbf**
}		
}		
}		
}		

Table 42 — Syntax of in-stream drcCoefficientsBasic() payload

Syntax	No. of bits	Mnemonic
drcCoefficientsBasic()		
{		
drcLocation;	**4**	**uimsbf**
drcCharacteristic;	**7**	**uimsbf**
}		

Table 43 — Syntax of drcCoefficientsBasic() payload for ISO/IEC 14496-12

```
aligned(8) class DRCCoefficientsBasic extends FullBox('udc1') {
        // N copies of this box, one of these per DRC_location
    bit(4)    reserved = 0;
    signed int(5) DRC_location;
    unsigned int(7) DRC_characteristic;
}
```

Table 44 — Syntax of in-stream drcCoefficientsUniDrc() payload

Syntax	No. of bits	Mnemonic
drcCoefficientsUniDrc()		
{		
drcLocation;	**4**	**uimsbf**
drcFrameSizePresent;	**1**	**bslbf**
if (drcFrameSizePresent==1) {		
bsDrcFrameSize;	**15**	**uimsbf**
}		
sequenceCount;	**6**	**uimsbf**
for (i=0; i<sequenceCount; i++) {		

Table 44 *(continued)*

Syntax	No. of bits	Mnemonic
gainCodingProfile;	**2**	**uimsbf**
gainInterpolationType;	**1**	**uimsbf**
fullFrame;	**1**	**uimsbf**
timeAlignment;	**1**	**uimsbf**
timeDeltaMinPresent;	**1**	**bslbf**
if (timeDeltaMinPresent==1) {		
bsTimeDeltaMin;	**11**	**uimsbf**
}		
if (gainCodingProfile!=3) {		
bandCount;	**4**	**uimsbf**
if (bandCount>1) {		
drcBandType;	**1**	**uimsbf**
}		
for (j=0; j<bandCount; j++) {		
drcCharacteristic;	**7**	**uimsbf**
}		
for (j=1; j<bandCount; j++) {		
if (drcBandType==1) {		
crossoverFreqIndex;	**4**	**uimsbf**
} else {		
startSubBandIndex;	**10**	**uimsbf**
}		
}		
}		
}		
}		

Table 45 — Syntax of drcCoefficientsUniDrc() payload for ISO/IEC 14496-12

```
aligned(8) class DRCCoefficientsUniDrc extends FullBox('udc2') {
                                        // N copies of this box, one of these per DRC_location
        bit(2)    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;
        }
        bit(1)    reserved = 0;
        int(1)    delayMode;
        unsigned int(6) sequence_count;
        for (sequence_index=1; sequence_index<=sequence_count; sequence_index++){
                        // each entry here in 1:1 correspondence with the
                        // gain coefficient sets (for the bands) in the given location
                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;       // must be >= 1
                        unsigned int(1) drc_band_type;
                        for (j = 1; j <= band_count; j++) {
                                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;
                                }
                        }
                }
        }
}
```

Table 46 — Syntax of in-stream drcInstructionsBasic() payload

Syntax	No. of bits	Mnemonic
drcInstructionsBasic()		
{		
drcSetId;	**6**	**uimsbf**
drcLocation;	**4**	**uimsbf**
downmixId;	**7**	**uimsbf**
additionalDownmixIdPresent;	**1**	**bslbf**
if (additionalDownmixIdPresent==1) {		
additionalDownmixIdCount;	**3**	**uimsbf**
for (j=0; j<additionalDownmixIdCount; j++) {		
additionalDownmixId;	**7**	**uimsbf**
}		
} else {		
additionalDownmixIdCount = 0;		
}		
drcSetEffect;	**16**	**bslbf**
if ((drcSetEffect and (3 < < 10)) == 0) {		
limiterPeakTargetPresent;	**1**	**bslbf**
if (limiterPeakTargetPresent==1) {		
bsLimiterPeakTarget;	**8**	**uimsbf**
}		
}		
drcSetTargetLoudnessPresent;	**1**	**bslbf**
if (drcSetTargetLoudnessPresent==1) {		
bsDrcSetTargetLoudnessValueUpper;	**6**	**uimsbf**
drcSetTargetLoudnessValueLowerPresent;	**1**	**bslbf**
if (drcSetTargetLoudnessValueLowerPresent==1) {		
bsDrcSetTargetLoudnessValueLower;	**6**	**uimsbf**
}		
}		
}		

Table 47 — Syntax of drcInstructionsBasic() payload for ISO/IEC 14496-12

```
aligned(8) class DRCInstructionsBasic extends FullBox('udi1') {
        // N copies, one for each overall combination DRC that can be applied
    bit(3)    reserved = 0;
    unsigned int(6) DRC_set_ID;        // must be non-zero and unique
    signed int(5) DRC_location;
    unsigned int(7) downmix_ID;      // if 0 – to base
        // if non-0, applies after downmix
        // if 0x7f, applies before or after downmix unsigned int(3) additional_downmix_ID_count;
    for (j=0; j<additional_downmix_ID_count; j++) {
        bit(1)    reserved = 0;
        unsigned int(7) additional_downmix_ID;
    }
    bit(16) DRC_set_effect;
    if ((DRC_set_effect & (3<<10)) == 0) {
        bit(7)    reserved = 0;
        bit(1) limiter_peak_target_present;
        if (limiter_peak_target_present == 1) {
            unsigned int(8) bs_limiter_peak_target;
        }
    }
    bit(7)    reserved = 0;
    bit(1) DRC_set_target_loudness_present;
    if (DRC_set_target_loudness_present==1) {
        bit(4)    reserved = 0;
        unsigned int(6) bs_DRC_set_target_loudness_value_upper;
        unsigned int(6) bs_DRC_set_target_loudness_value_lower;
    }
}
```

Table 48 — Syntax of in-stream drcInstructionsUniDrc() payload

Syntax	No. of bits	Mnemonic
drcInstructionsUniDrc()		
{		
drcSetId;	**6**	**uimsbf**
drcLocation;	**4**	**uimsbf**
downmixId;	**7**	**uimsbf**
additionalDownmixIdPresent;	**1**	**bslbf**
if (additionalDownmixIdPresent==1) {		
additionalDownmixIdCount;	**3**	**uimsbf**
for (j=0; j<additionalDownmixIdCount; j++) {		
additionalDownmixId;	**7**	**uimsbf**
}		
} else {		
additionalDownmixIdCount = 0;		
}		
drcSetEffect;	**16**	**bslbf**
if ((drcSetEffect & (3<<10)) == 0) {		
limiterPeakTargetPresent;	**1**	**bslbf**
if (limiterPeakTargetPresent==1) {		
bsLimiterPeakTarget;	**8**	**uimsbf**

Table 48 *(continued)*

Syntax	No. of bits	Mnemonic
}		
}		
drcSetTargetLoudnessPresent;	**1**	**bslbf**
if (drcSetTargetLoudnessPresent==1) {		
bsDrcSetTargetLoudnessValueUpper;	**6**	**uimsbf**
drcSetTargetLoudnessValueLowerPresent;	**1**	**bslbf**
if (drcSetTargetLoudnessValueLowerPresent==1) {		
bsDrcSetTargetLoudnessValueLower;	**6**	**uimsbf**
}		
}		
dependsOnDrcSetPresent;	**1**	**bslbf**
if (dependsOnDrcSetPresent==1) {		
dependsOnDrcSet;	**6**	**uimsbf**
} else {		
noIndependentUse;	**1**	**bslbf**
}		
channelCount = baseChannelCount;		
if ((drcSetEffect & (3<<10)) != 0) {		
for (i=0; i<channelCount; i++) {		
bsSequenceIndex;	**6**	**uimsbf**
duckingScalingPresent;	**1**	**bslbf**
if (duckingScalingPresent==1) {		
bsDuckingScaling;	**4**	**bslbf**
}		
repeatParameters;	**1**	**bslbf**
if (repeatParameters) {		
bsRepeatParametersCount;	**5**	**uimsbf**
i = i + repeatParametersCount;		
}		
}		
} else {		
if(downmixId!=0 && downmixId!=0x7F) {		
channelCount = targetChannelCountFromDownmixId;		
} else if (downmixId==0x7F) {		
channelCount = 1;		
}		
for (i=0; i<channelCount; i++) {		
bsSequenceIndex;	**6**	**uimsbf**
repeatSequenceIndex;	**1**	**bslbf**
if (repeatSequenceIndex) {		
bsRepeatSequenceCount;	**5**	**uimsbf**

Table 48 *(continued)*

Syntax	No. of bits	Mnemonic
i = i + repeatSequenceCount;		
}		
}		
for (i=0; i<nDrcChannelGroups; i++) {		
gainScalingPresent;	**1**	**bslbf**
if (gainScalingPresent==1) {		
bsAttenuationScaling;	**4**	**uimsbf**
bsAmplificationScaling;	**4**	**uimsbf**
}		
gainOffsetPresent;	**1**	**bslbf**
if (gainOffsetPresent==1) {		
bsGainOffset;	**6**	**bslbf**
}		
}		
}		
}		

Table 49 — Syntax of drcInstructionsUniDrc() payload for ISO/IEC 14496-12

```
aligned(8) class DRCInstructionsUniDrc extends FullBox('udi2') {
        // N copies, one for each overall combination DRC that can be applied
    bit(3)    reserved = 0;
    unsigned int(6) DRC_set_ID;        // must be non-zero and unique
    signed int(5) DRC_location;
    unsigned int(7) downmix_ID;      // if 0 – to base
        // if non-0, applies after downmix
        // if 0x7f, applies before or after downmix unsigned int(3) additional_downmix_ID_count;
    for (j=0; j<additional_downmix_ID_count; j++) {
        bit(1)    reserved = 0;
        unsigned int(7) additional_downmix_ID;
    }
    unsigned int(16) DRC_set_effect;
    if ((DRC_set_effect & (3<<10)) == 0) {
        bit(7)    reserved = 0;
        bit(1) limiter_peak_target_present;
        if (limiter_peak_target_present == 1) {
            unsigned int(8) bs_limiter_peak_target;
        }
    }
    bit(7)    reserved = 0;
    bit(1) DRC_set_target_loudness_present;
    if (DRC_set_target_loudness_present==1) {
        bit(4)    reserved = 0;
        unsigned int(6) bs_DRC_set_target_loudness_value_upper;
        unsigned int(6) bs_DRC_set_target_loudness_value_lower;
    }
    bit(1)    reserved = 0;
    unsigned int(6)  depends_on_DRC_set;
        // if non-zero, must match a DRC that must be applied before this one
        // it must be examined to see if that is before/after downmix
    if (depends_on_DRC_set==0) {
        bit(1) no_independent_use;
    } else {
        bit(1)    reserved = 0;
    }
    unsigned int(8) channel_count;
        // if downmix ID==0x7F, must be 1, and applies to all channels
        // else must match channel count of the signal
    for (i=1; i<=channel_count; i++){
        unsigned int (8) channel_group_index;
            // 1-based channel_group_index for channel
```

Table 49 *(continued)*

```
        }
        for (i=1; i<=channel_group_count; i++){
                bit(2)    reserved = 0;
                unsigned int(6) bs_sequence_index;
                        // if 0, then this channel_group/object is not processed
                        // else 1-based index into the DRC sequence array for this location
        }
        if ((DRC_set_effect & (3<<10)) != 0) {
                for (i=1; i<=channel_group_count; i++) {
                        bit(7)    reserved = 0;
                        bit(1) ducking_scaling_present;
                        if (ducking_scaling_present==1) {
                                bit(4)    reserved = 0;
                                bit(4) bs_ducking_scaling;
                        }
                }
        }
        else {
                for (i=1; i<=channel_group_count; i++) {
                        bit(7)    reserved = 0;
                        bit(1) gain_scaling_present;
                        if (gain_scaling_present==1) {
                                unsigned int(4) bs_attenuation_scaling;
                                unsigned int(4) bs_amplification_scaling;
                        }
                        bit(7)    reserved = 0;
                        bit(1) gain_offset_present;
                        if (gain_offset_present==1) {
                                bit(2)    reserved = 0;
                                bit(6) bs_gain_offset;
                        }
                }
        }
}
```

Table 50 — 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) {		
/* add future extensions here */		
default:		
for (i=0; i<extBitSize; i++) {		
otherBit;	**1**	**bslbf**
}		
}		
}		
}		

7.4 Syntax of DRC gain sequence

Table 51 — Syntax of drcGainSequence() payload

Syntax	No. of bits	Mnemonic
drcGainSequence()		
{		
for(b=0; b<nDrcBands[s]; b++) {		
drcGainCodingMode[s][b];	**1**	**uimsbf**
if (drcGainCodingMode[s][b] == 0) {		
gainInitialCode[s][b];	**1..11**	**vlclbf**
nNodes[s][b] = 1;		
}		
else {		
k=0;		
do {		
k++;		
}		
while (**endMarker**[s][b][k-1] != 1);	**1**	**uimsbf**
nNodes[s][b] = k;		
if (gainInterpolationType == 0) {		
for (k=0; k<nNodes[s][b]; k++) {		
slopeCode[s][b][k++];	**1..9**	**vlclbf**
}		
}		
if (fullFrame == 0) {		
frameEndFlag[s][b];	**1**	**uimsbf**
}		
else {		
frameEndFlag[s][b] = 1;		
}		
for (k=0; k<nNodes[s][b]-1; k++) {		
timeDeltaCode[s][b][k];	**2..Z+2**	**vlclbf**
}		
if (frameEndFlag[s][b] == 0) {		
k=nNodes[s][b]-1;		
timeDeltaCode[s][b][k];	**2..Z+2**	**vlclbf**
}		
gainInitialCode[s][b];	**1..11**	**vlclbf**
for (k=1; k<nNodes[s][b]; k++) {		
gainDeltaCode[s][b][k-1];	**1..11**	**vlclbf**
}		
}		
}		
}		

Annex A
(normative)

Tables

A.1 Coding of DRC gain values

Table A.1 — Coding of regular initial DRC gain values (*gainCodingProfile* == 0)

Encoding	Size	Mnemonic	gainInitial in [dB]	Range
$\{\sigma, \mu\}$	{1 bit, 8 bits}	{uimsbf, uimsbf}	$g_{DRC}(0) = (-1)^{\sigma}\,\mu 2^{-3}$	−31.875...31.875 dB, 0.125 dB step size

Table A.2 — Coding of initial DRC gain values for fading only (*gainCodingProfile* == 1)

Encoding	Size	Mnemonic	gainInitial in [dB]	Range
0	1 bit	bslbf	0	0
$\{1, \mu\}$	{1 bit, 10 bits}	{bslbf, uimsbf}	$g_{DRC}(0) = -(\mu + 1)2^{-3}$	−128...-0.125 dB, 0.125 dB step size

Table A.3 — Coding of initial DRC gain values for clipping prevention and ducking only (*gainCodingProfile* == 2)

Encoding	Size	Mnemonic	gainInitial in [dB]	Range
0	1 bit	bslbf	0	0
$\{1, \mu\}$	{1 bit, 8 bits}	{bslbf, uimsbf}	$g_{DRC}(0) = -(\mu + 1)2^{-3}$	−32...-0.125 dB, 0.125 dB step size

Table A.4 — Coding of regular DRC gain differences (*gainCodingProfile* ∈ [0,1])

Codeword size [bits]	gainValueDelta binary encoding	gainDelta in [dB]
4	0x000	−2.0
9	0x039	−1.875
11	0x0E2	−1.750
11	0x0E3	−1.625
10	0x070	−1.500
10	0x1AC	−1.375
10	0x1AD	−1.250
9	0x0D5	−1.125
7	0x00F	−1.000
7	0x034	−0.875
7	0x036	−0.750
6	0x019	−0.625
5	0x002	−0.500
5	0x00F	−0.375

Table A.4 *(continued)*

Codeword size [bits]	gainValueDelta binary encoding	gainDelta in [dB]
3	0x001	−0.250
2	0x003	−0.125
3	0x002	0.000
2	0x002	0.125
6	0x018	0.250
6	0x006	0.375
7	0x037	0.500
8	0x01D	0.625
9	0x0D7	0.750
9	0x0D4	0.875
5	0x00E	1.000

Table A.5 — Coding of DRC gain differences for clipping prevention and ducking only (*gainCodingProfile* == 2)

Codeword size [bits]	gainValueDelta binary encoding	gainDelta in [dB]
7	0x06A	−4.000
11	0x07A	−3.875
11	0x07B	−3.750
9	0x1AD	−3.625
10	0x03C	−3.500
9	0x1AC	−3.375
9	0x1A6	−3.250
9	0x0CD	−3.125
10	0x19E	−3.000
10	0x19F	−2.875
9	0x0CE	−2.750
9	0x1A7	−2.625
9	0x01F	−2.500
9	0x0CC	−2.375
8	0x0D2	−2.250
8	0x0AB	−2.125
8	0x0AA	−2.000
8	0x04F	−1.875
7	0x054	−1.750
7	0x068	−1.625
7	0x026	−1.500
7	0x006	−1.375
6	0x02B	−1.250
6	0x028	−1.125

Table A.5 *(continued)*

Codeword size [bits]	gainValueDelta binary encoding	gainDelta in [dB]
6	0x002	−1.000
5	0x011	−0.875
5	0x00E	−0.750
4	0x00C	−0.625
4	0x009	−0.500
4	0x005	−0.375
4	0x003	−0.250
3	0x007	−0.125
4	0x001	0.000
4	0x00B	0.125
5	0x005	0.250
5	0x004	0.375
5	0x008	0.500
5	0x000	0.625
5	0x00D	0.750
5	0x00F	0.875
5	0x010	1.000
5	0x01B	1.125
6	0x012	1.250
6	0x018	1.375
6	0x029	1.500
7	0x032	1.625
8	0x04E	1.750
8	0x0D7	1.875
8	0x00E	2.000

A.2 Coding of time differences

Table A.6 — Coding of time differences with *nNodesMax* = N_{DRC} and *Z* = ceil(log2(2nNodesMax*))**

Encoding	Size	Mnemonic	Time difference	Range
0x0	2 bits	bslbf	tDrcDelta = 1	1
{0x1,μ}	{2 bits, 2 bits}	{bslbf, uimsbf}	tDrcDelta = μ+2	2..5
{0x2,μ}	{2 bits, 3 bits}	{bslbf, uimsbf}	tDrcDelta = μ+6	6..13
{0x3,μ}	{2 bits, *Z* bits}	{bslbf, uimsbf}	tDrcDelta = μ+14	14..2*nNodesMax-1

A.3 Coding of slope steepness

Table A.7 — Coding of slope steepness (gainInterpolationType == 0)

Codeword size [bits]	Slope steepness binary encoding	Slope steepness
6	0x018	−3.0518
8	0x042	−1.2207
7	0x032	−0.4883
5	0x00A	−0.1953
5	0x009	−0.0781
5	0x00D	−0.0312
2	0x000	−0.0050
1	0x001	0.000
4	0x007	0.0050
5	0x00B	0.0312
6	0x011	0.0781
9	0x087	0.1953
9	0x086	0.4883
7	0x020	1.2207
7	0x033	3.0518

A.4 Coding of normalized crossover frequencies

Table A.8 — Coding of normalized crossover frequencies and associated filter coefficient parameters

crossoverFreqIndex	$f_{c,\mathrm{Norm}}$	γ	δ
0	2/1024	0.0000373252	0.9913600345
1	3/1024	0.0000836207	0.9870680830
2	4/1024	0.0001480220	0.9827947083
3	5/1024	0.0002302960	0.9785398263
4	6/1024	0.0003302134	0.9743033527
5	2/256	0.0005820761	0.9658852897
6	3/256	0.0012877837	0.9492662926
7	2/128	0.0022515827	0.9329321561
8	3/128	0.0049030350	0.9010958535
9	2/64	0.0084426929	0.8703307793
10	3/64	0.0178631928	0.8118317459
11	2/32	0.0299545822	0.7570763753
12	3/32	0.0604985076	0.6574551915
13	2/16	0.0976310729	0.5690355937
14	3/16	0.1866943331	0.4181633458
15	2/8	0.2928932188	0.2928932188

A.5 Coding of DRC gain extension types

Table A.9 — UniDrc gain extension types

Symbol	Value of uniDrcGainExtType	Purpose
UNIDRCGAINEXT_TERM	0x0	Termination tag
(reserved)	*(All remaining values)*	For future use

A.6 Coding of static DRC payload

A.6.1 Coding of top level fields of uniDrcConfig() and loudnessInfoSet()

Table A.10 — Coding of top level fields of uniDrcConfig() and loudnessInfoSet()

Field label	Encoding	Mnemonic	Decoded value	Description
bsSampleRate	μ 18 bits	uimsbf	$f_s = \mu$+1 000	Audio sample rate in [Hz]
downmixInstructionsCount	μ 7 bits	uimsbf	$N_D = \mu$	Number of downmixInstructions() blocks
drcCoefficientsBasicCount, drcCoefficientsUniDrcCount	μ 3 bits	uimsbf	$N_C = \mu$	Number of drcCoefficients blocks
drcInstructionsBasicCount, drcInstructionsUniDrcCount	μ 4/6 bits	uimsbf	$N_I = \mu$	Number of drcInstructions blocks
loudnessInfoAlbumCount	μ 6 bits	uimsbf	$N_{LA} = \mu$	Number of loudnessInfo() blocks for albums
loudnessInfoCount	μ 6 bits	uimsbf	$N_L = \mu$	Number of loudnessInfo() blocks

A.6.2 Coding of loudnessInfoSet extension types

Table A.11 — loudnessInfoSet extension types

Symbol	Value of loudnessInfoSetExtType	Purpose
UNIDRCLOUDEXT_TERM	0x0	Termination tag
(reserved)	*(All remaining values)*	For future use

A.6.3 Coding of DRC configuration extension types

Table A.12 — UniDrc configuration extension types

Symbol	Value of uniDrcConfigExtType	Purpose
UNIDRCCONFEXT_TERM	0x0	Termination tag
(reserved)	*(All remaining values)*	For future use

A.6.4 Coding of metadata that appears in multiple logical blocks of uniDrcConfig()

Table A.13 — Coding of metadata that appears in multiple logical blocks

Metadata field	Description
drcLocation	See 6.1.2.3
drcSetId	A unique number for each drcInstructions block. 0x0 reserved.
dependsOnDrcSet	Same encoding as drcSetId.
downmixId	A unique number for each downmixInstructions() block. 0x00 and 0x7F are reserved. A value of 0x7F in drcInstructions indicates that the DRC set can be applied to any downmix or to the original channel layout. In this case the DRC set must contain a single DRC gain sequence that is applied to all channels. A downmixId of zero indicates that the DRC set is applied to the base layout. See also ISO/IEC 14496-12.

A.6.5 Coded metadata in channelLayout()

Table A.14 — Coding of metadata in channelLayout()

Metadata field	Description
baseChannelCount	number of channels in the base layout / original audio input
definedLayout	See ChannelConfiguration in ISO/IEC 23001-8 (CICP)
speakerPosition	See OutputChannelPosition in ISO/IEC 23001-8 (CICP)

A.6.6 Coded metadata in downmixInstructions()

Table A.15 — Coding of metadata in downmixInstructions()

Metadata field	Description
targetLayout	See ChannelConfiguration in ISO/IEC 23001-8 (CICP)
bsDownmixCoefficient	See ISO/IEC 14496-12 (ISO base media file format)

A.6.7 Coded metadata in drcCoefficientsBasic() and drcCoefficientsUniDrc()

Table A.16 — Coding of metadata in drcCoefficientsBasic() and drcCoefficientsUniDrc()

Metadata field	Description
gainCodingProfile	See Table A.18
gainInterpolationType	See Table A.19
fullFrame	A value of 1 signals that the last node is always at the end of the frame. In low-delay mode, it shall be 1.
timeAlignment	A bit field that indicates whether the gain sample is aligned with the end (0) or the centre (1) of the *deltaTmin* interval.
delayMode	A bit field that indicates whether the received gains are applied immediately or with a delay of one frame (see Table A.20).
bsTimeDeltaMin	If present, this field indicates the custom time resolution of a DRC sequence which overrides the default *deltaTmin* value. The values are encoded according to Table A.21.
drcCharacteristic	A value of 0 means that the DRC characteristic is undefined for a DRC sequence. Values > 11 are reserved. See ISO/IEC 23001-8 (CICP)
drcBandType	A bit field, which signals if the "crossoverFreqIndex-syntax" (1) or the "startSubBandIndex-syntax" (0) should be used. If multi-band DRC gains are applied in the time-domain by using the multi-band DRC filter bank like specified in 6.4.11, only the "crossoverFreqIndex-syntax" is allowed. Note that if the "startSubBandIndex-syntax" is used, the frequency smoothing/fading with overlap weights according to Table 24 is not applied.
crossoverFreqIndex	See Table A.8
startSubBandIndex	zero-based sub-band index for an available sub-band domain. The field is used to signal the start index for a specific DRC band.

Table A.17 — Coding of bsDrcFrameSize field

Field	Encoding	Mnemonic	drcFrameSize	Range
bsDrcFrameSize	μ 15 bits	uimsbf	$M_{DRC} = \mu+1$	DRC frame size in units of audio sample intervals. Range: 1...2^{15}

Table A.18 — Coding of gainCodingProfile field

Value	Meaning
0	Regular gain coding
1	Fading gain coding
2	Clipping prevention and ducking gain coding
3	Constant gain (no gain sequence is transmitted)

Table A.19 — Coding of gainInterpolationType field

Value	Meaning
0	spline interpolation
1	linear interpolation

Table A.20 — Coding of delayMode field

Value	Meaning
0	Regular delay (Gains are applied with a delay of one frame)
1	Low delay (Received gains are applied immediately)

Table A.21 — Coding of bsTimeDeltaMin field

Field	Encoding	Mnemonic	timeDeltaMin	Range
bsTimeDeltaMin	μ 11 bits	uimsbf	deltaTmin = μ+1	deltaTmin in units of audio sample intervals. Range: 1...2^{11}

Table A.22 — Coding of drcCharacteristic field

See ISO/IEC 23001-8

Table A.23 — Coding of startSubBandIndex field

Field	Encoding	Mnemonic	Decoded value	Range
startSubBandIndex	μ 10 bits	Uimsbf	startSubBandIndex = μ	startSubBandIndex in zero-based indices of the sub-band domain. Range: 0...2^{10}-1

A.6.8 Coded metadata in drcInstructionsBasic() and drcInstructionsUniDrc()

Table A.24 — Coding of metadata in drcInstructionsBasic() and drcInstructionsUniDrc()

Metadata field	Description
bsSequenceIndex	A unique number which specifies which DRC sequence should be assigned to which channel/object of the configuration specified in downmixId. The reserved value 0x00 indicates that the assigned channel will not be processed by the DRC tool. All other values indicate the assigned DRC sequence index plus one. Due to the reserved value, the number of DRC gain sequences in one location cannot be more than 63. Note that *bsSequenceIndex* is effectively a 1-based index. The sequence indices (*sequenceIndex*) used for processing are 0-based.
additionalDownmixIdCount	Number of additional downmixIds connected with one DRC set. Note that only one DRC channel group is allowed if an additional downmixId refers to a different target channel layout than the first downmixId.
additionalDownmixId	Additional downmixId connected with a drcSetId. This field may be used to declare additional target layouts for which one single DRC set provides suitable gains, e.g. for clipping prevention of multiple target layouts with the same DRC set. Note that only one DRC channel group is allowed if an additional downmixId refers to a different target channel layout than the first downmixId.
bsRepeatSequenceCount	A field which declares that the current sequence index should be repeated for multiple channels. The assignment for-loop continues at position "i = i + repeatSequenceCount". bsRepeatSequenceCount is encoded according to Table A.26. If a sequence index should be repeated more than 32 times, bsSequenceIndex has to be signalled again.
bsRepeatParametersCount	A field similar to bsRepeatSequenceCount to indicate if the sequence index and the duckingScaling factor should be repeated for multiple channels. bsRepeatParametersCount is encoded according to Table A.26.
noIndependentUse	A flag which signals that the DRC set can only be used in combination with another DRC set as indicated by the *dependsOnDrcSet* field.
bsDrcSetTargetLoudness-ValueUpper	A field which contains the upper limit of the target loudness of a DRC set. The default value is 0 dB. The values are encoded according to Table A.31.
bsDrcSetTargetLoudness-ValueLower	A field which contains the lower limit of the target loudness of a DRC set. The default value is −63 dB. The values are encoded according to Table A.31.

Table A.25 — Coding of bsSequenceIndex

Encoding	Size	Mnemonic	sequenceIndex	Range
μ	6 bits	uimsbf	μ−1 if $\mu > 0$, else undefined	0...62

Table A.26 — Coding of bsRepeatParametersCount and bsRepeatSequenceCount field

Encoding	Size	Mnemonic	repeatParametersCount, repeatSequenceCount	Range
μ	5 bits	uimsbf	$N_R = \mu+1$	1...32

Table A.27 — Coding of bsLimiterPeakTarget field

Encoding	Size	Mnemonic	limiterPeakTarget in [dBFS]	Range
μ	8 bits	uimsbf	$L_{PT} = -\mu 2^{-3}$	−31.875 ... 0 dBFS, 0.125 dB step size

Table A.28 — Coding of bsDuckingScaling field

Encoding	Size	Mnemonic	duckingScaling	Range
$\{\sigma,\mu\}$	{1 bit, 3 bits}	{uimsbf, uimsbf}	$w = 1 + (-1)^{\sigma}(1+\mu)2^{-3}$	1 ± [0.125...1], 0.125 step size

Table A.29 — Coding of bsAttenuationScaling and bsAmplificationScaling field

Encoding	Size	Mnemonic	attenuationScaling, amplificationScaling	Range
μ	4 bits	uimsbf	$w = \mu 2^{-3}$	0... 1.875, 0.125 step size

Table A.30 — Coding of bsGainOffset field

Encoding	Size	Mnemonic	gainOffset in [dB]	Range
$\{\sigma,\mu\}$	{1 bit, 5 bits}	{uimsbf, uimsbf}	$g_{offs} = (-1)^{\sigma}(1+\mu)2^{-2}$	±[0.25...8], 0.25 step size

Table A.31 — Coding of bsDrcSetTargetLoudnessValueUpper/-Lower field

Encoding	Size	Mnemonic	drcSetTargetLoudnessValue in [dBFS]	Range
μ	6 bits	uimsbf	drcSetTargetLoudnessValue = μ − 63	−63 ... 0 dBFS, 1 dB step size

Table A.32 — Coding of drcSetEffect field. A bit value of 1 indicates that the effect is present

bit position	drcSetEffect	Short name	Description
1 (LSB)	Late night	“Night”	For quiet environment, listening at low level, avoiding to disturb others.
2	Noisy environment	“Noisy”	Optimized to get the best experience in noisy environments, for instance by amplifying soft sections.
3	Limited playback range	“Limited”	Reduced dynamic range to improve quality on playback devices with limited dynamic range.
4	Low playback level	“LowLevel”	Listening at a low playback level.
5	Dialog enhancement	“Dialog”	The main effect is a more prominent dialogue within the content.
6	General compression	“General”	A DRC effect that reduces the dynamic range and is applicable to multiple playback scenarios.
7	Dynamic expansion	“Expand”	Dynamics enhancement.
8	Artistic effect	“Artistic”	To create an artistic sound effect.
9	Clipping prevention	“Clipping”	The main purpose is clipping prevention.
10	Fade-in/fade-out	“Fade”	Fade-in and fade-out envelope for gapless content (applicable when not playing in Album mode). It has no dynamic compression.
11	Ducking other	“Duck other”	An effect that attenuates all audio content except for the channelGroup it is associated with. It has no dynamic compression.
12	Ducking self	“Duck self”	An effect that attenuates all channelGroups that it is associated with. It is identical to “Ducking other”, however, it has the same channelGroup assignment as regular DRC gains.
Remaining values are reserved			
NOTE: Any information on DRC sets with “Ducking” effect in this specification shall refer to both “Ducking other” and “Ducking self” if not stated otherwise.			

A.6.9 Coded metadata in loudnessInfo()

Table A.33 — Coding of bsSamplePeakLevel field

Encoding	Size	Mnemonic	samplePeakLevel in [dBFS]	Approximate range
μ	12 bits	uimsbf	$L_{sp} = \begin{cases} \text{"undefined"; if } \mu == 0 \\ L_{sp} = 20 - \mu 2^{-5}\text{;else} \end{cases}$	–107 ... 20, 0.0312 step size

Table A.34 — Coding of bsTruePeakLevel field (True Peak[4])

Encoding	Size	Mnemonic	truePeakLevel in [dBTP]	Approximate range
μ	12 bits	uimsbf	$L_{sp} = \begin{cases} \text{"undefined"; if } \mu == 0 \\ L_{sp} = 20 - \mu 2^{-5}\text{;else} \end{cases}$	–107 ... 20, 0.0312 step size

Table A.35 — Coding of methodValue field

Encoding	Format	methodDefinition	methodValue	Approx. range
μ	8 uimsbf	1, ..., 5	$L = -57.75 + \mu 2^{-2}$	−57.75 ... 6, 0.25 step size
See Table A.39	8 uimsbf	6	See Table A.39	0 ... 121
μ	5 uimsbf	7	$L = 80 + \mu$	80 ... 111 dB
μ	2 uimsbf	8	0x0: "not indicated" 0x1: "large room, X curve monitor"[1] 0x2: "small room, flat monitor"[1] 0x3: "reserved"	n/a
μ	8 uimsbf	9	$L = -116 + \mu 2^{-1}$	−116 ... 11.5 LKFS, 0.5 step size

Table A.36 — Coding of methodDefinition field in loudnessInfo()

Value	Value type	Meaning
0	n/a	Unknown/other
1	Loudness	program loudness (programLoudness as defined in ISO/IEC 23001-8)
2	Loudness	anchor loudness (anchorLoudness as defined in ISO/IEC 23001-8)
3	Loudness	maximum of the range, i.e. the 95th percentile of the loudness distribution according to EBU R-128[8],[7]
4	Loudness	maximum momentary loudness, measured using a 0.4s window according to ITU-R BS.1771-1[5] or EBU R-128[8],[6]
5	Loudness	maximum short-term loudness, measured using a 3s window according to ITU-R BS.1771–1[5] or EBU R-128[8],[6]
6	Loudness range	loudness range derived from EBU R-128[8],[7]
7	Sound pressure level	production mixing level measured according to[1]
8	Index	production room type according to[1]
9	Loudness	Short-term loudness, measured using a 3s window according to ITU-R BS.1771–1[5] or EBU R-128[8],[6] The 3s window shall include the current DRC frame.
Remaining values are reserved		

Table A.37 — Coding of measurementSystem field in loudnessInfo()

Value	Meaning
0	Unknown/other
1	EBU R-128[8]
2	ITU-R BS.1770–3[4]
3	ITU-R BS.1770–3 with pre-processing. The pre-processor is a 4th order Linkwitz-Riley filter with a cutoff frequency of 500 Hz.
4	User
5	Expert/panel
6	ITU-R BS.1771–1[5]
7 *(reserved, not permitted)*	Reserved Measurement System A (RMS_A)
8 *(reserved, not permitted)*	Reserved Measurement System B (RMS_B)

Table A.37 *(continued)*

Value	Meaning
9 *(reserved, not permitted)*	Reserved Measurement System C (RMS_C)
10 *(reserved, not permitted)*	Reserved Measurement System D (RMS_D)
11 *(reserved, not permitted)*	Reserved Measurement System E (RMS_E)
Remaining values are reserved	

Table A.38 — Coding of reliability field in loudnessInfo()

Value	Meaning
0	Reliability is unknown
1	Value is reported/imported but unverified
2	Value is a 'not to exceed' ceiling
3	Value is measured and accurate

Table A.39 — Coding algorithm for loudnessRange in dB

```
methodValue(loudnessRange) {
  if (loudnessRange < 0.0f)
                return 0;
        else if(loudnessRange <= 32.0f)
                return (UInt8)(4.0f*loudnessRange + 0.5f);
        else if(loudnessRange <= 70.0f)
                return (UInt8)(2.0f*(loudnessRange - 32.0f) + 0.5f) + 128;
        else if(loudnessRange < 121.0f)
                return (UInt8)(loudnessRange - 70.0f) + 0.5f) + 204;
        else
                return 255;
}
```

Table A.40 — Summary of all supported control parameters for loudness normalization supplied by host

Identifier	Default value	Unit	Description
targetLoudness	-	dB	Desired output loudness
albumMode	FALSE	Boolean	See 6.3.6
loudnessDeviationMax	63	dB	Permitted deviation from the exact loudness normalization
loudnessMeasurementMethod	Program Loudness	-	Requested method (see Table 26)
loudnessMeasurementSystem	ITU-R BS.1770–3	-	Requested system (see Table 27)
loudnessMeasurementPreProc	No pre-processing	-	Requested pre-processing (see Table 28)
deviceCutOffFrequency	20	Hz	Lower acoustic frequency limit of playback system response
loudnessNormalizationGainDbMax	∞	dB	Maximum permitted normalization gain
loudnessNormalizationGainModificationDb	0	dB	Normalization gain modification
outputPeakLevelMax	0, (6, if peakLimiterPresent)	dB	Maximum permitted peak value
peakLimiterPresent	FALSE	Boolean	Indicates if a peak limiter is present in the playback chain

Table A.41 — Summary of all supported control parameters for dynamic range compression supplied by host

Identifier	Default value	Unit	Description
dynamicRangeMeasurement	-	-	See Table 12
dynamicRangeMeasurementValue	-	dB	The requested value corresponding to dynamicRangeMeasurement
dynamicRangeMeasurementValueMin, dynamicRangeMeasurementValueMax	-	dB	The requested range corresponding to dynamicRangeMeasurement
drcCharacteristic	-	-	See Table E.4 and Table E.5
drcEffectType (desired/fallback)	-	-	See Table 11
compress	1	-	Scaling factor for negative DRC gains [dB]
boost	1	-	Scaling factor for positive DRC gains [dB]
drcCharacteristicTarget	0	-	Target DRC characteristic. See Table E.4 and Table E.5

Annex B
(normative)

External Interface to DRC tool

B.1 Description

This annex describes a normative decoder interface to the DRC tool for external parameter access in various applications. A non-exhaustive list of example use cases is listed below:

— Control of DRC tool from a higher system layer (e.g. all kinds of graphical user interfaces, which allow manipulating the DRC effect, the target loudness or others).

— Adjustment of DRC tool parameters to changed playback conditions (e.g. switch from loudspeaker to headphone playback).

— Remote control of DRC tool from a different device than it is implemented on.

— Adjustment of loudness normalization gain based on live measurements or other parameters available at the decoder.

The syntax and semantics of the interface are defined in B.2 and B.3, respectively. Decoders shall support (i.e. parse) received interface streams and normatively act on them. In local environments, the support of alternative interfaces is optionally permitted.

Although the interface is optional, it is recommended to use it whenever possible.

The syntax as defined in Table B.1 includes an optional interface signature, which can be used for various tasks. Some examples are listed below:

— Allow parameter changes only for authorized signatures.

— Allow hardware related parameter changes only to specific signatures.

— Allow loudness related parameter changes only to specific signatures.

— Define priorities for certain signatures if contradicting parameters are received.

NOTE It is expected that a registration authority will maintain a list of values of the uniDrcInterfaceSignatureDataType.

B.2 Syntax

Table B.1 — Syntax of uniDrcInterface() payload

Syntax	No. of bits	Mnemonic
uniDrcInterface()		
{		
uniDrcInterfaceSignaturePresent;	**1**	**bslbf**
if (uniDrcInterfaceSignaturePresent == 1) {		
uniDrcInterfaceSignatureType;	**8**	**uimsbf**
bsUniDrcInterfaceSignatureDataLength;	**8**	**uimsbf**
for (c = 0; c < bsUniDrcInterfaceSignatureDataLength+1; c++) {		

Table B.1 *(continued)*

Syntax	No. of bits	Mnemonic
uniDrcInterfaceSignatureData[c];	**8**	**uimsbf**
}		
}		
systemInterfacePresent;	**1**	**bslbf**
if (systemInterfacePresent == 1) {		
systemInterface();		
}		
loudnessNormalizationControlInterfacePresent;	**1**	**bslbf**
if (loudnessNormalizationControlInterfacePresent == 1) {		
loudnessNormalizationControlInterface();		
}		
loudnessNormalizationParameterInterfacePresent;	**1**	**bslbf**
if (loudnessNormalizationParameterInterfacePresent == 1) {		
loudnessNormalizationParameterInterface();		
}		
dynamicRangeControlInterfacePresent;	**1**	**bslbf**
If (dynamicRangeControlInterfacePresent == 1) {		
dynamicRangeControlInterface();		
}		
dynamicRangeControlParameterInterfacePresent;	**1**	**bslbf**
If (dynamicRangeControlParameterInterfacePresent == 1) {		
dynamicRangeControlParameterInterface();		
}		
uniDrcInterfaceExtensionPresent;	**1**	**bslbf**
if (uniDrcInterfaceExtensionPresent == 1) {		
uniDrcInterfaceExtension();		
}		
}		

Table B.2 — Syntax of systemInterface() payload

Syntax	No. of bits	Mnemonic
systemInterface()		
{		
targetConfigRequestType;	**2**	**uimsbf**
if (targetConfigRequestType == 0) {		
numDownmixIdRequests;	**4**	**uimsbf**
for (d = 0; d < numDownmixIdRequests; d++) {		
downmixIdRequested[d];	**7**	**uimsbf**
}		
} else if (targetConfigRequestType == 1) {		
targetLayoutRequested;	**8**	**uimsbf**
} else if (targetConfigRequestType == 2) {		
bsTargetChannelCountRequested;	**7**	**uimsbf**
}		
}		

Table B.3 — Syntax of loudnessNormalizationControlInterface() payload

Syntax	No. of bits	Mnemonic
loudnessNormalizationControlInterface()		
{		
loudnessNormalizationOn;	**1**	**bslbf**
if (loudnessNormalizationOn == 1) {		
targetLoudness;	**12**	**uimsbf**
}		
}		

Table B.4 — Syntax of loudnessNormalizationParameterInterface() payload

Syntax	No. of bits	Mnemonic
loudnessNormalizationParameterInterface()		
{		
albumMode;	**1**	**bslbf**
peakLimiterPresent;	**1**	**bslbf**
changeLoudnessDeviationMax;	**1**	**bslbf**
if (changeLoudnessDeviationMax == 1) {		
loudnessDeviationMax;	**6**	**uimsbf**
}		
changeLoudnessMeasurementMethod;	**1**	**bslbf**
if (changeLoudnessMeasurementMethod == 1) {		
loudnessMeasurementMethod;	**3**	**uimsbf**
}		

Table B.4 *(continued)*

Syntax	No. of bits	Mnemonic
changeLoudnessMeasurementSystem;	**1**	**bslbf**
if (changeLoudnessMeasurementSystem == 1) {		
loudnessMeasurementSystem;	**4**	**uimsbf**
}		
changeLoudnessMeasurementPreProc;	**1**	**bslbf**
if (changeLoudnessMeasurementPreProc == 1) {		
loudnessMeasurementPreProc;	**2**	**uimsbf**
}		
changeDeviceCutOffFrequency;	**1**	**bslbf**
if (changeDeviceCutOffFrequency == 1) {		
deviceCutOffFrequency;	**6**	**uimsbf**
}		
changeLoudnessNormalizationGainDbMax;	**1**	**bslbf**
if (changeLoudnessNormalizationGainDbMax == 1) {		
loudnessNormalizationGainDbMax;	**6**	**uimsbf**
}		
changeLoudnessNormalizationGainModificationDb;	**1**	**bslbf**
if (changeLoudnessNormalizationGainModificationDb == 1) {		
loudnessNormalizationGainModificationDb;	**10**	**uimsbf**
}		
changeOutputPeakLevelMax;	**1**	**bslbf**
if (changeOutputPeakLevelMax == 1) {		
outputPeakLevelMax;	**6**	**uimsbf**
}		
}		

Table B.5 — Syntax of dynamicRangeControlInterface() payload

Syntax	No. of bits	Mnemonic
dynamicRangeControlInterface()		
{		
dynamicRangeControlOn;	**1**	**bslbf**
if (dynamicRangeControlOn == 1) {		
numDrcFeatureRequests;	**3**	**uimsbf**
for (i = 0; i < numDrcFeatureRequests; i++) {		
drcFeatureRequestType[i];	**2**	**uimsbf**
switch (drcFeatureRequestType[i]) {		
case 0:		
numDrcEffectTypeRequests;	**4**	**uimsbf**
numDrcEffectTypeRequestsDesired;	**4**	**uimsbf**
for (j = 0; j < numDrcEffectTypeRequests; j++) {		
drcEffectTypeRequest[i][j];	**4**	**uimsbf**

Table B.5 *(continued)*

Syntax	No. of bits	Mnemonic
}		
break;		
case 1:		
dynRangeMeasurementRequestType[i];	**2**	**uimsbf**
dynRangeRequestedIsRange[i];	**1**	**bslbf**
if (dynRangeRequestedIsRange[i] == 0) {		
dynamicRangeRequestValue;	**8**	**uimsbf**
} else {		
dynamicRangeRequestValueMin[i];	**8**	**uimsbf**
dynamicRangeRequestValueMax[i];	**8**	**uimsbf**
}		
break;		
case 2:		
drcCharacteristicRequest[i];	**7**	**uimsbf**
break;		
}		
}		
}		
}		

Table B.6 — Syntax of dynamicRangeControlParameterInterface() payload

Syntax	No. of bits	Mnemonic
dynamicRangeControlParameterInterface()		
{		
changeCompress;	**1**	**bslbf**
changeBoost;	**1**	**bslbf**
if (changeCompress == 1) {		
compress;	**8**	**uimsbf**
}		
if (changeBoost == 1) {		
boost;	**8**	**uimsbf**
}		
changeDrcCharacteristicTarget;	**1**	**bslbf**
if (changeDrcCharacteristicTarget == 1) {		
drcCharacteristicTarget;	**8**	**uimsbf**
}		
}		

Table B.7 — Syntax of uniDrcInterfaceExtension() payload

Syntax	No. of bits	Mnemonic
uniDrcInterfaceExtension()		
{		
while (**uniDrcInterfaceExtType**!=UNIDRCINTERFACEEXT_TERM) {	**4**	**uimsbf**
extSizeBits = **bitSizeLen** + 4;	**4**	**uimsbf**
extBitSize = **bitSize** + 1;	**extSizeBits**	**uimsbf**
switch (uniDrcInterfaceExtType) {		
/* add future extensions here */		
default:		
for (i=0; i<extBitSize; i++) {		
otherBit;	**1**	**bslbf**
}		
}		
}		
}		

B.3 Semantics

B.3.1 Semantics of uniDrcInterface()

uniDrcInterfaceSignatureDataType This field defines the type of signature. The following values are possible:

Table B.8 — Coding of uniDrcInterfaceSignatureDataType field

Value	Meaning
0	Generic String in UTF-8 according to ISO/IEC 10646
1–127	Reserved for ISO use
128–255	Reserved for use outside of ISO scope

bsUniDrcInterfaceSignatureDataLength This field defines the length of the following interface signature in Bytes minus one.

uniDrcInterfaceSignatureData This field defines a signature, which, e.g. contains the originator of the interface data if uniDrcInterfaceSignatureDataType is set to 0.

B.3.2 Semantics of systemInterface()

targetConfigRequestType This field contains the requested target configuration type according to Table B.9.

Table B.9 — Coding of targetConfigRequestType field

Value	Meaning
0	downmixId or downmix ID list request
1	target layout request
2	target channel count request
3	*reserved*

numDownmixIdRequests This field contains the number of downmixId requests. The field can take values between 0 and 15.

downmixIdRequested This field contains a requested downmixId.

targetLayoutRequested This field contains a requested target layout according to ISO/IEC 23001-8.

bsTargetChannelCountRequested This field contains a requested target channel count. The values are encoded according to Table B.10.

Table B.10 — Coding of bsTargetChannelCountRequested field

Encoding	Size	Mnemonic	targetChannelCountRequested	Range
μ	7 bits	uimsbf	$N_{CH} = \mu+1$	1...128

B.3.3 Semantics of loudnessNormalizationControlInterface()

loudnessNormalizationOn This flag signals if loudness normalization processing should be switched on or off. If loudnessNormalizationOn == 0, loudnessNormalizationGainDb shall be set to 0 dB.

targetLoudness This field contains the desired output loudness. The values are encoded according to Table B.11.

Table B.11 — Coding of targetLoudness field

Encoding	Size	Mnemonic	Value in [LKFS]	Approximate range
μ	12 bits	uimsbf	$L_{TL} = -\mu 2^{-5}$	–128 ... 0 dB, 0.0312 dB step size

B.3.4 Semantics of loudnessNormalizationParameterInterface()

albumMode This flag signals if the playback system is in album mode, which means that successive songs of an album are played back. In album mode, the same loudness normalization is done for all songs/items of an album and not independently for each single song/item. The default value is FALSE.

peakLimiterPresent This flag signals if a peak limiter is present at the end of the processing chain. This field may be also used to signal that an available peak limiter is switched on or off. The default value is FALSE.

changeLoudnessDeviationMax This flag signals that the default value of loudnessDeviationMax should be changed.

loudnessDeviationMax This field contains a changed value for loudnessDeviationMax. The values are encoded according to Table B.12. The default value is 63 dB.

Table B.12 — Coding of loudnessDeviationMax field

Encoding	Size	Mnemonic	Value in [LKFS]	Range
μ	6 bits	uimsbf	loudnessDeviationMax = μ	0 ... 63 dB, 1 dB step size

changeLoudnessMeasurementMethod This flag signals that the default value of loudnessMeasurementMethod should be changed.

loudnessMeasurementMethod This field contains a changed loudness measurement method according to Table 26. The default value is 0.

changeLoudnessMeasurementSystem This flag signals that the default value of loudnessMeasurementSystem should be changed.

loudnessMeasurementSystem This field contains a changed loudness measurement system according to Table 27. The default value is 0.

changeLoudnessMeasurementPreProc This flag signals that the default value of loudnessMeasurementPreProc should be changed.

loudnessMeasurementPreProc This field contains a changed loudness measurement pre-processing according to Table 28. The default value is 0.

changeDeviceCutOffFrequency This flag signals that the default value of deviceCutOffFrequency should be changed.

deviceCutOffFrequency This field contains a changed deviceCutOffFrequency according to Table B.13. The default value is 20 Hz. This value is limited to [20, 500] and it will be used to interpolate loudness values between loudness measurement pre-processing type 0 and 1.

Table B.13 — Coding of deviceCutOffFrequency field

Encoding	Size	Mnemonic	Value in [Hz]	Range
μ	6 bits	uimsbf	$\text{deviceCutOffFreq} = \max(\min(\mu 10, 500), 20)$	20 ... 500 Hz, 10 Hz step size

changeLoudnessNormalization-GainDbMax This flag signals that the default value of loudnessNormalization-GainDbMax should be changed.

loudnessNormalizationGainDbMax This field contains a changed value for loudnessNormalization-GainDbMax. The values are encoded according to Table B.14. The default value is plus infinity.

Table B.14 — Coding of loudnessNormalizationGainDbMax field

Encoding	Size	Mnemonic	Value in [dB]	Range
μ	6 bits	uimsbf	$\text{loudNormGainDbMax} = \begin{cases} \mu 2^{-1}; \text{ if } \mu < 63 \\ \infty; \text{ else} \end{cases}$	0 ... +inf dB, 0.5 dB step size

changeLoudnessNormalization-GainModificationDb This flag signals that the default value of loudnessNormalization-GainModificationDb should be changed.

loudnessNormalizationGainModificationDb This field contains a changed value for loudnessNormalization-GainModificationDb. The values are encoded according to Table B.15. The default value is 0 dB.

Table B.15 — Coding of loudnessNormalizationGainModificationDb field

Encoding	Size	Mnemonic	Value in [dB]	Approximate range
μ	10 bits	uimsbf	$\text{loudNormGainModDb} = -16 + \mu 2^{-5}$	−16 ... 16 dB, 0.0312 dB step size

changeOutputPeakLevelMax This flag signals that the default value of outputPeakLevelMax should be changed.

outputPeakLevelMax This field contains a changed value for outputPeakLevelMax. The values are encoded according to Table B.16. The default value is 0 dB. If peakLimiterPresent = = 1, the default value is 6 dB.

Table B.16 — Coding of outputPeakLevelMax field

Encoding	Size	Mnemonic	Value in [dBFS]	Range
μ	6 bits	uimsbf	$\text{outputPeakLevelMax} = \mu 2^{-1}$	0 ... 31.5 dBFS, 0.5 dB step size

B.3.5 Semantics of dynamicRangeControlInterface()

dynamicRangeControlOn	This flag signals if dynamic range control processing should be switched on or off. If dynamicRangeControlOn == 0, any selected DRC set shall not be applied except for fading or ducking.
numDrcFeatureRequests	This field contains the number of feature requests to the dynamic range control system. The field can take values between 0 and 7.
drcFeatureRequestType	This field contains the requested dynamic range control feature type according to Table 10.
numDrcEffectTypeRequests	This field contains the total number of requested dynamic range control effect types. The field can take values between 0 and 15.
numDrcEffectTypeRequestsDesired	This field contains the number of requested dynamic range control effect types, which are desired and not fallback requests. The field can take values between 0 and 15.
drcEffectTypeRequest	This field contains the requested dynamic range control effect type according to Table 11.
dynRangeMeasurementRequestType	This field contains the requested dynamic range measurement type according to Table 12.
dynRangeRequestedIsRange	This flag signals if the requested dynamic range is specified as range (0) or as single value (1).
dynamicRangeRequestValue	This field contains a requested dynamic range value. The values are encoded according to methodDefinition #6 in Table A.35.
dynamicRangeRequestValueMin	This field contains the lower boundary of a requested dynamic range. The values are encoded according to methodDefinition #6 in Table A.35.
dynamicRangeRequestValueMax	This field contains the upper boundary of a requested dynamic range. The values are encoded according to methodDefinition #6 in Table A.35.
drcCharacteristicRequest	This field contains the requested dynamic range control characteristic used on encoder side according to ISO/IEC 23001-8.

B.3.6 Semantics of dynamicRangeControlParameterInterface()

changeCompress This flag signals that the default value for compress should be changed.

compress This field contains a scaling factor for negative DRC gains [dB]. The default value is 1. The values are encoded according to Table B.17.

changeBoost This flag signals that the default value for boost should be changed.

boost This field contains a scaling factor for positive DRC gains [dB]. The default value is 1. The values are encoded according to Table B.17.

Table B.17 — Coding of compress and boost field

Encoding	Size	Mnemonic	Value	Approximate range
μ	8 bits	uimsbf	$\text{scalingFactor} = \begin{cases} 1-\mu 2^{-8}; \text{if } \mu < 255 \\ 0.0; \text{else} \end{cases}$	0.0 ... 1.0, 0.0039 step size

changeDrcCharacteristicTarget This flag signals that the default value for drcCharacteristicTarget should be changed.

drcCharacteristicTarget This field contains a target DRC characteristic index, which can be used for the mapping of DRC gains (see 6.4.6**). T**he default value is 0 (no mapping). The values are encoded according to ISO/IEC 23001-8.

B.3.7 Semantics of uniDrcInterfaceExtension()

Table B.18 — UniDrc interface extension types

Symbol	Value of uniDrcInterfaceExtType	Purpose
UNIDRCINTERFACEEXT_TERM	0x0	Termination tag
(reserved)	*(All remaining values)*	For future use

Annex C
(informative)

Audio codec specific information

C.1 Introduction

If applicable, audio codec specific information for the DRC tool is included in the corresponding audio codec standard. For example, this information can include the location of the DRC payload and any configuration that differs from the default. The following summarizes as much codec specific information as was available during the editing period of this standard. Please refer to the latest codec standards for up-to-date specifications.

C.2 AAC

C.2.1 DRC metadata extension for AAC

For AAC (ISO/IEC 14496-3) the DRC data uniDrc() is carried in an extension payload in a Fill Element. The extension type ID is given in Table C.1.

Table C.1 — Definition of new extension_type for AAC

Symbol	Value of extension type (binary)	Purpose
EXT_UNIDRC	0100	Unified DRC

C.2.2 Delay mode for AAC

AAC uses the default delay mode.

C.2.3 DRC frame size and time resolution for AAC

The DRC frame size has the default size, i.e. it has the same time duration as the AAC frame size.

The value of *deltaTmin* in number of samples at the audio rate is calculated as specified in 6.4.3. Specific values are provided here for convenience based on the following formula:

$$\text{deltaTmin} = 2^M \tag{C.1}$$

The applicable exponent M is found by looking up the audio sample rate range that fulfils:

$$f_{s,\min} \le f_s \le f_{s,\max} \tag{C.2}$$