INTERNATIONAL STANDARD

ISO/IEC 23008-3

Second edition
2019-02-20

AMENDMENT 1
2019-06

Information technology — High efficiency coding and media delivery in heterogeneous environments —

Part 3: 3D audio

AMENDMENT 1: Audio metadata enhancements

Technologies de l'information — Codage à haute efficacité et livraison des medias dans des environnements hétérogènes —

Partie 3: Audio 3D

AMENDEMENT 1: Améliorations de la prise en charge des métadonnées audio

Reference number
ISO/IEC 23008-3:2019/Amd.1:2019(E)

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

Published in Switzerland

Foreword

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

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of ISO documents 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 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 of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see: www.iso.org/iso/foreword.html.

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

A list of all parts in the ISO/IEC 23008 series can be found on the ISO website.

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

Information technology — High efficiency coding and media delivery in heterogeneous environments —

Part 3:
3D audio

AMENDMENT 1: Audio metadata enhancements

5.2.2.1 General configuration syntax

In subclause 5.2.2.1 replace Table 14 with:

Table 14 — Syntax of Signals3d()

Syntax	No. of bits	Mnemonic

Signals3d()
{
numAudioChannels = 0;
numAudioObjects = 0;
numSAOCTransportChannels = 0;
numHOATransportChannels = 0;
 	
bsNumSignalGroups;
	5	**uimsbf**

for (grp = 0; grp < bsNumSignalGroups + 1 ; grp++) {
signal_groupID[grp] = grp;
differsFromReferenceLayout[grp] = 0;
signalGroupType[grp];
	3	**bslbf**

bsNumberOfSignals[grp] = escapedValue(5, 8, 16);
if (SignalGroupType[grp] == SignalGroupTypeChannels) {
numAudioChannels += bsNumberOfSignals[grp] + 1;
differsFromReferenceLayout[grp];
	1	**bslbf**

if(differsFromReferenceLayout[grp]) {
audioChannelLayout[grp] = SpeakerConfig3d();
}
else {
audioChannelLayout[grp] = referenceLayout;
}
}
if (SignalGroupType[grp] == SignalGroupTypeObject) {
numAudioObjects += bsNumberOfSignals[grp] + 1;
}
if (SignalGroupType[grp] == SignalGroupTypeSAOC) {
numSAOCTransportChannels += bsNumberOfSignals[grp] + 1;		

Table 14 *(continued)*

Syntax	No. of bits	Mnemonic
saocDmxLayoutPresent;	**1**	**bslbf**
if (saocDmxLayoutPresent == 1) {		
saocDmxChannelLayout = SpeakerConfig3d();		
}		
}		
if (SignalGroupType[grp] == SignalGroupTypeHOA) {		
numHOATransportChannels += bsNumberOfSignals[grp] + 1;		
}		
}		
}		

5.2.2.3 Core decoder configuration

In 5.2.2.3 replace Table 23 with:

Table 23 — Syntax of mpegh3daExtElementConfig()

Syntax	No. of bits	Mnemonic
mpegh3daExtElementConfig()		
{		
usacExtElementType = escapedValue(4, 8, 16);		
usacExtElementConfigLength = escapedValue(4, 8, 16);		
if (**usacExtElementDefaultLengthPresent**) {	**1**	**uimsbf**
usacExtElementDefaultLength = escapedValue(8, 16, 0) + 1;		
} else {		
usacExtElementDefaultLength = 0;		
}		
usacExtElementPayloadFrag;	**1**	**uimsbf**
switch (usacExtElementType) {		
case ID_EXT_ELE_FILL:		
/* No configuration element */		
break;		
case ID_EXT_ELE_MPEGS:		
SpatialSpecificConfig();		
break;		
case ID_EXT_ELE_SAOC:		
SAOCSpecificConfig();		
break;		
case ID_EXT_ELE_AUDIOPREROLL:		
/* No configuration element */		

[a] The default entry for the usacExtElementType is used for unknown extElementTypes so that legacy decoders can cope with future extensions.

Table 23 *(continued)*

Syntax	No. of bits	Mnemonic
break;		
case ID_EXT_ELE_UNI_DRC:		
mpegh3daUniDrcConfig();		
break;		
case ID_EXT_ELE_OBJ_METADATA:		
ObjectMetadataConfig();		
break;		
case ID_EXT_ELE_SAOC_3D:		
SAOC3DSpecificConfig();		
break;		
case ID_EXT_ELE_HOA:		
HOAConfig();		
break;		
case ID_EXT_ELE_FMT_CNVRTR		
/* No configuration element */		
break;		
case ID_EXT_ELE_MCT:		
MCTConfig();		
break;		
case ID_EXT_ELE_TCC:		
TccConfig();		
break;		
case ID_EXT_ELE_HOA_ENH_LAYER:		
HOAEnhConfig();		
break;		
case ID_EXT_ELE_HREP:		
HREPConfig(current_signal_group);		
break;		
case ID_EXT_ELE_ENHANCED_OBJ_METADATA:		
EnhancedObjectMetadataConfig();		
break;		
case ID_EXT_ELE_PROD_METADATA:		
prodMetadataConfig();		
break;		
default:	[a]	
while (usacExtElementConfigLength--) {		
tmp;	**8**	**uimsbf**
}		
break;		
}		
}		

[a] The default entry for the usacExtElementType is used for unknown extElementTypes so that legacy decoders can cope with future extensions.

5.3.4 Core decoder configuration data elements

In 5.3.4 replace Table 75 with:

Table 75 — Value of usacExtElementType

usacExtElementType	Value
ID_EXT_ELE_FILL	0
ID_EXT_ELE_MPEGS	1
ID_EXT_ELE_SAOC	2
ID_EXT_ELE_AUDIOPREROLL	3
ID_EXT_ELE_UNI_DRC	4
ID_EXT_ELE_OBJ_METADATA	5
ID_EXT_ELE_SAOC_3D	6
ID_EXT_ELE_HOA	7
ID_EXT_ELE_FMT_CNVRTR	8
ID_EXT_ELE_MCT	9
ID_EXT_ELE_TCC	10
ID_EXT_ELE_HOA_ENH_LAYER	11
ID_EXT_ELE_HREP	12
ID_EXT_ELE_ENHANCED_OBJ_METADATA	13
ID_EXT_ELE_PROD_METADATA	14
/* reserved for ISO use */	15-127
/* reserved for use outside of ISO scope */	128 and higher
NOTE Application-specific usacExtElementType values are mandated to be in the space reserved for use outside of ISO scope. These are skipped by a decoder as a minimum of structure is required by the decoder to skip these extensions.	

In 5.3.4 replace Table 76 with:

Table 76 — Interpretation of data blocks for extension payload decoding

usacExtElementType	The concatenated usacExtElementSegmentData represents:
ID_EXT_ELE_FILL	Series of **fill_byte**
ID_EXT_ELE_MPEGS	SpatialFrame() as defined in ISO/IEC 23003-1
ID_EXT_ELE_SAOC	SAOCFrame() as defined in ISO/IEC 23003-2
ID_EXT_ELE_AUDIOPREROLL	AudioPreRoll()
ID_EXT_ELE_UNI_DRC	uniDrcGain() as defined in ISO/IEC 23003-4
ID_EXT_ELE_OBJ_METADATA	objectMetadataFrame()
ID_EXT_ELE_SAOC_3D	Saoc3DFrame()
ID_EXT_ELE_HOA	HOAFrame()
ID_EXT_ELE_FMT_CNVRTR	FormatConverterFrame()
ID_EXT_ELE_MCT	MultichannelCodingFrame()
ID_EXT_ELE_TCC	TccGroupOfSegments()
ID_EXT_ELE_HOA_ENH_LAYER	HOAEnhFrame()
ID_EXT_ELE_HREP	HREPFrame(outputFrameLength, current_signal_group)
ID_EXT_ELE_ENHANCED_OBJ_METADATA	EnhancedObjectMetadataFrame()

Table 76 *(continued)*

usacExtElementType	The concatenated usacExtElementSegmentData represents:
ID_EXT_ELE_PROD_METADATA	prodMetadataFrame()
unknown	Unknown data. The data block shall be discarded.

12.2.1 Configuration of HOA elements

In subclause 12.2.1 replace Table 188 with:

Table 188 — Syntax of HOADecoderConfig()

Syntax	No. of bits	Mnemonic
HOADecoderConfig(numHOATransportChannels)		
{		
MinAmbHoaOrder = escapedValue(3,5,0) – 1;	**3,8**	**uimsbf**
MinNumOfCoeffsForAmbHOA = (MinAmbHoaOrder + 1)^2;		
NumOfAdditionalCoders = numHOATransportChannels – MinNumOfCoeffsForAmbHOA;		
NumLayers = 1;		
NumHOAChannelsLayer[0] = numHOATransportChannels;		
if(**SingleLayer** == 0){	**1**	**bslbf**
HOALayerChBits = ceil(log2(NumOfAdditionalCoders));		
NumHOAChannelsLayer[0] = **codedLayerCh** + MinNumOfCoeffsForAmbHOA;	**HOALayerChBits**	**uimsbf**
remainingCh = numHOATransportChannels – NumHOAChannelsLayer[0];		
while (remainingCh>1) {		
HOALayerChBits = ceil(log2(remainingCh));		
NumHOAChannelsLayer[NumLayers] = NumHOAChannelsLayer[NumLayers-1] + **codedLayerCh** + 1;	**HOALayerChBits**	**uimsbf**
remainingCh = numHOATransportChannels – NumHOAChannelsLayer[NumLayers];		
NumLayers++;		
}		
if (remainingCh) {		
NumHOAChannelsLayer[NumLayers] = numHOATransportChannels;		
NumLayers++;		
}		
}		
CodedSpatialInterpolationTime;	**3**	**uimsbf**
SpatialInterpolationMethod;	**1**	**bslbf**
NOTE MinAmbHoaOrder = 30 … 37 are reserved. HOAFrameLengthIndicator = 3 is reserved. CodedVVecLength = 3 is reserved.		

Table 188 *(continued)*

Syntax	No. of bits	Mnemonic
CodedVVecLength;	**2**	**uimsbf**
MaxGainCorrAmpExp;	**3**	**uimsbf**
HOAFrameLengthIndicator;	**2**	**uimsbf**
if(MinAmbHoaOrder < HoaOrder) {		
DiffOrderBits = ceil(log2(HoaOrder- MinAmbHoaOrder+1))		
MaxHoaOrderToBeTransmitted = **DiffOrder** + MinAmbHoaOrder;	**DiffOrderBits**	**uimsbf**
}		
else {		
MaxHoaOrderToBeTransmitted = HoaOrder;		
}		
MaxNumOfCoeffsToBeTransmitted = (MaxHoaOrderToBeTransmitted + 1)^2;		
MaxNumAddActiveAmbCoeffs = MaxNumOfCoeffsToBeTransmitted - MinNumOfCoeffsForAmbHOA;		
VqConfBits = ceil (log2(ceil(log2(NumOfHoaCoeffs+1))));		
NumVVecVqElementsBits;	**VqConfBits**	**uimsbf**
if(MinAmbHoaOrder == 1) {		
UsePhaseShiftDecorr;	**1**	**bslbf**
}		
if(SingleLayer==1) {		
HOADecoderEnhConfig();		
}		
AmbAsignmBits = ceil(log2(MaxNumAddActiveAmbCoeffs));		
ActivePredIdsBits = ceil(log2(NumOfHoaCoeffs));		
i = 1;		
while(i * ActivePredIdsBits + ceil(log2(i)) < NumOfHoaCoeffs){		
i++;		
}		
NumActivePredIdsBits = ceil(log2(max(1, i – 1)));		
GainCorrPrevAmpExpBits = ceil(log2(ceil(log2(1.5 * NumOfHoaCoeffs)) + MaxGainCorrAmpExp + 1));		
for (i=0; i<NumOfAdditionalCoders; ++i){		
AmbCoeffTransitionState[i] = 3;		
}		
}		
NOTE MinAmbHoaOrder = 30 … 37 are reserved. HOAFrameLengthIndicator = 3 is reserved. CodedVVecLength = 3 is reserved.		

14.2.1 Main MHAS syntax elements

In 14.2.1 replace Table 220 with:

Table 220 — Syntax of MHASPacketPayload()

Syntax	No. of bits	Mnemonic
MHASPacketPayload(MHASPacketType)		
{		
switch (MHASPacketType) {		
case PACTYP_SYNC:		
0xA5; /* syncword*/	**8**	**uimsbf**
break;		
case PACTYP_MPEGH3DACFG:		
mpegh3daConfig();		
break;		
case PACTYP_MPEGH3DAFRAME:		
mpegh3daFrame();		
break;		
case PACTYP_AUDIOSCENEINFO:		
mae_AudioSceneInfo();		
break;		
case PACTYP_FILLDATA:		
for (i=0; i< MHASPacketLength; i++) {		
mhas_fill_data_byte(i);	**8**	**bslbf**
}		
break;		
case PACTYP_SYNCGAP:		
syncSpacingLength = escapedValue(16,24,24);	**16,40,64**	**uimsbf**
break;		
case PACTYP_MARKER:		
for (i=0; i< MHASPacketLength; i++) {		
marker_byte(i);	**8**	**bslbf**
}		
break;		
case PACTYP_CRC16:		
mhasParity16Data;	**16**	**bslbf**
break;		
case PACTYP_CRC32:		
mhasParity32Data;	**32**	**bslbf**
break;		
case PACTYP_GLOBAL_CRC16:		
global_CRC_type;	**2**	**bslbf**
numProtectedPackets;	**6**	**bslbf**
mhasParity16Data;	**16**	**bslbf**
break;		

Table 220 *(continued)*

Syntax	No. of bits	Mnemonic
case PACTYP_ GLOBAL_CRC32:		
global_CRC_type;	**2**	**bslbf**
numProtectedPackets;	**6**	**bslbf**
mhasParity32Data;	**32**	**bslbf**
break;		
case PACTYP_DESCRIPTOR:		
for (i=0; i< MHASPacketLength; i++) {		
mhas_descriptor_data_byte(i);	**8**	**bslbf**
}		
break;		
case PACTYP_USERINTERACTION:		
mpegh3daElementInteraction();		
break;		
case PACTYP_LOUDNESS_DRC:		
mpegh3daLoudnessDrcInterface();		
break;		
case PACTYP_BUFFERINFO:		
mhas_buffer_fullness_present	**1**	**uimsbf**
if (mhas_buffer_fullness_present)		
mhas_buffer_fullness = escapedValue(15,24,32);	**15,39,71**	**uimsbf**
}		
break;		
case PACTYP_AUDIOTRUNCATION:		
audioTruncationInfo();		
break;		
case PACTYP_GENDATA:		
GenDataPayload();		
break;		
case PACTYP_EARCON:		
earconInfo();		
break;		
case PACTYP_PCMCONFIG:		
pcmDataConfig();		
break;		
case PACTYP_PCMDATA:		
pcmDataPayload();		
break;		
case PACTYP_LOUDNESS:		
mpegh3daLoudnessInfoSet();		
break;		
}		
ByteAlign();		
}		

14.3.1 mpeghAudioStreamPacket()

In 14.3.1 replace Table 223 with:

Table 223 — Value of MHASPacketType

MHASPacketType	Value
PACTYP_FILLDATA	0
PACTYP_MPEGH3DACFG	1
PACTYP_MPEGH3DAFRAME	2
PACTYP_AUDIOSCENEINFO	3
/ reserved for ISO use */*	4-5
PACTYP_SYNC	6
PACTYP_SYNCGAP	7
PACTYP_MARKER	8
PACTYP_CRC16	9
PACTYP_CRC32	10
PACTYP_DESCRIPTOR	11
PACTYP_USERINTERACTION	12
PACTYP_LOUDNESS_DRC	13
PACTYP_BUFFERINFO	14
PACTYP_GLOBAL_CRC16	15
PACTYP_GLOBAL_CRC32	16
PACTYP_AUDIOTRUNCATION	17
PACTYP_GENDATA	18
PACTYP_EARCON	19
PACTYP_PCMCONFIG	20
PACTYP_PCMDATA	21
PACTYP_LOUDNESS	22
/ reserved for ISO use */*	23-127
/ reserved for use outside of ISO scope */*	128-261
/ reserved for ISO use */*	262-389
/ reserved for use outside of ISO scope */*	390-517

NOTE Application-specific MHASPacketType values are mandated to be in the space reserved for use outside of ISO scope. These are skipped by a decoder as a minimum of structure is required by the decoder to skip these extensions.

14.3.2 MHASPacketPayload()

At the end of subclause 14.3.2 add:

earconInfo() Earcon Info structure as defined in 28.2.

pcmDataConfig() PCM data configuration structure as defined in 28.2.

pcmDataPayload() PCM data payload structure as defined in 28.2.

mpegh3daLoudnessInfoSet() Loudness metadata structure as defined in 6.3.1.

14.4 Description of MHASPacketTypes

At the end of subclause 14.4 add:

14.4.15 PACTYP_EARCON

The MHASPacketType PACTYP_EARCON may be used to embed information about the earcons available in the earconInfo() structure and to feed earcon info data in the form of the earconInfo() structure to the decoder.

If the earconInfo() structure contains at least one earcon of type PCM (i.e. earconType == 5) the MHAS stream shall contain at least one MHAS packet of type PACTYP_PCMCONFIG and at least one MHAS packet of type PACTYP_PCMDATA.

14.4.16 PACTYP_PCMCONFIG

The MHASPacketType PACTYP_PCMCONFIG may be used to carry configuration information for PCM payload data and to feed the PCM data configuration information in the form of the pcmDataConfig() structure to the decoder.

If an MHASPacketType PACTYP_PCMCONFIG is present after an MHASPacketType PACTYP_EARCON, the pcmDataConfig() structure shall be used together with the previous earconInfo() structure. If no MHASPacketType PACTYP_EARCON is present in the stream the pcmDataConfig() structure shall be ignored.

14.4.17 PACTYP_PCMDATA

The MHASPacketType PACTYP_PCMDATA may be used to embed PCM payload data corresponding to the PCM signals defined in the pcmDataConfig() structure and to feed the PCM data in the form of the pcmDataPayload() structure to the decoder.

If an MHASPacketType PACTYP_ PCMDATA is present after an MHASPacketType PACTYP_ PCMCONFIG, the pcmDataPayload() structure shall be used together with the previous earconInfo() and pcmDataConfig() structures. If no MHASPacketType PACTYP_EARCON and MHASPacketType PACTYP_ PCMCONFIG are present in the stream the pcmDataPayload() structure shall be ignored.

14.4.18 PACTYP_LOUDNESS

The MHASPacketType PACTYP_LOUDNESS may be used to embed loudness metadata as defined in the mpegh3daLoudnessInfoSet() structure. If present and supported by a decoder, it shall take precedence over the in-stream loudness information conveyed via mpegh3daConfigExtension() as defined in Table 24.

If present, the MHASPacketType PACTYP_LOUDNESS shall follow PACTYP_MPEGH3DACFG for each random access point and stream access point.

Updated loudness information may be available for instance after editing. The MHASPacketType PACTYP_LOUDNESS can be used to convey the updated loudness information to the decoder without requiring an update of the audio stream.

17.10.3.1 General

In subclause 17.10.3.1, extend paragraphs by:

— Enhanced object metadata;

  — diffuseness;

  — divergence and divergence azimuth range;

  — exclusion sector metadata;

— Production Metadata.

17.10.3.2 Syntax of an interface for object-based metadata

In 17.10.3.2 replace Table 265 with:

Table 265 — Syntax of mpegh3da_getObjectAudioAndMetadata()

Syntax	No. of bits	Mnemonic
mpegh3da_getObjectAudioAndMetadata()		
{		
/* FRAME CONFIGURATION */		
goa_frameLength;	**6**	**uimsbf**
goa_audioTruncation;	**2**	**bslbf**
if (goa_audioTruncation>0) {		
goa_numSamples;	**13**	**uimsbf**
} else {		
goa_numSamples = goa_frameLength << 6;		
}		
/* OBJECT METADATA */		
goa_numberOfOutputObjects;	**9**	**uimsbf**
for (o = 0; o < goa_numberOfOutputObjects; o++) {		
goa_elementID[o];	**9**	**uimsbf**
goa_hasDynamicObjectPriority[o];	**1**	**bslbf**
goa_hasUniformSpread[o];	**1**	**bslbf**
/* OAM Data */		
goa_numOAMframes[o]	**6**	**uimsbf**
for (nf = 0; nf < goa_numOAMframes[o]; nf++) {		
goa_objectMetadataPresent;	**1**	**bslbf**
if (goa_objectMetadataPresent==1) {		
goa_positionAzimuth[o][nf];	**8**	**uimsbf**
goa_positionElevation[o][nf];	**6**	**uimsbf**
goa_positionRadius[o][nf];	**4**	**uimsbf**
goa_objectGainFactor[o][nf];	**7**	**uimsbf**
if (goa_hasDynamicObjectPriority[o]) {		
goa_dynamicObjectPriority[o][nf];	**3**	**uimsbf**
}		
if (goa_hasUniformSpread[o]) {		
goa_uniformSpread[o][nf];	**7**	**uimsbf**
} else {		
goa_spreadWidth[o][nf];	**7**	**uimsbf**
goa_spreadHeight[o][nf];	**5**	**uimsbf**
goa_spreadDepth[o][nf];	**4**	**uimsbf**

Table 265 *(continued)*

Syntax	No. of bits	Mnemonic
} } }		
/* Signal group related data */		
goa_fixedPosition[o];	**1**	**bslbf**
goa_groupPriority[o];	**3**	**uimsbf**
/* Enhanced Object Metadata */		
goa_diffuseness[o];	**7**	**uimsbf**
goa_divergence[o];	**7**	**uimsbf**
goa_divergenceAzimuthRange[o];	**6**	**uimsbf**
goa_numExclusionSectors[o];	**4**	**uimsbf**
for (s = 0; s < goa_numExclusionSectors[o]; s++) {		
goa_usePredefinedSector[o][s];	**1**	**bslbf**
if (goa_usePredefinedSector[o][s]) {		
goa_excludeSectorIndex[o][s];	**4**	**uimsbf**
} else {		
goa_excludeSectorMinAzimuth[o][s];	**7**	**uimbsf**
goa_excludeSectorMaxAzimuth[o][s];	**7**	**uimbsf**
goa_excludeSectorMinElevation[o][s];	**5**	**uimbsf**
goa_excludeSectorMaxElevation[o][s]	**5**	**uimbsf**
}		
} */* for (s = 0; s < goa_numExclusionSectors[o]; s++) */*		
} */* for (o = 0; o < goa_numberOfOutputObjects; o++) */*		
/* GOA EXTENSION ELEMENTS */		
goa_numberOfExtensionElements;	**3**	**uimsbf**
if (goa_numberOfExtensionElements)		
{		
for (ext = 0; ext < goa_numberOfExtensionElements; ext++) {		
goa_extElementType;	**3**	**uimbsf**
goa_extElementLength;	**10**	**uimsbf**
switch (goa_extElementType) {		
case ID_EXT_GOA_PROD_METADATA:		
goa_Production_Metadata();		
break;		
default:		
break;		
}		
}		

Table 265 *(continued)*

Syntax	No. of bits	Mnemonic
}		
}		

Add new tables after Table 265:

Table AMD1.1 — Syntax of goa_Production_Metadata()

Syntax	No. of bits	Mnemonic
goa_Production_Metadata()		
{		
/* PRODUCTION METADATA CONFIGURATION */		
goa_ hasObjectDistance;	**1**	**bslbf**
if (goa_hasObjectDistance) {		
for (o = 0; o < goa_numberOfOutputObjects; o++) {		
goa_bsObjectDistance[o];	**9**	**uimsbf**
}		
}		
}		

Table AMD1.2 — Syntax of goa_extElementType

goa_extElementType	Value
ID_EXT_GOA_PROD_METADATA	0
/* reserved */	1-7

17.10.3.3. Semantics of the interface for object-based metadata

At the end of 17.10.3.3. add:

goa_numberOfExtensionElements Defines the number of extension elements to the GOA output interface.

goa_extElementType Defines the type of the extension element.

goa_extElementLength Defines the length of the extension element.

goa_hasObjectDistance This flag defines if the object distance parameter is signalled in the production metadata frame.

goa_bsObjectDistance This field describes the distance of an object. The field can take values between 0 and 511, which maps to distance values between 0 m and 177 km. Table AMD1.3 provides the mapping of goa_bsObjectDistance field to the distance.

Table AMD1.3 — Mapping of position_distance field to the distance

goa_bsObjectDistance	distance
0	distance = 0 m
1 – 511	distance = 0.01 * 2^(0.0472188798661443 * (goa_bsObjectDistance - 1))

17.10.4.1 General

In subclause 17.10.4.1, replace paragraph 8 with:

If a channel output interface is provided by an implementation, the following metadata shall be provided via the interface to be evaluated by possible external renderers:

— Number of channels;

— Number of valid PCM samples for the current frame;

— elementIDs for the referenced audio channels;

— Channel configuration;

— “fixed position” flag;

— Static group priority;

— Downmix matrix elements, if transmitted and matching the selected Reproduction Layout (according to 10.3.1);

— Production metadata.

17.10.4.2 Syntax of an interface for channel-based metadata

In subclause 17.10.4.2 replace Table 267 with:

Table 267 — Syntax of mpegh3da_getChannelMetadata()

Syntax	No. of bits	Mnemonic
mpegh3da_getChannelMetadata()		
{		
/* FRAME CONFIGURATION */		
gca_frameLength;	**6**	**uimsbf**
gca_audioTruncation;	**2**	**bslbf**
if (gca_audioTruncation>0) {		
gca_numSamples;	**13**	**uimsbf**
} else {		
gca_numSamples = gca_frameLength << 6;		
}		
/* CHANNEL METADATA */		
gca_numberOfOutputChannelGroups;	**9**	**uimsbf**
for (cGrp = 0; cGrp < gca_numberOfOutputChannelGroups; cGrp ++) {		
gca_numberOfChannels[cGrp];	**16**	**uimsbf**
gca_channelLayout[cGrp] = SpeakerConfig3d();		
for (nChn = 0; nChn < gca_numberOfChannels[cGrp]; nChn++ {		
gca_elementID[cGrp][nChn];	**9**	**uimsbf**
}		
/* TRACKING-RELATED METADATA */		
gca_fixedChannelsPosition[cGrp];	**1**	**bslbf**
/* GROUP-RELATED METADATA */		
gca_groupPriority[cGrp];	**3**	**uimsbf**
gca_channelGain[cGrp];	**8**	**uimsbf**
/* DOWNMIX MATRIX ELEMENT */		
gca_downmixAvailable;	**1**	**bslbf**
if (gca_downmixAvailable) {		
gca_downmixConfig();		
}		
}		
/* GCA EXTENSION ELEMENTS */		
gca_numberOfExtensionElements;	**3**	**uimsbf**
if (gca_numberOfExtensionElements)		
{		
for (ext = 0; ext < gca_numberOfExtensionElements; ext++) {		

Table 267 *(continued)*

Syntax	No. of bits	Mnemonic
gca_extElementType;	**3**	**uimbsf**
gca_extElementLength;	**10**	**uimsbf**
switch (gca_extElementType) {		
case ID_EXT_GCA_PROD_METADATA:		
gca_Production_Metadata();		
break;		
default:		
break;		
}		
}		
}		
}		

Add new tables following Table 267:

Table AMD1.4 — Syntax of gca_Production_Metadata()

Syntax	No. of bits	Mnemonic
gca_Production_Metadata()		
{		
/* PRODUCTION METADATA CONFIGURATION */		
for (gp = 0; gp < numChannelGroups; gp++) {		
gca_directHeadphone[gp]	**1**	**bslbf**
}		
gca_hasReferenceDistance;	**1**	**bslbf**
if (gca_hasReferenceDistance) {		
gca_bsReferenceDistance;	**7**	**uimsbf**
} else {		
gca_bsReferenceDistance = 57;		
}		
}		

Table AMD1.5 — Syntax of gca_extElementType

gca_extElementType	Value
ID_EXT_GCA_PROD_METADATA	0
/* reserved */	1-7

17.10.4.3 Semantics of the interface for channel-based metadata

In subclause 17.10.4.3, replace:

gca_groupPriority This field defines the priority of the group to which the current object belongs to. It can take integer values between 0 and 7.

with:

gca_groupPriority This field defines the priority of the group to which the current channel belongs to. It can take integer values between 0 and 7.

At the end of 17.10.4.3 add:

gca_numberOfExtensionElements Defines the number of extension elements to the GCA output interface.

gca_extElementType Defines the type of the extension element.

gca_extElementLength Defines the length of the extension element.

gca_directHeadphone This flag defines that the corresponding signal group of type channels goes directly to the headphone output. The signals are routed to left and right headphone channel. For mono, the signal is mixed to left and right headphone channel with a gain factor of 0.707.

gca_hasReferenceDistance This flag defines if the **gca_bsReferenceDistance** parameter is signalled in the production metadata config. If it is 0, the **gca_bsReferenceDistance** is set to 57, meaning the reference loudspeaker distance of input layout as 3.1748 m, by default.

gca_bsReferenceDistance This field describes the reference loudspeaker distance of input layout. The field can take values between 0 and 127, which maps to reference loudspeaker distance values between 0.5 m and 31.4 m. Table AMD1.6 provides the mapping of **gca_bsReferenceDistance** field to the reference loudspeaker distance.

Table AMD1.6 — Mapping of gca_bsReferenceDistance field to the reference loudspeaker distance

gca_bsReference Distance	reference distance
0 – 127	reference distance = 0.01 * 2^(0.0472188798661443 *(gca_bsReferenceDistance + 119))

17.10.5.1 General

In subclause 17.10.5.1, replace paragraph 7 with:

If the HOA output interface is provided by an implementation, the following metadata shall be provided via the interface to be interpreted and acted upon by potential external renderers:

— HOA order;

— Number of valid PCM samples for the current frame;

— Signal group related priority and fixedPosition parameter;

— NFC metadata;

— A flag that indicates if HOA content is relative to a screen and if so, the production screen size information;

— HOA rendering matrix elements, if transmitted and matching the selected reproduction layout;

— Production metadata.

17.10.5.2 Syntax of an interface for HOA metadata

In 17.10.5.2 replace Table 269 with:

Table 269 — Syntax of mpegh3da_getHoaMetadata()

Syntax	No. of bits	Mnemonic
mpegh3da_getHoaMetadata()		
{		
/* FRAME CONFIGURATION */		
gha_frameLength;	**6**	**uimsbf**
gha_audioTruncation;	**2**	**bslbf**
if (gha_audioTruncation>0) {		
gha_numSamples;	**13**	**uimsbf**
} else {		
gha_numSamples = gha_frameLength << 6;		
}		
gha_numberOfHoaGroups;	**9**	**uimsbf**
for (hGrp = 0; hGrp < gha_numberOfHoaGroups; hGrp ++) {		
/* Signal group related data */		
gha_fixedPosition[hGrp];	**1**	**bslbf**
gha_groupPriority[hGrp];	**3**	**uimsbf**
/* HOA METADATA */		
gha_HoaOrder[hGrp];	**9**	**uimsbf**
gha_UsesNfc[hGrp];	**1**	**bslbf**
if (gha_UsesNfc[hGrp]) {		
gha_NfcReferenceDistance[hGrp];	**32**	**bslbf**
}		
gha_hasSignalledHoaMatrix[hGrp];	**1**	**uimsbf**
if (gha_hasSignalledHoaMatrix[hGrp]) {		
gha_HoaRenderingMatrixSet();		
}		
gha_isScreenRelative[hGrp];	**1**	**uimsbf**
if (gha_isScreenRelative[hGrp]) {		
mae_ProductionScreenSizeData();		
mae_ProductionScreenSizeDataExtension();		
}		

Table 269 *(continued)*

Syntax	No. of bits	Mnemonic
}		
/* GHA EXTENSION ELEMENTS */		
gha_numberOfExtensionElements;	**3**	**uimsbf**
if (gha_numberOfExtensionElements)		
{		
for (ext = 0; ext < gha_numberOfExtensionElements; ext++) {		
gha_extElementType;	**3**	**uimbsf**
gha_extElementLength;	**10**	**uimsbf**
switch (gha_extElementType) {		
case ID_EXT_GHA_PROD_METADATA:		
gha_Production_Metadata();		
break;		
default:		
break;		
}		
}		
}		
}		

Add new tables following Table 269:

Table AMD1.7 — Syntax of gha_Production_Metadata()

Syntax	No. of bits	Mnemonic
gha_Production_Metadata()		
{		
/* PRODUCTION METADATA CONFIGURATION */		
gha_hasReferenceDistance;	**1**	**bslbf**
if (gha_hasReferenceDistance) {		
gha_bsReferenceDistance;	**7**	**uimsbf**
}		
else {		
gha_bsReferenceDistance = 57;		
}		
}		

Table AMD1.8 — Syntax of gha_extElementType

gha_extElementType	Value
ID_EXT_GHA_PROD_METADATA	0
/* reserved */	1-7

17.10.5.3 Semantics of the interface for HOA metadata

At the end of subclause 17.10.5.3, add:

gha_fixedPosition This field defines if the HOA soundfield orientation shall be updated during processing of scene displacement (tracking) data. If the soundfield orientation shall not be updated, the flag is set to 1.

gha_groupPriority This field defines the priority of the group to which the current HOA soundfield belongs to. It can take integer values between 0 and 7.

gha_numberOfExtensionElements Defines the number of extension elements to the GHA output interface.

gha_extElementType Defines the type of the extension element.

gha_extElementLength Defines the length of the extension element.

gha_isScreenRelative This element indicates if the HOA representation shall be rendered with respect to the reproduction screen size.

gha_hasReferenceDistance This flag defines if the **gha_bsReferenceDistance** parameter is signalled in the production metadata config. If it is 0, the **gha_bsReferenceDistance** is set to 57, meaning the reference loudspeaker distance of input layout as 3.1748 m, by default.

gha_bsReferenceDistance This field describes the reference loudspeaker distance of input layout. The field can take values between 0 and 127, which maps to reference loudspeaker distance values between 0.5 m and 31.4 m. Table AMD1.9 provides the mapping of **gha_bsReferenceDistance** field to the reference loudspeaker distance.

Table AMD1.9 — Mapping of gha_bsReferenceDistance field to the reference loudspeaker distance

gha_bsReferenceDistance	reference distance
0 – 127	reference distance = 0.01 * 2^(0.0472188798661443 *(gha_bsReferenceDistance + 119))

17.10.6 Audio PCM data

In subclause 17.10.6, replace paragraph 3 with:

The decoder shall signal the offset index of the PCM buffer for the first un-rendered output object and the offset index of the PCM buffer for the first HOA audio signal.

17.10 Interfaces for channel-based, object-based, and HOA metadata and audio data

At the end of subclause 17.10 add:

17.11 Interface for positional scene displacement data

17.11.1 General

For applications which allow small user movements (–25 cm … +25 cm) in the audio scene, the user position data for the binaural rendering may be provided to the decoder by using the syntax element mpegh3daPositionalSceneDisplacementData(). This will allow the scene displacement processing to account for user orientation changes and positional displacement.

17.11.2 Syntax of the positional scene displacement interface

Table AMD1.10 — Syntax of mpegh3daPositionalSceneDisplacementData()

Syntax	No. of bits	Mnemonic
mpegh3daPositionalSceneDisplacementData()		
{		
sd_azimuth;	**8**	**uimsbf**
sd_elevation;	**6**	**uimsbf**
sd_radius;	**4**	**uimsbf**
}		

17.11.3 Semantics of the positional scene displacement interface

sd_azimuth This field defines the scene displacement azimuth position. This field can take values from –180 to 180:

$$az_{offset} = (\text{sd\_azimuth} - 128) \cdot 1.5$$

$$az_{offset} = \min(\max(az_{offset}, -180), 180)$$

sd_elevation This field defines the scene displacement elevation position. This field can take values from –90 to 90:

$$el_{offset} = (\text{sd\_elevation} - 32) \cdot 3.0$$

$$el_{offset} = \min(\max(el_{offset}, -90), 90)$$

sd_radius This field defines the scene displacement radius. This field can take values from 0 and 0.25:

$$r_{offset} = \text{sd\_radius} / 60$$

17.11.4 Processing

When mpegh3daPositionalSceneDisplacementData() is used, the scene displacement defined in 18.8 must be adjusted with the following values:

$$az'_{offset} = az_{offset} + 90°$$

$$el'_{offset} = 90° - el_{offset}$$

This results in new position transferred to Cartesian coordinates (x,y,z):

$$x = r \cdot \sin(el') \cdot \cos(az') + r_{offset} \cdot \sin\left(el'_{offset}\right) \cdot \cos\left(az'_{offset}\right)$$

$$y = r \cdot \sin(el') \cdot \sin(az') + r_{offset} \cdot \sin\left(el'_{offset}\right) \cdot \sin\left(az'_{offset}\right)$$

$$z = r \cdot \cos(el') + r_{offset} \cdot \cos\left(el'_{offset}\right)$$

20.5.1 Definition

In subclause 20.5.1 replace the following text:

Box Types: `'mhaC'`, `'mha1'`, `'mha2'`

Container: Sample Table Box (`'stbl'`)

Mandatory: The mha1 box is mandatory

with:

Box Types: `'mhaC'`, `'mha1'`, `'mha2'`

Container: Sample Table Box (`'stbl'`)

Mandatory: No

20.9.5.3 Semantics

In subclause 20.9.5.3 remove:

`multiStream` defined in subclause 20.8

Clause 26

Add new Clauses 27 and 28 after Clause 26:

27 Production metadata decoding

27.1 General

Audio metadata originates from production tools and production formats. Audio metadata should be made available in the bit stream to enable a renderer to perform advanced rendering of immersive audio. This clause describes the production metadata and the decoding process thereof.

27.1.1 Object distance coding

The object distance is signalled as an 9-bit value allowing coding of values from 0 m up to 177 km when using an exponential mapping. The resolution of the distance is highest for near positions (<1 mm) and lowest in the far positions (around 5 km). The very low distances, below about 1 cm, are considered less important, thus the distance coding starts from 1cm for the second quantized value (=1). The lowest value signals distance =0.

27.1.2 Direct headphone signalling

The **directHeadphone** flag defines that the corresponding signal group of type channels goes to the headphone output directly. The channel group can be mono or stereo, i.e. the **directHeadphone** flag shall be 0 for all signal groups of type channels, which have a different layout than mono or stereo assigned to them. For stereo, the two signals are mixed to left and right headphone channel, directly. For mono:

— the signal is mixed to the left channel directly, if the CICPspeakerIdx == 0,

— the signal is mixed to the right channel directly, if the CICPspeakerIdx == 1,

— the signal is mixed to left and right headphone channel with a gain factor of 0.707, otherwise.

Over loudspeakers, the signals would come out at the speakers indicated in the CICP layout index.

The signals flow for the **directHeadphone** channels is modified only if a binaural output signal is generated. For decoding and rendering for loudspeaker playback, no change is needed and the signal is mixed to the output channels according to the rule set of the format converter.

When using the channel output interface, the **directHeadphone** signal is provided to the output interface, as shown in Figure AMD1.1.

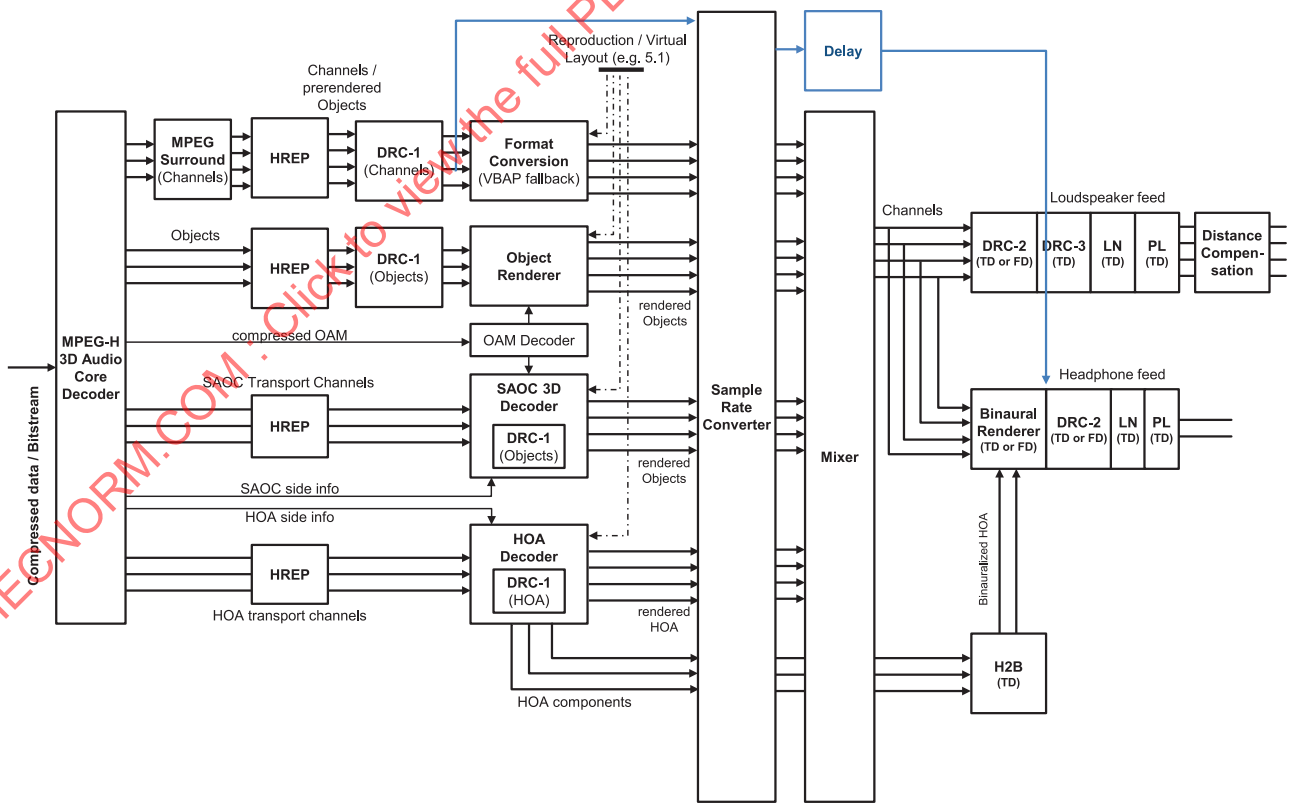

Figure AMD1.1 — Signal flow diagram showing the routing of directHeadphone channel groups to the output

In case of binaural rendering, the **directHeadphone** channels are processed by DRC1 and then bypass the format converter, mixer and binaural renderer. The sampling rate of the **directHeadphone** channels is converted to match the output sampling rate. The **directHeadphone** channels are delay-aligned to match the delay introduced to the non-directHeadphone signals by the format converter and the binaural renderer. The **directHeadphone** channels are then mixed with the input of DRC2 from the binaural renderer.

27.1.3 Reference distance coding

The reference loudspeaker distance of input layout is signalled as a 7-bit value allowing coding of values from 0.5 m up to 31.4 m. The distances below 0.5 m are considered less important in terms of loudspeaker layout, thus the distance coding starts from 0.5 m for the first quantized value (=0). When the reference distance is not defined in the bitstream, it is assumed to be 3.1748 m. Note that each quantized reference distance value is identical to one of the quantized object distance values.

27.2 Syntax

27.2.1 Production metadata configuration

Table AMD1.11 — Syntax of prodMetadataConfig()

Syntax	No. of bits	Mnemonic
prodMetadataConfig()		
{		
/* high resolution reference distance */		
has_reference_distance;	**1**	**bslbf**
if (has_reference_distance) {		
bs_reference_distance;	**7**	**uimsbf**
} else {		
bs_reference_distance = 57;		
}		
/* high resolution object distance */		
for (gp = 0; gp < numObjectGroups; gp++) { /* NOTE 1 */		
has_object_distance[gp];	**1**	**bslbf**
}		
/* direct to headphone */		
for (gp = 0; gp < numChannelGroups; gp++) { /* NOTE 2 */		
directHeadphone[gp];	**1**	**bslbf**
}		
}		

NOTE 1 numObjectGroups represents the number of signal groups with signalGroupType == SignalGroupTypeObject as given by the Signals3d() structure in Table 14.

NOTE 2 numChannelGroups represents the number of signal groups with signalGroupType == SignalGroupTypeChannel as given by the Signals3d() structure in Table 14.

27.2.2 Production metadata frame

Table AMD1.12 — Syntax of prodMetadataFrame()

Syntax	No. of bits	Mnemonic

prodMetadataFrame()
{
for (gp = 0; gp < numObjectGroups; gp++) { /* NOTE 1 */
if (has_object_distance [gp]) {
has_intracoded_data;
	1	**bslbf**

if (has_intracoded_data) {
intracodedProdMetadataFrame();
}
else {
dynamicProdMetadataFrame();
}		

} } }		
NOTE 1 numObjectGroups represents the number of signal groups with signalGroupType == SignalGroupTypeObject as given by the Signals3d() structure in Table 14. NOTE 2 intracodedProdMetadataFrame() shall occur in the first frame and at each random access point.		

Table AMD1.13 — Syntax of intracodedProdMetadataFrame()

Syntax	No. of bits	Mnemonic

intracodedProdMetadataFrame();
{
if (num_objects>1) { /* NOTE 1 */
fixed_distance;
	1	**bslbf**

if (fixed_distance) { **default_distance;**	**9**	**tcimsbf**

}
else {
common_distance;
	1	**bslbf**

if (common_distance) { **default_distance;**	**9**	**tcimsbf**

}
else {
for (o = 0; o < num_objects; o++) {
position_distance[o];
	9	**tcimsbf**

} } }		
} else {		
NOTE 1 num_objects is equal to the number of objects in the associated signal group.		

Table AMD1.13 *(continued)*

Syntax	No. of bits	Mnemonic
position_distance[0];	**9**	**tcimsbf**
}		
}		
NOTE 1 num_objects is equal to the number of objects in the associated signal group.		

Table AMD1.14 — Syntax of dynamicProdMetadataFrame()

Syntax	No. of bits	Mnemonic
dynamicProdMetadataFrame() {		
flag_dist_absolute;	**1**	**bslbf**
for (o = 0; o < num_objects; o++) { /* NOTE 1 */		
if (has_object_metadata) { /* NOTE 2 */		
singleDynamicProdMetadataFrame(flag_dist_absolute);		
}		
}		
}		
NOTE 1 num_objects is equal to the number of objects in the associated signal group.		
NOTE 2 has_object_metadata is given by the dynamic_object_metadata() structure in Table 142.		

Table AMD1.15 — Syntax of singleDynamicProdMetadataFrame()

Syntax	No. of bits	Mnemonic
singleDynamicProdMetadataFrame(flag_dist_absolute) {		
if (flag_dist_absolute) {		
if (!fixed_distance) { /* NOTE 1 */		
position_distance;	**9**	**tcimsbf**
}		
} else {		
if (!fixed_distance) { /* NOTE 1 */		
flag_distance;	**1**	**bslbf**
if (flag_distance) {		
nBitsDistance;	**3**	**uimsbf**
num_bits = nBitsDistance + 2;		
position_distance_difference;	**num_bits**	**tcimsbf**
}		
}		
}		
}		
NOTE 1 fixed_distance given in the preceding intracodedProdMetadataFrame().		

27.3 Semantics

27.3.1 Production metadata configuration

has_reference_distance This flag defines if the **bs_reference_distance** parameter is signalled in prodMetadataConfig(). If **has_reference_distance** == 0 the **bs_reference_distance** is set to 57, meaning the reference loudspeaker distance of input layout as 3,1748 m, by default.

bs_reference_distance This field describes the reference loudspeaker distance of input layout. The field can take values between 0 and 127, which maps to distance values between 0.5 m and 31.4 m. Table AMD1.16 provides the mapping of bs_reference_distance field to the reference loudspeaker distance.

Table AMD1.16 — Mapping of bs_reference_distance field to the reference loudspeaker distance

bs_reference_distance	reference distance
0 – 127	reference distance = 0.01 * 2^(0.0472188798661443 * (bs_reference_distance + 119))

has_object_distance This flag defines if the object distance parameter is signalled in prodMetadataFrame().

directHeadphone This flag defines that the corresponding signal group of type channels goes to the headphone output, directly, if the binaural output is rendered. The signals are routed to left and right headphone channel. For mono, the signal is mixed to left and right headphone channel with a gain factor of 0.707.

has_intracoded_data Flag indicating that the current frame holds intracoded data.

position_distance This field describes the distance between the centre of the head of the listener at the sweet spot position and an object. The field can take values between 0 and 511, which map to distance values between 0 m and 177 kilometres. Table AMD1.17 provides the mapping of **position_distance** field to the distance.

Table AMD1.17 — Mapping of position_distance field to the distance

position_distance	distance
0	distance = 0 m
1 – 511	distance = 0.01 * 2^(0.0472188798661443 * (position_distance - 1))

fixed_distance Flag indicating whether the distance value is fixed for all objects.

common_distance Indicates whether a common distance value is used for all objects.

default_distance Defines the value of the common distance for all objects.

flag_dist_absolute Flag indicating whether the values of the components are transmitted differentially or in absolute values.

flag_distance	Flag per object indicating whether the distance value changes for this intra-frame period.
nBitsDistance	Defines how many bits are required to represent the differential value minus 2.
position_distance_difference	value of the difference between the linearly interpolated and the actual value of distance.

27.4 Decoding process

The prodMetadataConfig() is defined in and mpegh3daExtElement() structure, as defined in Table 23. The prodMetadataFrame() structure is located in an mpegh3daExtElementConfig(), as defined in Table 76.

28 Earcon metadata

28.1 General

Region-of-interest (ROI) information is supported in OMAF via the sphere region timed metadata. One way for guiding a user to pay attention and view an ROI is using audible messages called earcons. The earcon metadata fully describes the properties and the spatial position of the individual earcons.

28.2 Syntax

Table AMD1.18 — Syntax of earconInfo()

Syntax	No. of bits	Mnemonic
earconInfo()		
{		
bsNumEarcons;	**7**	**uimsbf**
for (i=0; i< bsNumEarcons + 1; i++) {		
earconIsIndependent[i];	**1**	**uimsbf**
earconID[i];	**7**	**uimsbf**
earconType[i];	**4**	**uimsbf**
earconActive[i];	**1**	**bslbf**
earconPositionType[i];	**2**	**bslbf**
if (earconPositionType[i] == 0) {		
earcon_CICPspeakerIdx[i];	**7**	**uimsbf**
} else {		
if (earconPositionType[i] == 1) {		
earcon_azimuth[i];	**8**	**uimsbf**
earcon_elevation[i];	**6**	**uimsbf**
earcon_distance[i];	**9**	**uimsbf**
} else {		
/* default position information */		
earcon_azimuth[i] = 0;		
earcon_elevation[i]; = 0;		
earcon_distance[i] = 177 /* default reference distance*/		
}		

Table AMD1.18 *(continued)*

Syntax	No. of bits	Mnemonic
}		
earconHasGain;	**1**	**bslbf**
if (earconHasGain) {		
earcon_gain[i];	**7**	**uimsbf**
}		
earconHasTextLabel;	**1**	**bslbf**
if (earconHasTextLabel) {		
earconNumLanguages[i];	**4**	**uimsbf**
for (n=0; n< earconNumLanguages[i]; n++) {		
earconLanguage[i][n];	**24**	**uimsbf**
earconTextDataLength[i][n];	**8**	**uimsbf**
for (c=0; c< earconTextDataLength[i][n]; c++) {		
earconTextData[i][n][c];	**8**	**uimsbf**
}		
}		
}		
}		
}		

Table AMD1.19 — Syntax of pcmDataConfig()

Syntax	No. of bits	Mnemonic
pcmDataConfig()		
{		
bsNumPcmSignals;	**7**	**uimsbf**
pcmAlignAudioFlag;	**1**	**bslbf**
pcmSamplingRateIndex;	**5**	**bslbf**
if (pcmSamplinRateIndex == 0x1f) {		
pcmSamplingRate;	**24**	**uimsbf**
}		
pcmBitsPerSampleIndex;	**3**	**uimsbf**
pcmFrameSizeIndex;	**3**	**uimsbf**
if (pcmFrameSizeIndex == 5) {		
pcmFixFrameSize;	**16**	**uimsbf**
}		
for (i=0; i< bsNumPcmSignals + 1; i++) {		
pcmSignal_ID[i];	**7**	**uimsbf**
}		
bsPcmLoudnessValue;	**8**	**uimsbf**
pcmHasAttenuationGain;	**2**	**uimsbf**
if (pcmHasAttenuationGain == 1)		
bsPcmAttenuationGain;	**8**	**uimsbf**
}		