SMPTE ST 2098-2:2022
Revision of 2098-2:2019

SMPTE STANDARD

Immersive Audio Bitstream Specification

Table of Contents

Approved May 25, 2022

Foreword

SMPTE (the Society of Motion Picture and Television Engineers) is an internationally recognized standards developing organization. Headquartered and incorporated in the United States of America, SMPTE has members in over 80 countries on six continents. SMPTE's Engineering Documents, including Standards, Recommended Practices, and Engineering Guidelines, are prepared by SMPTE's Technology Committees. Participation in these Committees is open to all with a bona fide interest in their work. SMPTE cooperates closely with other standards-developing organizations, including ISO, IEC and ITU.

SMPTE Engineering Documents are drafted in accordance with the rules given in its Standards Operations Manual. This SMPTE Engineering Document was prepared by Technology Committee 25CSS.

The following summarizes the substantive changes made from SMPTE ST 2098-2:2019 to this edition:

1. Clarified usage of "used" and "rendered" throughout the document
2. Clarified MaxRendered value (10.2.5 and 10.7.1)
3. Added definitions for Sub-Element, Ancestor, Descendant, Parent, and Child/Children (Clause 4)
4. Removed use of commas in large numbers (Table 23)
5. Clarified mapped value of DEFAULT_OBJ_SNAP_TOL (Table 32)
6. Clarified AuthoringToolURI constraints (10.9.1)
7. Clarified ChannelCount (10.3.4)
8. Clarified the sets of ChannelID and DestinationChannelID codes allowed to be present in BedDefinition (10.3.5) or BedRemap (10.4.4) elements
9. Deleted use of "declared" (10.2.5), "default bed" and "default object" (Annex A and Annex B), and "active" (Annex C)
10. Clarified Bitstream Organization examples (A.2, A.3, and A.5)

Intellectual Property

At the time of publication no notice had been received by SMPTE claiming patent rights essential to the implementation of this Engineering Document. However, attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. SMPTE shall not be held responsible for identifying any or all such patent rights.

Introduction

This clause is entirely informative and does not form an integral part of this Engineering Document.

This document has two main sections to reflect the structure of the Immersive Audio Bitstream. The structure is defined to have two levels of wrapping. At the top level, there are two defined segments as defined in Clause 7: the preamble segment and the IAFrame segment. This wrapping is documented in order to comply with existing practice, but does not add value to the bitstream. The second level of wrapping is within the IAFrame and is described starting in Clause 9. All immersive audio data is carried in the IAFrame.

The Immersive Audio Bitstream can carry audio channels (bed channels) that are destined for particular Loudspeakers or Loudspeaker arrays within the auditorium. The bitstream can also carry Audio Objects that are “rendered” to particular Loudspeakers or combinations of Loudspeakers based on positional data in the bitstream and auditorium-specific information about Loudspeakers and their locations. Bitstream immersive audio data is broken into IAFrames, each corresponding to an image edit unit. For example, if the image edit rate is 24 edit units per second, an IAFrame contains the audio to be played during that 1/24 second period of time. An IAFrame contains a number of AudioData elements (AudioDataPCM or AudioDataDLC), each containing a single audio waveform encoded as PCM or a losslessly compressed version of PCM.

BedDefinition elements tell the renderer which particular audio element is expected to be routed to which Loudspeaker or array of Loudspeakers. BedDefinition elements “point to” AudioData elements using the AudioDataID and Loudspeaker channels using the ChannelID. BedRemap elements are Sub-Elements of BedDefinition elements. BedRemap elements define how a Bed mixed for one Loudspeaker configuration can be played in another.

Object elements tell the renderer which particular AudioData element is expected to be “panned” to a particular location in the auditorium. As with a Bed channel, an object element “points to” AudioData elements and pans the audio as per the panning instructions included in the object element. Because the object could be moving during the IAFrame, the object element is broken into “sub blocks” to increase the update rate of position data.

1 Scope

The normative portions of this document define a coded representation (bitstream) of an audio program. Information is included to describe typical uses of the format. The bitstream carries audio essence and metadata necessary to reproduce a complete audio program.

2 Conformance Notation

Normative text is text that describes elements of the design that are indispensable or contains the conformance language keywords: "shall", "should", or "may". Informative text is text that is potentially helpful to the user, but not indispensable, and can be removed, changed, or added editorially without affecting interoperability. Informative text does not contain any conformance keywords.

All text in this document is, by default, normative, except: the Introduction, any clause explicitly labeled as "Informative" or individual paragraphs that start with "Note:”

The keywords "shall" and "shall not" indicate requirements strictly to be followed in order to conform to the document and from which no deviation is permitted.

The keywords, "should" and "should not" indicate that, among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

The keywords "may" and "need not" indicate courses of action permissible within the limits of the document.

The keyword “Reserved” indicates a provision that is not defined at this time, shall not be used, and may be defined in the future. The keyword “forbidden” indicates “Reserved” and in addition indicates that the provision will never be defined in the future.

A conformant implementation according to this document is one that includes all mandatory provisions ("shall") and, if implemented, all recommended provisions ("should") as described. A conformant implementation need not implement optional provisions ("may") and need not implement them as described.

Unless otherwise specified, the order of precedence of the types of normative information in this document shall be as follows: Normative prose shall be the authoritative definition; Tables shall be next; followed by formal languages; then figures; and then any other language forms.

3 Normative References

The following standards contain provisions which, through reference in this text, constitute provisions of this engineering document. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this engineering document are encouraged to investigate the possibility of applying the most recent edition of the standards indicated below.

SMPTE ST 2098-1:2018 Immersive Audio Metadata

SMPTE ST 298:2009, Universal Labels for Unique Identification of Digital Data

SMPTE ST 428-12:2013, D-Cinema Distribution Master - Common Audio Channels and Soundfield Groups

SMPTE ST 2098-5:2018, D-Cinema Immersive Audio Channels and Soundfield Groups

IETF RFC 3986, Uniform Resource Identifiers (URI): Generic Syntax, January 2005

IETF RFC 1035, Domain names - implementation and specification, November 1987

Recommendation ITU-R BS.2051-2, Advanced sound system for programme production

4 Terms and Definitions

For the purposes of this document, the following terms and definitions apply. See Normative References for definitions of other terms used in this document.

4.1
Bed (also, Immersive Audio Bed)

Soundfield Group, such as a 5.1, 7.1 or 9.1, that serves as the foundation of the immersive soundtrack mix

Note 1 to entry: A Bed is typically present for the duration of the program. Audio Objects may be added to the immersive soundtrack mix to augment the Bed. The Loudspeaker assignments for a Bed are typically static and do not change during the program.

4.2
IABitstream frame

smallest editable unit of the Immersive Audio Bitstream

4.3
IAFrame

portion of an IABitstream frame that contains all audio essence and metadata necessary to reconstruct one frame of audio

4.4
IAFrame Rate

number of IAFrames per second

Note 1 to entry: The IAFrame Rate normally matches the edit rate of an associated picture track.

4.5
Target Environment

specific set of conditions that is present in the playback environment

4.6
Sub-Element

element contained within another element

4.7
Ancestor

element or Sub-Element that contains the reference element

4.8
Descendant

Sub-Element contained within the reference element

4.9
Parent

Ancestor element with no intervening Sub-Elements

4.10
Child
Children

Descendant with no intervening Sub-Elements

Note 1 to entry to Definitions 4.7, 4.8, 4.9, and 4.10: With reference to Figure 1,

A is an Ancestor of B, C, D, and E.
A is a Parent of B and C.
B is an Ancestor and Parent of D.
C is an Ancestor and Parent of E.
B, C, D, and E are Descendants of A.
B and C are Descendants and Children of A.
D is a Child and Descendant of B.
E is a Child and Descendant of C.

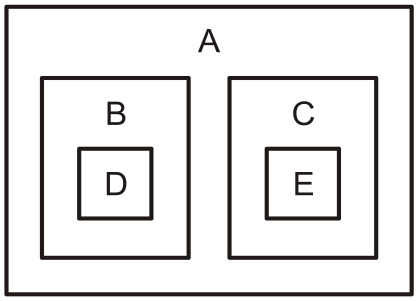

Figure 1 **— Family relationships.**

5 Coding of Numbers and Data Types

5.1 Encoding of data fields

Unless otherwise specified, bitstream data fields that are larger than 1 bit shall be encoded as follows: most-significant-bit-first for fields that represent numeric data (e.g., PreambleLength); or left-bit-first for fields that represent non-numeric data (e.g., UseCase codes). When the bitstream is represented as a bytestream, the first bit of the bitstream shall correspond to the most significant bit of the first byte of the bytestream. Consecutive bits of the bitstream shall correspond in sequence to the remaining bits of the first byte and then in the same pattern to the bits of consecutive bytes of the bytestream.

5.2 Plex encoding

Plex(n) defines a field size of n bits that may be expanded if it is insufficient to hold the desired symbol. A data field of all ones is an escape code. If the desired symbol is not the escape code and fits within the allocated field, the symbol shall be placed therein, with leading zeros (as needed) to fill the field. Otherwise, an escape code is used to signal that a replacement symbol follows in the bitstream. The field size for the replacement symbol is twice as long as the field it replaces. This process continues, with additional escape codes and doubling of field size, until a field with size sufficient to carry the desired symbol is provided. The desired symbol shall be placed therein, with leading zeros (as needed) to fill the field. Prior to Plex coding, any leading zeros of a symbol (e.g., an unsigned integer) shall be dropped in order to fit the symbol into a smaller container. During decoding, leading zeros shall be prepended as needed to reproduce the full symbol. Plex coded symbols shall be encoded in the smallest container possible for the value to be encoded. For example, 0x1234 is to be Plex(8) encoded as 0xFF1234 instead of 0xFFFFFF00001234. In this document, symbols to be Plex encoded shall have a value less than or equal to 0xFFFFFFFE.

EXAMPLE: Plex encoding (Informative)

Encode the 32-bit number 0x12345678 as Plex(4).

F – Number does not fit in 4 bits. Put down escape code and double field size to 8 bits.

FF – Number does not fit in 8 bits. Put down escape code and double field size to 16 bits.

FFFF – Number does not fit in 16 bits. Put down escape code and double field size to 32 bits.

12345678 – Number fits in 32 bits.

Full coding is F FF FFFF 12345678.

Encode the 32-bit number 0x12345678 as Plex(8).

FF – Number does not fit in 8 bits. Put down escape code and double field size to 16 bits.

FFFF – Number does not fit in 16 bits. Put down escape code and double field size to 32 bits.

12345678 – Number fits in 32 bits.

Full coding is FF FFFF 12345678

Encode the 8-bit number 0xFF as Plex(8)

FF – The number fits in 8 bits, but FF is reserved as an escape. Put down escape code and double field size to 16 bits

00FF – Number fits in 16 bits.

Full coding is FF 00FF.

5.3 UseCase encoding

A UseCase code is an 8-bit value that indicates the Target Environment for which a bitstream element is intended. The defined UseCase codes are shown in Table 1 and are associated with Soundfield Configurations as defined in Table 1.

NOTE Decoders can compare the UseCase code associated with an element to configuration information, such as the Loudspeakers used for channel support, to determine whether that element can be used.

Table 1 — UseCase Codes.

UseCase code	General Description	SMPTE ST 428-12:2013 (Annex C) Reference	SMPTE ST 2098-5:2018 (Annex C) Reference	ITU-R BS.2051-2 Reference
0x01	Intended for use with a specific Soundfield Configuration	5.1		
0x02	Intended for use with a specific Soundfield Configuration	7.1DS		
0x03	Intended for use with a specific Soundfield Configuration	7.1SDS		
0x04	Intended for use with a specific Soundfield Configuration		11.1HT	
0x05	Intended for use with a specific Soundfield Configuration		13.1HT	
0x06	Intended for use with a specific Soundfield Configuration		9.1OH	
0x07-0x2F	Reserved for D-Cinema			
0x30	Intended for use with a specific Soundfield Configuration			Speaker Layout associated with sound system A
0x31	Intended for use with a specific Soundfield Configuration			Speaker Layout associated with sound system D
0x32	Intended for use with a specific Soundfield Configuration			Speaker Layout associated with sound system J
0xFF	Always Use			
All other values	Reserved			

5.4 Relative distance coding

Throughout the bitstream, distance metadata on or within the unit cube shall be coded as an n-bit unsigned integer (D_n) that maps linearly into the range [0,1], but the mapping depends on the axis. For the X and Y axes, the DistanceXY formula shall be used. For the Z axis, the DistanceZ formula shall be used. See 11.1 for an explanation of the unit cube and the associated axes.

$$\text{DistanceXY} = D_n/2^{(n-1)} - (2^{(n-1)} - 1)/2^{(n-1)},\ \ 2^{n-1} - 1 <= D_n <= (2^n) - 1$$

$$\text{DistanceZ} = D_n/(2^n - 1),\ \ 0 <= D_n <= (2^n) - 1$$

5.5 Amplitude Gain

Gains, except zone gains, shall be expressed as a 10-bit unsigned integer. If the 10-bit value (G) is 0x3ff, samples shall be multiplied by zero (linear gain is zero or negative infinity dB). Otherwise, samples shall be multiplied by $2^{-G/64}$. For example, if the 10-bit encoded value G is 0, the linear gain is $2^0 = 1$. Samples are multiplied by 1. If G = 64 (0x40), samples are multiplied by $2^{-1} = 1/2$, which corresponds to a gain of -6 dB.

6 Bitstream Container and Syntax Description - Informative

The Immersive Audio Bitstream is composed of a number of sequential frames, where each frame is an IABistream frame and contains a Preamble segment and an IAFrame segment. The Preamble segment and tag-length-value wrapping of the IAFrame are documented to allow existing files to be compliant with this specification and do not provide any other function. The bulk of this document describes the IAFrame.

The IAFrame uses a hierarchical container format composed of “elements” similar in concept to “chunks” in the Resource Interchange File Format (RIFF). Each element consists of a header and payload. The header consists of a unique identifier (ElementID), and the ElementSize. The ElementID specifies how the data within the element is interpreted. ElementSize indicates the size in bytes of the following payload. The bits to code ElementID and ElementSize are not included in the calculation of ElementSize. The element payload can contain Sub-Elements. The ElementSize includes the size of all Sub-Elements contained within the payload. Sub-Elements contain additional description data related to the Parent element.

The bitstream specification in Clauses 7 and 9 is documented using pseudocode. The pseudocode describes the order of arrival of symbols within the bitstream. The pseudocode is roughly based on C language syntax, but simplified for ease of reading. Symbols carried in the bitstream are designated by bold face font.

The length of each symbol (symbol length) is indicated in bits to the right of the symbol name. In some cases, the symbol is coded using a method called Plex coding to allow for variable length symbols. See 5.2 for details.

7 Bitstream Segment Specification

The Immersive Audio bitstream shall be a sequence of IABitstream frames. Each IABitstream frame consists of two tag-length-value wrapped segments: a preamble segment and an IAFrame segment as shown in Table 2. Successive frames shall correspond to adjoining, non-overlapping audio segments.

Table 2 — IABitstream Syntax.

IABitstream Syntax	symbol length
IAB IABitstream	
{	
while(true)	
{	
PreambleTag	8
PreambleLength.....	32
PreambleValue.....	variable
IAFrameTag.....	8
IAFrameLength.....	32
IAFrame	variable
}	
} / /* end of IABitstream */	

8 IABitstream Field Description

The following sections define the bit fields referenced by name in Clause 7. Data fields are defined in the following sections, and each field name is followed by the number of bits used by the field.

8.1 IABitstream data fields

8.1.1 PreambleTag – 8 bits

The PreambleTag value shall be equal to 0x01.

8.1.2 PreambleLength – 32 bits

The PreambleLength field shall indicate the size, in bytes, of the PreambleValue field. The PreambleLength field may have a value of 0.

8.1.3 PreambleValue – Variable

The PreambleValue field shall be comprised of an integer number of payload bytes. Specification of the content of the payload is outside the scope of this specification.

8.1.4 IAFrameTag – 8 bits

The IAFrameTag value shall be 0x02.

8.1.5 IAFrameLength – 32 bits

The IAFrameLength field shall indicate the size, in bytes, of the IAFrame.

8.1.6 IAFrame – Variable

The IAFrame is a data structure described in Clause 9.

9 Bitstream IAFrame Specification

Each IAFrame contains all the information (metadata and essence) necessary to describe the entire audio program for one time segment. The duration of each coded time segment shall correspond to one of the supported frame rates as listed in Table 17. The audio frames shall be aligned with the audio program edit units. In this document, a Frame shall mean an IAFrame element (see 9.1). The IAFrame is an instance of an IAElement syntax.

The IAFrame uses a hierarchical container format composed of “elements.” The IAElement syntax in Table 3 describes the header structure that begins all Immersive Audio elements.

Table 3 — IAElement Syntax.

IAElement Syntax	symbol length

```
IAElement
{
   ElementID.....................................................Plex(8)
   ElementSize.................................................. Plex(8)

   switch(ElementID)
   {
      case(IA_FRAME)
         IAFrame
      break
      case(BED_DEFINITION)
         BedDefinition
      break
      case(BED_REMAP)
         BedRemap
      break
      case(OBJECT_DEFINITION)
         ObjectDefinition
      break
      case (OBJECT_ZONE_DEFINITION19)
          ObjectZoneDefinition19
      break
      case (AUTHORING_TOOL_INFO)
          AuthoringToolInfo
      break
      case (USER_DATA)
          UserData
      break
      case(AUDIO_DATA_DLC)
         AudioDataDLC
      break
      case(AUDIO_DATA_PCM)
         AudioDataPCM
      break
      default
         UnknownData.....................................ElementSize * 8
   }

} /* end of IAElement */
```

Table 4 specifies which Children an element can carry.

Table 4 — Allowed Children.

Element Name	Allowed Children
IAFrame	BedDefinition ObjectDefinition AudioDataDLC AudioDataPCM AuthoringToolInfo UserData
BedDefinition	BedDefinition BedRemap
BedRemap	None
ObjectDefinition	ObjectDefinition ObjectZoneDefinition19
AudioDataDLC	None
AudioDataPCM	None
AuthoringToolInfo	None
UserData	None
ObjectZoneDefinition19	None

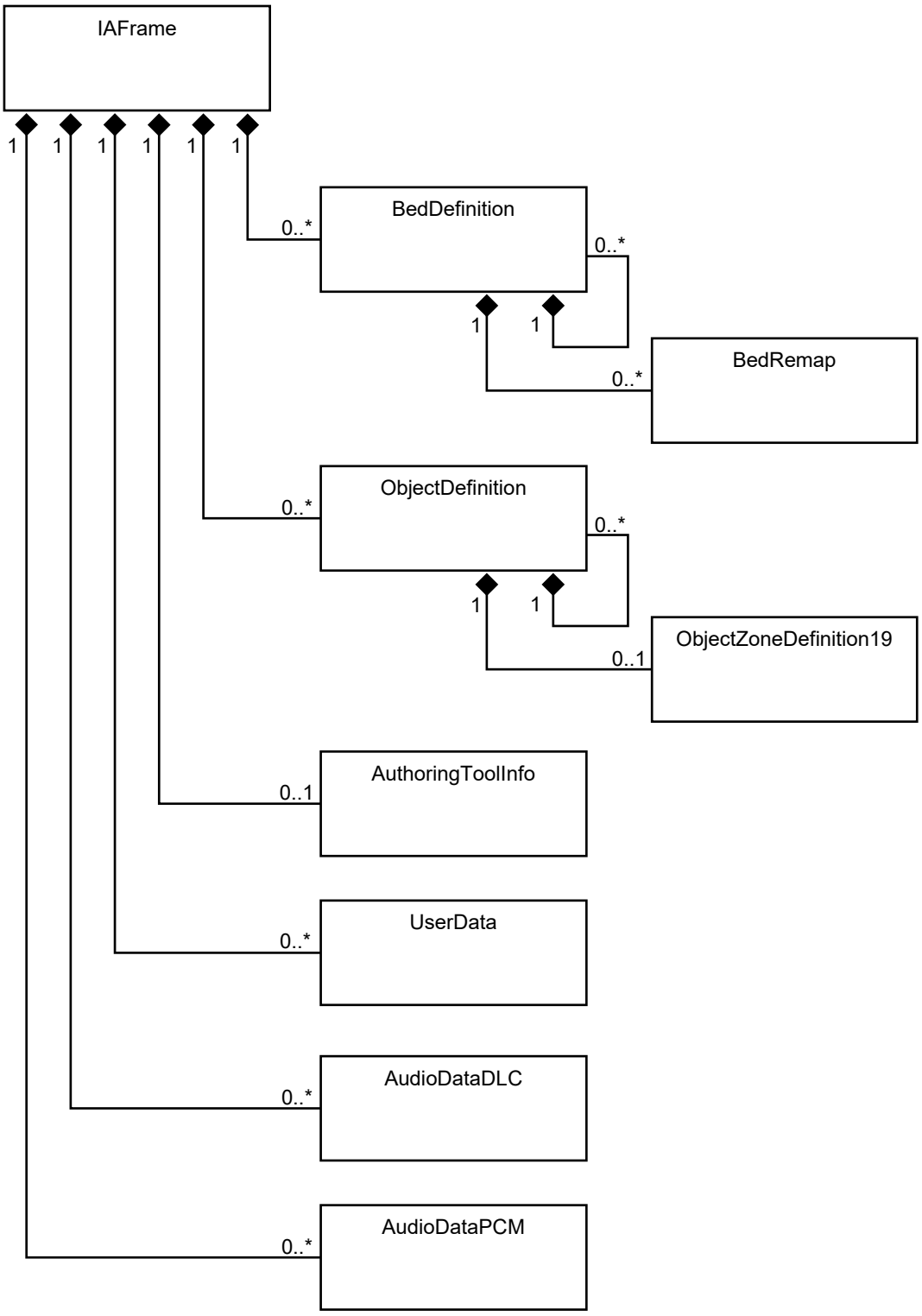

Figure 2 — UML class diagram showing possible element Parent-Child relationships and cardinalities.

Figure 2 illustrates the possible element Parent-Child relationships and cardinalities. A connecting line with a diamond shape at one end indicates a composition aggregation meaning that an element type on the side of the diamond shape is composed of element types on the other side of the connecting line. The number of elements that the element at the side with the diamond shape is composed of is given by the cardinality at the other end of the connecting line. A cardinality of “0..*” is interpreted as “zero or more”. For example, the composition of an IAFrame element includes zero or more AudioDataDLC elements. A reflexive composition is indicated by a connecting line starting and ending at the same element type. For example, an ObjectDefinition element can be composed of zero or more further ObjectDefinition elements.

9.1 IAFrame Syntax

The IAFrame element contains all audio and metadata elements required to decode one frame of audio. The IAFrame element shall have the structure shown in Table 5. The following elements shall be supported as Children of an IAFrame: BedDefinition, ObjectDefinition, AudioDataDLC, AudioDataPCM, AuthoringToolInfo, and UserData. Other Child elements shall be ignored.

Table 5 — IAFrame Syntax.

IAFrame Syntax	symbol length
IAFrame	
{	
Version	8
SampleRate	2
BitDepth	2
FrameRate	4
MaxRendered	Plex(8)
AlignBits /* extra bits to get to byte alignment relative to the start of the element */	
SubElementCount	Plex(8)
for(n = 0; n < SubElementCount; n ++){	
IAElement	
}	
} /* end of IAFrame */	

9.2 BedDefinition Syntax

The BedDefinition element contains metadata and pointers to audio essence to generate one frame of one audio Bed. The BedDefinition element shall have the structure shown in Table 6. The following elements shall be supported as Children of a BedDefinition element: BedDefinition and BedRemap. Other Child elements shall be ignored.

Table 6 — BedDefinition Syntax.

BedDefinition Syntax	symbol length
BedDefinition	
{	
MetaID	Plex(8)
ConditionalBed	1
if(ConditionalBed == 1){	
BedUseCase	8
}	
ChannelCount	Plex(4)
for(n = 0; n < ChannelCount; n ++){	
ChannelID[n]	Plex(4)
AudioDataID[n]	Plex(8)
ChannelGainPrefix	2
if(ChannelGainPrefix > 1){	
ChannelGain[n]	10
}	
ChannelDecorInfoExists	1
if(ChannelDecorInfoExists == 1){	
Reserved (set to 0)	4
ChannelDecorCoefPrefix	2
If(ChannelDecorCoefPrefix>1) {	
ChannelDecorCoef[n]	8
}	
}	
}	
Reserved (set to 0x180)	10
/* reads extra bits to get to byte alignment relative to the start of the element*/	
AlignBits(set to 0)	VARIABLE
AudioDescription	8
if(AudioDescription & 0x80){	
n = -1	
do {	
/* NULL-terminated, strict ASCII Text */	
n++	
AudioDescriptionText[n]	8
}while(AudioDescriptionText[n] != 0x00)	
}	

```
BedDefinition Syntax                                          symbol length
    SubElementCount.....................................................Plex(8)
    for(n = 0; n < SubElementCount; n ++){
        IAElement
    }


} /* end of BedDefinition */
```

9.3 BedRemap Syntax

BedRemap is an optional element that describes how to convert an audio Bed from the distribution configuration to a different playback configuration. For example, BedRemap can provide information to downmix a 9.1 Bed to a 5.1 Bed for playback in a 5.1 cinema. The BedRemap element shall have the structure shown in Table 7. The BedRemap element shall only be used as a Child element of a BedDefinition element.

On a sub block by sub block basis, BedRemap provides mixing coefficients to distribute each channel of the Parent BedDefinition (bed channel) into zero or more of the channels supported by the playback system (playback channel). For example, the BedRemap could specify that the Center Bed channel be distributed with the following coefficients for a stereo playback system: CtoL_gain = 0.7, CtoR_gain = 0.7.

There is additional processing required to convert channel signals to Loudspeaker feeds including the application of the decorrelation and ChannelGains specified in the BedDefinition element.

NOTE The iChan in BedRemap corresponds to n in the ChannelCount ‘for loop’ in the BedDefinition element.

Table 7 — BedRemap Syntax.

```
BedRemap Syntax                                               symbol length
BedRemap
{
    MetaID.....................................................Plex(8)

    RemapUseCase...............................................................8

    SourceChannels.............................................Plex(4)
    DestinationChannels........................................Plex(4)

    /* NumRemapSubBlocks is derived from the Sample Rate and Frame Rate and is
    the same as NumPanSubBlocks. See Table 23. */
    for(subBlk = 0; subBlk < NumRemapSubBlocks; subBlk ++){

        if(subBlk == 0){
            RemapInfoExists = 1
        }
        else{
            RemapInfoExists ...................................................1
        }

        if(RemapInfoExists == 1){
            for(oChan = 0; oChan < DestinationChannels; oChan ++){
                /* Destination channel position.  See Table 19 */
                DestinationChannelID[subBlk][oChan].................Plex(4)
```

BedRemap Syntax	symbol length

```
            for(iChan = 0; iChan < SourceChannels; iChan ++){
                RemapGainPrefix .....................................2
                if(RemapGainPrefix > 1) {
                    RemapGain[subBlk][oChan][iChan] ...................10
                }
            }
        }
    }

    /* reads extra bits to get to byte alignment relative to the start of the
    element */
    AlignBits(set to 0).....................................VARIABLE

    Reserved(set to 0).....................................Plex(8)

    }

} /* end of BedRemap*/
```

9.4 ObjectDefinition Syntax

ObjectDefinition contains metadata, most importantly position, and pointers to audio essence to generate one frame of one Audio Object. The ObjectDefinition element shall have the structure shown in Table 8. The following elements shall be supported as Child elements of the ObjectDefinition element: ObjectDefinition and ObjectZoneDefinition19. Other Child elements shall be ignored.

Table 8 — ObjectDefinition Syntax.

ObjectDefinition Syntax	symbol length

```
ObjectDefinition
{
    MetaID.....................................................Plex(8)

    AudioDataID................................................Plex(8)

    ConditionalObject..........................................1
    if(ConditionalObject == 1){
        Reserved (set to 1) ...................................1
        ObjectUseCase .........................................8
    }

    Reserved (set to 0)........................................1

    /* NumPanSubBlocks is dependent on frame rate. See Table 23 */
    for(sb = 0; sb < NumPanSubBlocks; sb ++){
        if(sb == 0){
            PanInfoExists = 1
        }
        else{
            PanInfoExists .....................................1
        }

        if(PanInfoExists == 1){
            ObjectGainPrefix ..................................2
```

ObjectDefinition Syntax	symbol length

```
        if(ObjectGainPrefix > 1) {
            ObjectGain[sb].....10
        }

        Reserved (set to 0b001).....3

        ObjectPosX[sb].....16
        ObjectPosY[sb].....16
        ObjectPosZ[sb].....16

        ObjectSnap[sb].....1
        if(ObjectSnap[sb] == 1){
            ObjectSnapTolExists.....1
            if(ObjectSnapTolExists== 1){
                ObjectSnapTolerance[sb].....12
            }

            Res2[sb] (set to 0).....1
        }

        ObjectZoneControl.....1
        if(ObjectZoneControl == 1){
            /* For each zone listed in Table 24 */
            for(n = 0; n < 9; n ++){
                ZoneGainPrefix.....2
                if(ZoneGainPrefix > 0x1){
                    ZoneGain[sb][n].....10
                }
            }
        }

        ObjectSpreadMode.....2
        if(ObjectSpreadMode == OBJECT_SPREAD_LOWREZ){
            ObjectSpread[sb].....8
        }
        else if(ObjectSpreadMode== OBJECT_SPREAD_1D){
            ObjectSpread[sb].....12
        }
        else if(ObjectSpreadMode == OBJECT_SPREAD_3D){
            ObjectSpreadX[sb].....12
            ObjectSpreadY[sb].....12
            ObjectSpreadZ[sb].....12
        }
        Reserved (set to 0).....4
        ObjectDecorCoefPrefix.....2
        if(ObjectDecorCoefPrefix > 1){
            ObjectDecorCoef[sb].....8
        }
    } /* end if(PanInfoExists) */
}

/* reads extra bits to get to byte alignment relative to the start of the
element */
AlignBits(set to 0).....VARIABLE

AudioDescription.....8
```

ObjectDefinition Syntax	symbol length

```
    if(AudioDescription & 0x80){
       n = -1
       do {
         /* NULL-terminated, strict  ASCII Text */
         n++
         AudioDescriptionText[n] .....................................8
        }while(AudioDescriptionText[n] != 0x00)
    }



    SubElementCount.............................................Plex(8)
    for(n = 0; n < SubElementCount; n ++){
        IAElement
    }

} /* end of ObjectDefinition*/
```

9.5 ObjectZoneDefinition19 Syntax

ObjectZoneDefinition19 is an optional Sub-Element of an ObjectDefinition element. It describes an alternative zone definition that is associated with a different set of zones. An ObjectZoneDefinition19 element shall have the structure shown in Table 9.

Table 9 — ObjectZoneDefinition19 Syntax.

ObjectZoneDefinition19 Syntax	word size

```
ObjectZoneDefinition19
{
    /* NumPanSubBlocks is dependent on frame rate. See Table 23. */

    for(sb = 0; sb < NumPanSubBlocks; sb ++){
      if (sb == 0){
          ZoneInfoExists = 1
        }
        else {
          ZoneInfoExists ..................................................... 1
        }
       if(ZoneInfoExists == 1){

       /* For each zone listed in Table 28 */
       for(n = 0; n < 19; n ++){
           ZoneGainPrefix.....................................................2
           if(ZoneGainPrefix > 0x1){
               ZoneGain19[sb][n].....................................10
           }
         }
       }
    }
    /* reads extra bits to get to byte alignment relative to the start of the
    element*/

     AlignBits(set to 0)..................................... VARIABLE

} /* end of ObjectZoneDefinition19*/
```

9.6 AudioDataDLC Syntax

Each AudioDataDLC element contains losslessly-coded audio for one frame of one monaural audio waveform. AudioDataDLC elements are referenced by the audio Bed and Audio Object elements. An AudioDataDLC element shall have the structure shown in Table 10. Audio essence carried in this element shall be encoded according to the coder specification in Annex B.

Table 10 — AudioDataDLC Syntax.

AudioDataDLC Syntax	symbol length
AudioDataDLC	
{	
AudioDataID	Plex(8)
DLCSize	16
DLCSampleRate	2
ShiftBits	5
/* Predictor information */	
NumPredRegions48	2
for(n = 0; n < NumPredRegions48; n ++) {	
RegionLength48[n]	4
Order48[n]	5
for(m = 1; m <= Order48[n]; m ++) {	
KCoeff48[n][m]	10
}	
}	
/* Coded residual */	
for(n = 0; n < NumDLCSubBlocks; n ++) {	
CodeType	1
if(CodeType == 0) {	
/* PCM Residual */	
BitDepth	5
for(i= 0; i< DLCSubBlockSize48; i++) {	
if (BitDepth == 0) {	
Residual48[n * DLCSubBlockSize48 + i] = 0	
}	
else {	
Residual48[n * DLCSubBlockSize48 + i]	BitDepth
}	
if(Residual48[n * DLCSubBlockSize48 + i] != 0){	
sign	1
if(sign == 1){	
Residual48[n * DLCSubBlockSize48 + i] *= -1	
}	
}	
}	
}	
else {	
/*Rice/Golomb Residual */	
RiceRemBits	5
for(i= 0; i< DLCSubBlockSize48; i++) {	
quotient = 0	

AudioDataDLC Syntax	symbol length

```
            UnaryBit.....................................................1
            while(UnaryBit == 1){
                quotient ++
                UnaryBit ................................................1
            }
            if (RiceRemBits == 0) {
              Residual48[n * DLCSubBlockSize48 + i] = 0
            }
            else {
              Residual48[n * DLCSubBlockSize48 + i].........RiceRemBits
            }
            Residual48[n * DLCSubBlockSize48 + i]  += quotient  <<
            RiceRemBits

            if(Residual48[n * DLCSubBlockSize48 + i]  !=  0){
                sign ....................................................1
                if(sign == 1){
                    Residual48[n * DLCSubBlockSize48 + i]  *= -1
                }
            }
        }
    }
}
/* Information for 96 kHz extension layer */
if(SampleRate == 0x1){
    /* Predictor information */
    NumPredRegions96 ..............................................2
    for(n = 0; n < NumPredRegions96; n ++ ) {
        RegionLength96[n] .....................................4
        Order96[n] ..................................................5
        for(m = 1; m <= Order96[n]; m ++)  {
            KCoeff96[n][m].....................................10
        }
    }

    /* Coded residual */
    for(n = 0; n <  NumDLCSubBlocks; n ++)  {
        CodeType ..................................................1

        if(CodeType == 0)  {
            /* PCM Residual */
            BitDepth.....................................................5
            for(i= 0; i< DLCSubBlockSize96; i++) {
               if (BitDepth == 0) {
                Residual96[n * DLCSubBlockSize96 + i] = 0
              }
              else {
                Residual96[n * DLCSubBlockSize96 + i ..........BitDepth
              }
               if(Residual96[n * DLCSubBlockSize96 + i]  !=  0){
                   sign ..........................................1
                   if(sign == 1){
                       Residual96[n * DLCSubBlockSize96 + i]  *= -1
                   }
               }
            }
```

AudioDataDLC Syntax	symbol length

```
            }
            else {
                /*Rice/Golomb Residual */
                RiceRemBits.....................................................5
                for(i= 0; i< DLCSubBlockSize96; i++) {
                    quotient = 0
                    UnaryBit .....................................................1
                    while(UnaryBit == 1){
                        quotient ++
                        UnaryBit .................................................1
                    }
                    if (RiceRemBits == 0) {
                      Residual96[n * DLCSubBlockSize96 + i] = 0
                    }
                     else {

                      Residual96[n * DLCSubBlockSize96 + i] ......RiceRemBits
                      Residual96[n * DLCSubBlockSize96 + i]  += quotient  <<
                      RiceRemBits

                      if(Residual96[n * DLCSubBlockSize96 + i]  !=  0){
                        sign .....................................................1
                        if(sign == 1){
                            Residual96[n * DLCSubBlockSize96 + i]  *= -1
                        }
                      }
                    }
                }
            }
        }
    }

    /* Each element must keep track of the number of bits read */
    AlignBits(set to 0) ...........................................VARIABLE

} /* end of AudioDataDLC*/
```

9.7 AudioDataPCM Syntax

Each AudioDataPCM element contains linear PCM audio essence for one frame of one monaural audio waveform. AudioDataPCM elements are referenced by the audio Bed and Audio Object elements. The AudioDataPCM element shall have the structure shown in Table 11. This element shall contain linear PCM audio essence.

Table 11 — AudioDataPCM Syntax.

Syntax	symbol length
AudioDataPCM	
{	
AudioDataID.....Plex(8)	
/* SampleCount is a function of frame rate and sample rate as indicated in Table 18 */	
for(n = 0; n < SampleCount; n++){	
PCMData[n].....16 or 24	
}	
/* Each element must keep track of the number of bits read */	
AlignBits(set to 0).....VARIABLE	
} /* end of AudioDataPCM*/	

9.8 AuthoringToolInfo Element

The AuthoringToolInfo element identifies the vendor and tool (including version) with which the Immersive Audio Frame was created. The AuthoringToolInfo element shall have the structure shown in Table 12. Decoders may skip this element.

Table 12 — AuthoringToolInfo Syntax.

AuthoringToolInfo Syntax	symbol length
AuthoringToolInfo	
{	
n = -1	
do {	
/* NULL-terminated, strict ASCII Text */	
n++	
AuthoringToolURI[n].....8	
}while(AuthoringToolURI[n] != 0x00)	
} /* end of AuthoringToolInfo */	

9.9 UserData Element

The UserData element carries undefined user data that is identified using a SMPTE UL. The UserData element shall have the structure shown in Table 13. Decoders may skip this element.

Table 13 — UserData Syntax.

UserData Syntax	symbol length
UserData	
{	
UserID.....	128
UserDataBytes.....	VARIABLE
} /* end of UserData */	

10 IAFrame Data Fields

10.1 to 10.10 define the bit fields referenced by name in Clause 9. If a bit field is not listed in a clause, it is because it is defined in an earlier clause and the definition remains the same. Data fields are defined in 10.1 to 10.10, and each field name is followed by the number of bits used by the field.

10.1 IAElement Data Fields

10.1.1 ElementID – Plex(8)

Each Element block shall start with an ElementID. The ElementID shall indicate the type of element. Depending on the ElementID the decoder will perform different tasks. A decoder shall skip ElementIDs not defined in this standard. Table 14 provides a list of defined ElementIDs and their meaning.

Table 14 — ElementIDs.

Value	ElementID Name	Meaning
0x08	IA_FRAME	Frame header
0x10	BED_DEFINITION	Bed definition
0x20	BED_REMAP	Bed remap
0x40	OBJECT_DEFINITION	Object definition
0x80	OBJECT_ZONE_DEFINITION19	Extended object zone definition
0x100	AUTHORING_TOOL_INFO	Authoring tool information
0x101	USER_DATA	User defined data
0x200	AUDIO_DATA_DLC	Audio data (DLC encoded)
0x400	AUDIO_DATA_PCM	Audio data PCM
All other values		Reserved

10.1.2 ElementSize – Plex(8)

ElementSize shall indicate the size in bytes of the entire element, not including the ElementID and ElementSize.

For the IAFrame Element, the ElementSize shall be the size in bytes of the entire IAFrame, not including the IAFrame ElementID and ElementSize, as all other elements are contained as Sub-Elements.

10.2 IAFrame Data Fields

10.2.1 Version – 8 bits

The Version shall specify the version of the Immersive Audio Bitstream. Bitstreams complying with this specification shall have a value of 1 for the Version. The values of 0 and 2 are forbidden and shall not be used for future revisions.

10.2.2 SampleRate – 2 bits

The SampleRate code shall specify the sampling rate of the audio data. All audio, channels and objects, shall be coded using the same sampling rate. The SampleRate code has following definitions as shown in Table 15. The DLCSampleRate field of AudioDataDLC shall be the same as this field.

Table 15 — SampleRate Code.

SampleRate Code	Meaning
0x0	48000 samples per second
0x1	96000 samples per second
0x2	Reserved
0x3	Reserved

10.2.3 BitDepth – 2 bits

The BitDepth code shall specify the bit depth of the audio data. All audio shall be coded using the same bit depth. All audio samples ‘a’ within the IAFrame shall be in the range $-2^{n-1} <= a < 2^{n-1}$ where n is the bit depth. The BitDepth code has the following meanings as specified in Table 16.

Table 16 — BitDepth Code.

BitDepth Code	Meaning
0x0	16 bits per audio sample
0x1	24 bits per audio sample
0x2	Reserved
0x3	Reserved

10.2.4 FrameRate – 4 bits

The FrameRate code shall specify the IAFrame Rate. The FrameRate code value and meaning are specified in Table 17.

Table 17 — FrameRate Code.

FrameRate Code	Meaning
0x0	24 frames per second
0x1	25 frames per second
0x2	30 frames per second
0x3	48 frames per second
0x4	50 frames per second
0x5	60 frames per second
0x6	96 frames per second
0x7	100 frames per second
0x8	120 frames per second
0x9	24000/1001 frames per second
0xA-0xF	Reserved

The FrameRate code also controls the sample count (SampleCount) contained in each audio asset as specified in Table 18.

Table 18 — Sample Count versus Frame Rate Code and Sample Rate.

FrameRate Code	SampleCount48	SampleCount96
0x0	2000	4000
0x1	1920	3840
0x2	1600	3200
0x3	1000	2000
0x4	960	1920
0x5	800	1600
0x6	500	1000
0x7	480	960
0x8	400	800
0x9	2002	4004
0xA-0xF	Reserved	Reserved

10.2.5 MaxRendered – Plex(8)

The MaxRendered value shall indicate the maximum number of Bed channels and objects (i.e., ObjectDefinition elements), including silent channels and objects, that could be rendered during the playback of the frame considering all UseCases. For example, for an IAFrame element with a 9.1 channel unconditional Bed and 118 unconditional objects, the MaxRendered count would be set to 128 for that IAFrame.

Multiple BedDefinition elements may be rendered at one time, and in this case the MaxRendered value shall be set by taking into account the maximum number of Bed channels that can be rendered by these multiple BedDefinition elements. If multiple rendered BedDefinition elements share one or more channels, then such shared channels shall be counted as many times as each is present for the purpose of obtaining the MaxRendered value. For example, if two rendered BedDefinition elements share one or more channels (e.g., the center channel and left channel) then such channels are to be counted twice for the purposes of obtaining the MaxRendered value.

If the SourceChannels field value of a BedRemap element is different from its DestinationChannels field value, the larger of these two values shall be used when computing the MaxRendered value.

See Annex A for examples of MaxRendered.

10.2.6 SubElementCount – Plex(8)

The SubElementCount code is the number of Children contained in the current element.

10.3 BedDefinition Data Fields

10.3.1 MetaID – Plex(8)

MetaID is the ID that allows the system to track metadata information between audio frames. Within an IAFrame, no two elements with the same ElementID shall have the same MetaID. Elements with the same ElementID and MetaID in contiguous IAFrames typically represent continuous audio.

10.3.2 ConditionalBed – 1 bit

The ConditionalBed code is a Boolean. A value of FALSE(0) indicates that the BedDefinition is Activated. A value of TRUE(1) indicates that additional BedUseCase metadata follows this code and the BedDefinition is Activated if and only if this UseCase matches the Target Environment or is set to Always Use.

The bitstream shall be constructed such that: of the Child elements of a BedDefinition, at most one BedDefinition or BedRemap Child element is Activated. A BedDefinition element shall be rendered if and only if (a) it is Activated, (b) either it has no Ancestor BedDefinition or all its Ancestor BedDefinition elements are Activated, and (c) it does not have a Child BedDefinition that is Activated. If a BedDefinition element is rendered and has a Child BedRemap element that is Activated, then this BedRemap element shall be applied to it.

10.3.3 BedUseCase – 8 bits

The BedUseCase code indicates the Target Environments for which this element shall be Activated. The supported UseCase codes are the UseCase codes defined in Table 1. UseCase codes other than those listed in Table 1 shall be ignored.

10.3.4 ChannelCount – Plex(4)

The channel count shall indicate the number of channels in the bed, including silent channels.

10.3.5 ChannelID[n] – Plex(4)

The ChannelID code shall uniquely identify a channel within a Bed along with its routing destination. Channels carried in the bitstream provide the same function as those defined in SMPTE ST 428-12: 2013, ITU-R BS.2051-2, and SMPTE ST 2098-5: 2018. The routing destination shall be as defined in those documents and referenced by the Channel Name parameter. Table 19 lists ChannelID codes along with the Channel Name reference from those documents. A ChannelID shall not be indicated more than once within a BedDefinition element. Values of ChannelID code that are not represented in Table 19 are Reserved.

NOTE ChannelID provides the Routing Destination function defined in SMPTE ST 2098-1: 2018. The combination of ChannelID with MetaID provides the Channel Identifier function defined in SMPTE ST 2098-1 2018.

The set of ChannelIDs present in a BedDefinition element shall be a subset of, or the whole set of, a Soundfield Group or sound system as specified in Table 1.

Table 19 — ChannelID and DestinationChannelID Codes.

ChannelID and DestinationChannelID Codes	SMPTE ST 428-12:2013 Channel Name	SMPTE ST 2098-5:2018 Channel Name	ITU-R BS.2051-2 Channel Name / System
0x0	Left		
0x1	Left Center		
0x2	Center		
0x3	Right Center		
0x4	Right		
0x5	Left Side Surround		
0x6	Left Surround		
0x7	Left Rear Surround		
0x8	Right Rear Surround		
0x9	Right Side Surround		
0xA	Right Surround		
0xB		Left Top Surround	
0xC		Right Top Surround	
0xD	LFE		
0xE		Left Height	
0xF		Right Height	
0x10		Center Height	
0x11		Left Surround Height	
0x12		Right Surround Height	
0x13		Left Side Surround Height	

ChannelID and DestinationChannelID Codes	SMPTE ST 428-12:2013 Channel Name	SMPTE ST 2098-5:2018 Channel Name	ITU-R BS.2051-2 Channel Name / System
0x14		Right Side Surround Height	
0x15		Left Rear Surround Height	
0x16		Right Rear Surround Height	
0x17		Top Surround	
0x18-0x7F	Reserved for D-Cinema		
0x80			Left Top Front / J
0x81			Right Top Front / J
0x82			Left Top Back / J
0x83			Right Top Back / J
0x84			Top side left / H
0x85			Top side right / H
0x86			LFE1 / H
0x87			LFE2 / H
0x88			Front Left (Wide) / H
0x89			Front Right (Wide) / H

10.3.6 AudioDataID[n] – Plex(8)

The AudioDataID code shall identify the associated mono audio asset carried in the bitstream. An AudioDataID field value of 0 shall indicate no audio asset. A bitstream encoder may choose to use an AudioDataID of zero to represent a mono asset with silent audio.

10.3.7 ChannelGainPrefix – 2 bits

The ChannelGainPrefix code shall specify the method for coding the channel gain values. The meaning of the prefix code is shown in Table 20.

Table 20 — Gain Prefix Code.

Prefix Code	Gain
0x0	Set gain to 1.0
0x1	Set gain to 0
0x2	Gain Code follows in the bitstream. Set gain based on Gain Code.
0x3	Reserved

10.3.8 ChannelGain[n] – 10 bits

The ChannelGain code shall specify the gain to be applied to each Bed channel. The Channel gain for each channel is converted to a value between 0.0 and 1.0 as described in 5.5.

10.3.9 ChannelDecorInfoExists – 1 bit

The ChannelDecorInfoExists code is a Boolean value; if the code is set to 1, then a ChannelDecorCoef code follows. If the code is set 0 then no decorrelation shall be applied to the Bed channel's audio.

10.3.10 ChannelDecorCoefPrefix – 2 bits

The ChannelDecorCoefPrefix code shall specify the method for coding the decorrelation values. The meaning of the prefix code is shown in Table 21.

Table 21 — ChannelDecorCoefPrefix Code.

Prefix Code	Meaning
0x0	No decorrelation
0x1	Maximum decorrelation
0x2	Decorrelation coefficient follows in the bitstream.
0x3	Reserved

10.3.11 ChannelDecorCoef[n] – 8 bits

During object and channel rendering it is possible for a single audio asset to be routed or panned to multiple Loudspeakers. The decorrelation parameter indicates the degree of decorrelation desired between multiple output signals derived from a single audio waveform. The ChannelDecorCoef code shall specify the amount of decorrelation to be applied to the Bed channel's audio asset. ChannelDecorCoef = 0 means no decorrelation. ChannelDecorCoef = 255 means maximum decorrelation. The decorrelation algorithm is out of scope of this specification.

10.3.12 AudioDescription – 8 bits

The AudioDescription code provides a description of the contents of the audio. The AudioDescription code is a bit field with bits assigned as shown in Table 22. Hence audio contents can be described as combinations of elemental types. There is an exception to this in the case of the presence of 1 in the least significant bit (bit 0). In such a case, bits 1-6 shall be ignored. If the most significant bit is a 1, then a NULL terminated ASCII text field (AudioDescriptionText) shall follow.

Table 22 — AudioDescription Code.

AudioDescription bit mask	Meaning
0x01	Not Indicated
0x02	Dialog
0x04	Music
0x08	Effects
0x10	Foley
0x20	Ambience
0x40	Reserved
0x80	Additional Information transmitted as a NULL terminated text string

10.3.13 AudioDescriptionText[n] - 8 bits

AudioDescriptionText is present if the most significant bit of AudioDescription is set. AudioDescriptionText is a null terminated ASCII text string with each character contained in an 8-bit byte with the most significant bit clear.

10.4 BedRemap Data Fields

10.4.1 RemapUseCase – 8 bits

The RemapUseCase code indicates the Target Environment for which this element shall be Activated. The supported UseCase codes are the UseCase codes defined in Table 1. UseCase codes other than those listed in Table 1 shall be ignored.

10.4.2 SourceChannels – Plex(4)

SourceChannels shall equal the number of channels contained in the Bed specified in the Parent BedDefinition element.

10.4.3 DestinationChannels – Plex(4)

DestinationChannels shall equal the number of output channels defined by the remap matrix.

10.4.4 DestinationChannelID[subBlk][oChan] – Plex(4)

DestinationChannelID is the ChannelID that indicates the output of a specific BedRemap as shown in Table 7.

Each DestinationChannelID value shall be unique within each BedRemap sub block. Table 19 lists DestinationChannelID codes along with the Channel Name reference from those documents.

The set of DestinationChannelIDs present in a BedRemap element shall be a subset of, or the whole set of, a Soundfield Group or sound system as specified in Table 1.

10.4.5 RemapGainPrefix – 2 bits

The RemapGainPrefix code shall specify the method for coding the RemapGain values. The meaning of the prefix code is shown in Table 20.

10.4.6 RemapGain[subBlk][oChan][iChan] – 10 bits

The RemapGain code shall specify the gain to be applied to each Bed channel. The channel gain for each channel is converted to a value between 0.0 and 1.0 as described in 5.5.

10.5 ObjectDefinition Data Fields

10.5.1 ConditionalObject – 1 bit

The ConditionalObject code is a Boolean. A value of FALSE(0) indicates that the ObjectDefinition is Activated. A value of TRUE(1) indicates that additional ObjectUseCase metadata follows this code and the ObjectDefinition is Activated if and only if this UseCase matches the Target Environment or is set to Always Use.

The bitstream shall be constructed such that: of the Child elements of an ObjectDefinition, at most one ObjectDefinition Child element is Activated. An ObjectDefinition element shall be rendered if and only if (a) it is Activated, (b) either it has no Ancestor ObjectDefinition element or all its Ancestor ObjectDefinition elements are Activated, and (c) it does not have a Child ObjectDefinition element that is Activated.

10.5.2 ObjectUseCase – 8 bits

The ObjectUseCase code indicates the Target Environment for which this element shall be Activated. The supported UseCase codes are the UseCase codes defined in Table 1. UseCase codes other than those listed in Table 1 shall be ignored.

10.5.3 NumPanSubBlocks (not a bitstream symbol)

To provide sufficient time resolution for object metadata parameters, the time interval corresponding to one IAFrame is divided into sub blocks. In the case of the 24000/1001 frame rate, the sub block sizes are listed in the order they will appear in the bitstream. The NumPanSubBlocks parameter shall be derived from the IAFrame Rate, as specified in Table 23.

NOTE The duration associated with each sub block is also indicated.

Table 23 — Number of Sub Blocks versus Frame Rate.

IAFrame Rate (frames per second)	Number of Sub Blocks per IAFrame	48 kHz Samples per Sub Block	96 kHz Samples per Sub Block	Sub Block Duration (ms)
24000/1001	8	251, 250, 250, 250, 251, 250, 250, 250	501, 500, 501, 500, 501, 500, 501, 500	5.2
24	8	250	500	5.2
25	8	240	480	5.0
30	8	200	400	4.2
48	4	250	500	5.2
50	4	240	480	5.0
60	4	200	400	4.2
96	2	250	500	5.2
100	2	240	480	5.0
120	2	200	400	4.2

10.5.4 PanInfoExists – 1 bit

Certain "high-rate" panning information can be updated on a sub block basis. This panning information shall always be provided for the first sub block, which shall not have a PanInfoExists bit. For the remaining sub blocks the PanInfoExists bitstream parameter shall indicate the presence of a panning information update in the bitstream. A value of 1 indicates that the panning information is present; a value of 0 indicates that new panning information does not exist for this sub block, and the previous panning information remains valid.

10.5.5 ObjectGainPrefix – 2 bits

The ObjectGainPrefix code shall specify the method for coding the object gain values. The meaning of the prefix code is shown in Table 20.

10.5.6 ObjectGain[sb] – 10 bits

The ObjectGain code shall specify the gain to be applied to the audio essence of the panned Audio Object. The coding follows the standard gain coding mechanism described in 5.5.

10.5.7 ObjectPosX[sb], ObjectPosY[sb], ObjectPosZ[sb] – 16 bits

The ObjectPosX, ObjectPosY, and ObjectPosZ codes shall specify the 3-dimensional position of a panned Audio Object. The ObjectPos field is encoded using the position coding method described in 11.1.

10.5.8 ObjectSnap[sb] – 1 bit

The ObjectSnap[sb] code is a Boolean value. A value of 1 shall indicate that object timbre is more important than precise location. If ObjectSnap[sb] is set to 1, then additional information shall follow.

10.5.9 ObjectSnapTolExists – 1 bit

The ObjectSnapTolExists code is a Boolean value; if it is set to 1 then the ObjectSnapTolerance[sb] code shall follow. If the ObjectSnapTolExists code is set to 0; then the ObjectSnapTolerance[sb] shall be set to DEFAULT_OBJ_SNAP_TOL defined in 11.2.

10.5.10 ObjectSnapTolerance[sb] – 12 bits

The ObjectSnapTolerance[sb] code shall specify the maximum acceptable location displacement between the desired object location and the target “snap-to” Loudspeaker. The displacement shall be computed as the maximum of the absolute differences in each of the 3 dimensions between the object location and the Loudspeaker location. The ObjectSnapTolerance[sb] code maps to the range 0.0 to 1.0 in unit room distance using the DistanceZ coding method described in 5.4. For example, if the tolerance maps to a value of 0.5, then the object shall be snapped only if the displacement is less than 0.5 unit distance in all three dimensions. If the tolerance maps to 1.0, the maximum value, then tolerance is effectively disabled, and the object will always snap.

10.5.11 Zone Definition

Zones are collections of Loudspeakers, defined by regions within a cinema. Zones are identified by their order in the bitstream. Zones are described in Table 24. Zone Definition data may be superseded by ObjectZoneDefinition19 zones, if present.

Table 24 — Zone Definition.

Description
All screen Loudspeakers left of center
Screen center Loudspeakers
All screen Loudspeakers right of center
All Loudspeakers on left wall
All Loudspeakers on right wall
All Loudspeakers on left half of rear wall
All Loudspeakers on right half of rear wall
All overhead Loudspeakers left of center
All overhead Loudspeakers right of center

10.5.12 ObjectZoneControl – 1 bit

The ObjectZoneControl is a Boolean value; if the value is set to 0, zone control is not used. If present, the control information shall be included in the bitstream for every zone listed in Table 24, in the order listed in Table 24.

10.5.13 ZoneGainPrefix – 2 bits

The ZoneGainPrefix code shall specify the method for coding the zone gain values. The meaning of the prefix code is shown in Table 25. ZoneGainPrefix shall be associated with a Zone according to the order it appears in the bitstream.

Table 25 — ZoneGainPrefix Code.

Prefix Code	Gain
0x0	Set gain to 0.0
0x1	Set gain to 1.0
0x2	ZoneGain Code follows in the bitstream. Set gain based on ZoneGain Code.
0x3	Reserved

10.5.14 ZoneGain[sb][n] – 10 bits

The ZoneGain code shall specify the degree to which Loudspeakers within the zone shall contribute to the rendering of the object. ZoneGain shall only be included in the bitstream if the ZoneGainPrefix is greater than 1. The code is converted to a value ranging between 0.0 and 1.0 using the formula: gain = ZoneGain/(2^{10}-1)

10.5.15 ObjectSpreadMode – 2 bits

The ObjectSpreadMode code shall specify the type of spreading information that will follow in the bitstream as shown in Table 26.

Table 26 — ObjectSpreadMode Code.

Code	Identifier	Description
0x0	OBJECT_SPREAD_LOWREZ	Equal spreading in each dimension using 8-bit coding
0x1	OBJECT_SPREAD_NONE	Point source (no ObjectSpread value is sent)
0x2	OBJECT_SPREAD_1D	Equal spreading in each dimension using 12-bit coding
0x3	OBJECT_SPREAD_3D	Specified spreading in each dimension

10.5.16 ObjectSpread[sb] – 8 or 12 bits

The ObjectSpread[sb] code shall specify the amount of spread to be applied to an object and corresponds to the extent of spread in each direction (X, Y, or Z). The code is converted to a value ranging between 0.0 and 1.0 using the DistanceZ coding method described in 5.4. ObjectSpread is the spread of the object in each dimension; i.e., the spread corresponds to +/- 0.5 * ObjectSpread from the object's position.

10.5.17 ObjectSpreadX[sb], ObjectSpreadY[sb], ObjectSpreadZ[sb] – 12 bits

The ObjectSpreadX-Y-Z[sb] code shall specify the amount of spread to be applied to an object and corresponds to the extent of spread in each direction. The code is converted to a value ranging between 0.0 and 1.0 using the DistanceZ coding method described in 5.4. ObjectSpread is the spread of the object in each dimension; i.e., the spread corresponds to +/- 0.5 * ObjectSpread from the object's position.

10.5.18 ObjectDecorCoefPrefix – 2 bits

The ObjectDecorCoefPrefix code shall specify the method for coding the decorrelation gain values. The meaning of the prefix code is shown in Table 27.

Table 27 — ObjectDecorCoefPrefix Code.

Code	Decorrelation Coefficient
0x0	No decorrelation
0x1	Maximum decorrelation
0x2	Decorrelation coefficient follows in the bitstream.
0x3	Reserved

10.5.19 ObjectDecorCoef[sb] – 8 bits

During object and channel rendering, it is possible for a single audio asset to be routed or panned to multiple Loudspeakers. The decorrelation parameter indicates the degree of decorrelation desired between multiple output signals derived from a single audio waveform. The ObjectDecorCoef code shall specify the amount of decorrelation to be applied to the object's audio asset. ObjectDecorCoef = 0 means no decorrelation. ObjectDecorCoef = 255 means maximum decorrelation. The decorrelation algorithm is out of scope of this specification.

10.6 ObjectZoneDefinition19 Data Fields

The ObjectZoneDefinition19 element, if recognized by the decoder, shall be associated with the zones listed in Table 28. A decoder shall use this element if it supports the control zones described in Table 28. If used, the ZoneGainPrefix and ZoneGain values in the ObjectZoneDefinition19 element shall replace the ZoneGainPrefix and ZoneGain values in the ObjectDefinition element. The zone gain values provided by the ObjectZoneDefinition19 element shall be associated with the zones defined in Table 28.

Table 28 — Zone Definition of the ObjectZoneDefinition19 element.

Description
Base layer screen Loudspeakers left of center
Base layer center screen Loudspeakers
Base layer Loudspeakers right of center
Height layer screen Loudspeakers left of center
Height layer center screen Loudspeakers
Height layer screen Loudspeakers right of center
Base layer rear wall Loudspeakers left of center
Base layer rear wall center Loudspeakers
Base layer rear wall Loudspeakers right of center

Description
Height layer rear wall Loudspeakers left of center
Height layer rear wall center Loudspeakers
Height layer rear wall Loudspeakers right of center
Base layer left wall Loudspeakers
Height layer left wall Loudspeakers
Base layer right wall Loudspeakers
Height layer right wall Loudspeakers
Ceiling Loudspeakers left of center
Center ceiling Loudspeakers
Ceiling Loudspeakers right of center

10.6.1 ZoneInfoExists[sb] – 1 bit

ZoneInfo can be updated on a sub block basis. A ZoneGainPrefix for each zone listed in Table 28, in the order shown in Table 28, shall always be provided for the first sub block, which shall not have a ZoneInfoExists bit. For the remaining sub blocks, the ZoneInfoExists parameter shall indicate the presence of a zone gain information update in the bitstream. A value of 1 indicates that the zone gain information is present; a value of 0 indicates that new zone gain information does not exist for this sub block, and the previous zone gain information remains valid.

10.6.2 ZoneGainPrefix – 2 bits

The ZoneGainPrefix code shall specify the method for coding the zone gain values. The meaning of the prefix code is shown in Table 25. ZoneGainPrefix shall be associated with a Zone according to the order it appears in the bitstream.

10.6.3 ZoneGain19[sb][n] – 10 bits

The ZoneGain19 code shall specify the degree to which Loudspeakers within the zone shall contribute to the rendering of the object. The code is converted to a value ranging between 0.0 and 1.0 using the formula: gain = ZoneGain19/(2^{10}-1). ZoneGain19 shall be associated with a Zone according to the order it appears in the bitstream.

10.7 AudioDataDLC Data Fields

The AudioDataDLC element carries losslessly-coded audio essence. The coding algorithm and derivation of the fields specified in 10.7.1 to 10.7.20 are explained in Annex B. AudioDataDLC elements shall not be present in IAFrames with non-integer frame rates.

10.7.1 AudioDataID – Plex(8)

AudioDataID shall identify this instance of a mono audio essence. AudioDataID shall be unique across all audio essence types within an IAFrame. The value of the AudioDataID field of the AudioDataDLC element or the AudioDataPCM element shall not be 0.

An AudioDataID code that is not referenced by any BedDefinition or ObjectDefinition elements, if present within an AudioDataDLC element or within an AudioDataPCM element, shall not be counted as a part of MaxRendered and shall be ignored.

10.7.2 DLCSize – 16

DLCSize shall indicate the size in bytes of the remainder of the AudioDataDLC element, or equivalently, the size of the entire element, not including ElementID, ElementSize, AudioDataID, and DLCSize.

10.7.3 DLCSampleRate – 2 bits

The DLCSampleRate shall specify the sample rate of the audio contained in the AudioDataDLC element. The meaning of the code is shown in Table 29.

Table 29 — DLCSampleRate code.

DLCSampleRate Code	Meaning
0x0	48000 samples per second
0x1	96000 samples per second
0x2	Reserved
0x3	Reserved

10.7.4 ShiftBits – 5 bits

The ShiftBits code shall specify the number of least significant bits that shall be added to each output audio sample in the decoder.

10.7.5 NumPredRegions48 – 2 bits

The NumPredRegions48 code shall specify the number of predictor regions within an audio frame for the 48 kHz sample rate audio. The code shall be interpreted as an unsigned integer. A value of 0 shall indicate that the predictor is disabled. Each predictor region is a group of predictor region blocks as indicated by the parameter RegionLength48.

10.7.6 RegionLength48[n] – 4 bits

The RegionLength48 code shall specify the number of predictor region blocks used in the relevant predictor region. RegionLength48 is an unsigned integer greater than 0. The sum of RegionLength48 codes for all predictor regions in the 48 kHz part of an AudioDataDLC must equal NumDLCSubBlocks (see 10.7.9) except when NumPredRegions48 is 0.

10.7.7 Order48[n] – 5 bits

The Order48[n] code shall specify the filter order for the predictor for the n^{th} predictor region of the 48 kHz sample rate audio. A value of 0 shall indicate the filter is not used in the n^{th} region.

10.7.8 KCoeff48[n][m] – 10 bits

The KCoeff48[n][m] code is a 10-bit code that shall specify a lattice prediction coefficient for the n^{th} region and m^{th} coefficient index for the 48 kHz sample rate audio. The KCoeff48 values shall be represented as unsigned integers ranging from 0 to 1023 inclusive. The conversion to direct-form prediction coefficients and the application of the prediction coefficients to the residual data is shown in Annex B.

10.7.9 NumDLCSubBlocks, DLCSubBlockSize48, DLCSubBlockSize96 (not carried in the bitstream)

The number of sub blocks (NumDLCSubBlocks) and sub block size (DLCSubBlockSize48 / DLCSubBlockSize96) shall be determined by the IAFrame Rate as shown in Table 30.

Table 30 — Number of Sub Blocks and Sub Block Size versus Sample Rate and IAFrame Rate.

IAFrame Rate	NumDLCSubBlocks	DLCSubBlockSize48 (samples)	DLCSubBlockSize96 (samples)
24	10	200	400
25	10	192	384
30	8	200	400
48	5	200	400
50	5	192	384
60	4	200	400
96	5	100	200
100	4	120	240
120	4	100	200

10.7.10 CodeType – 1 bit

The CodeType code shall specify the entropy coding technique applied to the residual signal as specified in Table 31.

Table 31 — CodeType Code.

Code	Residual Coding Type
0x0	Direct PCM
0x1	Rice/Golomb Coding

10.7.11 BitDepth – 5 bits

If direct PCM is used to transmit the residual signal, then the BitDepth code shall specify the number of bits used to represent the magnitude of the PCM data in the sub block.

10.7.12 Residual48[n * DLCSubBlockSize + i] – Variable

The Residual48[n * DLCSubBlockSize + i] data shall contain the residual signal (error signal after prediction) for the 48 kHz audio. If the CodeType value is 0 (Direct PCM) then the magnitude of the residual shall be binary coded with BitDepth bits. If the CodeType value is 1, (Rice/Golomb coding), then the magnitude of the residual signal shall be coded as the combination of a unary coded quotient and a value binary coded with RiceRemBits. In both transmission cases, for non-zero magnitude values, the Sign bit shall be transmitted to specify the final sign of the residual signal.

10.7.13 Sign – 1 bit

The Sign code is a single bit code that shall specify the final sign of the residual signal. A value of 0 shall specify the residual is positive and a value of 1 shall specify the residual is negative.

10.7.14 RiceRemBits – 5 bits

If Rice/Golomb coding is used to transmit the residual signal, then the RiceRemBits code shall specify the number of bits used to code the remainder bits (bypass bits) after the unary coded quotient.

10.7.15 UnaryBit – 1 bit

The UnaryBit shall specify the unary encoded quotient of the residual signal. A UnaryBit code of 1 shall indicate an increment in the quotient. A UnaryBit code of 0 shall terminate the unary coding of the quotient.

10.7.16 NumPredRegions96 – 2 bits

The NumPredRegions96 code shall specify the number of predictor regions within an audio frame for the 96 kHz sample rate extension audio (if present). The code shall be interpreted as an unsigned integer. A value of 0 shall indicate that the predictor is disabled. Each predictor region is a group of predictor region blocks as indicated by the parameter RegionLength96.

10.7.17 RegionLength96[n] – 4 bits

The RegionLength96 code shall specify the number of predictor region blocks used in the relevant predictor region. RegionLength96 is an unsigned integer greater than zero. The sum of RegionLength96 codes for all predictor regions in the 96 kHz part of an AudioDataDLC must equal NumDLCSubBlocks (see 10.7.9) except when NumPredRegions96 is zero.

10.7.18 Order96[n] – 5 bits

The Order96[n] code shall specify the filter order for the predictor for the n^{th} predictor region of the 96 kHz sample rate extension audio (if present). A value of 0 shall indicate the filter is not used in the n^{th} region.

10.7.19 KCoeff96[n][m] – 10 bits

The KCoeff96[n][m] code is a 10-bit code that shall specify a lattice prediction coefficient for the n^{th} region and m^{th} coefficient index for the 96 kHz sample rate extension audio (if present). The KCoeff96 values shall be represented as unsigned integers ranging from 0 to 1023 inclusive. The conversion to direct-form prediction coefficients and the application of the prediction coefficients to the residual data is shown in Annex B.

10.7.20 Residual96[n * DLCSubBlockSize + i] – Variable

The Residual96[n * DLCSubBlockSize + i] data shall contain the residual signal (error signal after prediction) for the 96 kHz audio. If the CodeType value is 0 (Direct PCM) then the magnitude of the residual shall be binary coded with BitDepth bits. If the CodeType value is 1, (Rice/Golomb coding), then the magnitude of the residual signal shall be coded as the combination of a unary coded quotient and a value binary coded with RiceRemBits. In both transmission cases, for non-zero magnitude values, the Sign bit shall be transmitted to specify the final sign of the residual signal.

10.8 AudioDataPCM Data Fields

10.8.1 PCMData[n] – 16 or 24 bits

The PCMData is the audio data for a frame in uncompressed PCM format. The bit depth used to specify the audio data is transmitted as part of the IAFrame element. Each PCMData value shall be encoded and transmitted as bytes in little endian format. I.e., the least significant byte — the byte containing the least significant bit — is stored (has the lowest address) or transmitted first, then the following bytes are stored or transmitted in increasing significance order, with the most significant byte — the one containing the most significant bit — stored last (having the highest address) or transmitted last. Each byte is transmitted most significant bit first. This is consistent with RIFF/WAV encoding.

10.9 AuthoringToolInfo Element

The AuthoringToolInfo element identifies the vendor and tool (including version) with which the Immersive Audio Frame was created.

10.9.1 AuthoringToolURI

AuthoringToolURI shall be a null terminated ASCII text string with each character contained in an 8-bit byte with the most significant bit clear.

The authoringToolURI shall be a Uniform Resource Identifier as defined by IETF RFC 3986 with the following constraints:

- the authority component shall be present;
- the host subcomponent shall be a DNS fully qualified domain name as specified in IETF RFC 1035;
- the host subcomponent shall be a domain name registered to the authoring tool vendor to prevent duplication between tool identifiers provided by different vendors.

10.10 UserData Element

The UserData element carries undefined user data that is identified using a SMPTE Label.

10.10.1 UserID – 128 bits

UserID shall be a SMPTE-Administered Universal Label per SMPTE ST 298: 2009 that identifies the entity creating the UserDataBytes and their definition. UserDataBytes associated with an unrecognized UserID shall be ignored.

10.10.2 UserDataBytes – Variable

UserDataBytes shall contain an integer number of data bytes. The interpretation of the data in this field is outside the scope of this specification.

11 Bitstream Conventions

11.1 Position

Many of the metadata elements contained in the bitstream specify a relative position or size. Position is described relative to the playback environment using a unit cube to describe the room boundaries. The origin is taken to be the front left corner of the room. Position is then described using Cartesian (x,y,z) coordinates, assigned as follows:

x: lateral, or left/right position	x=0 corresponds to left wall; x=1 corresponds to right wall.
y: longitude, or front/back position	y=0 corresponds to front wall; y=1 corresponds to back wall.
z: elevation, or up/down position	z=0 corresponds to a horizontal plane at of the height of the main screen Loudspeakers, the side and rear surround Loudspeakers; z=1 corresponds to the ceiling.

For example,

(0, 0, 0) -> front left corner, 0 elevation (left screen Loudspeaker),
(1, 0, 0) -> front right corner, 0 elevation (right screen Loudspeaker), and
(0.5, 0.5, 1) -> middle of ceiling.

Position values outside the room (i.e. values <0 or >1) are not supported.

Metadata that describes position relative to the room uses the unit axes and origin described above in this clause; the location along each axis is coded using the distance coding method described in 5.4.

11.2 Bitstream Constants

The constants listed in Table 32 are defined for bitstream version 1.

Table 32 — Bitstream Constants.

Symbol	Value
DEFAULT_OBJ_SNAP_TOL	1.0

Annex A Example Bitstreams (Informative)

A.1 Example Bitstream Organization for Two Simultaneous Beds

Figure A.1 shows an example bitstream organization for the presentation of two simultaneous Beds. Each of the Beds are intended to be mixed together and played simultaneously. Both of the *BedDefinition* elements are Children of the Parent *IAFrame* element. In the case shown in Figure A.1, the *BedDefinition* elements both have 10 audio assets, so the **MaxRendered** field of the *IAFrame* element is 20.

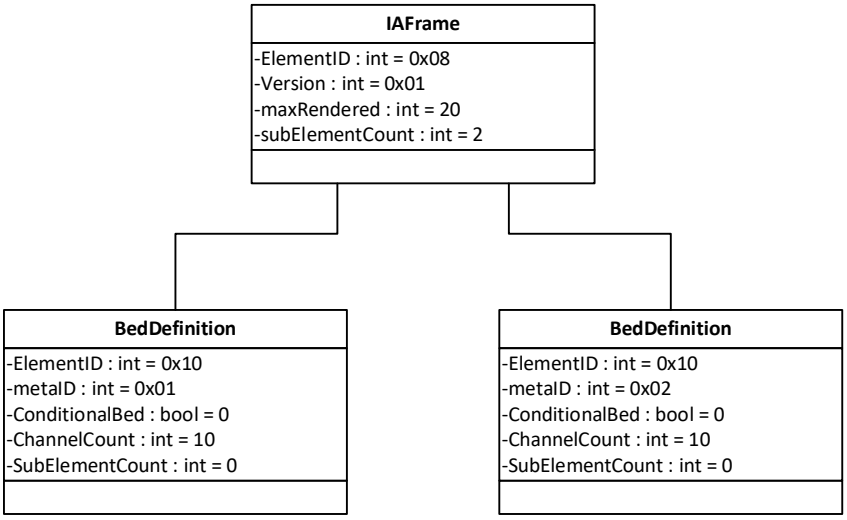

Figure A.1 — Bitstream organization for two simultaneous Beds.

A.2 Example Bitstream Organization for Conditional Beds

Figure A.2 shows an example bitstream organization for a Conditional Bed scenario. In the case shown, 6, 8, and 10 channel Beds are included in the bitstream. In this example, the Bed that will be rendered for all playback configurations is specified by the 10 channel BedDefinition element that is the Child of the IAFrame element – it has the ConditionalBed bit set to 0 – unless the Target Environment corresponds to one of the BedUseCase codes specified in the 2 Child BedDefinition elements.

The Conditional Beds for 6 and 8 channel support – they have their ConditionalBed bit set to 1 – are Children of the unconditional Bed and have their BedUseCases set to specify the replacement for 6 and 8 channel playback, respectively.

As the maximum audio assets that can be rendered at any time and under all possible configurations is 10, the MaxRendered field in the IAFrame element is set to 10.

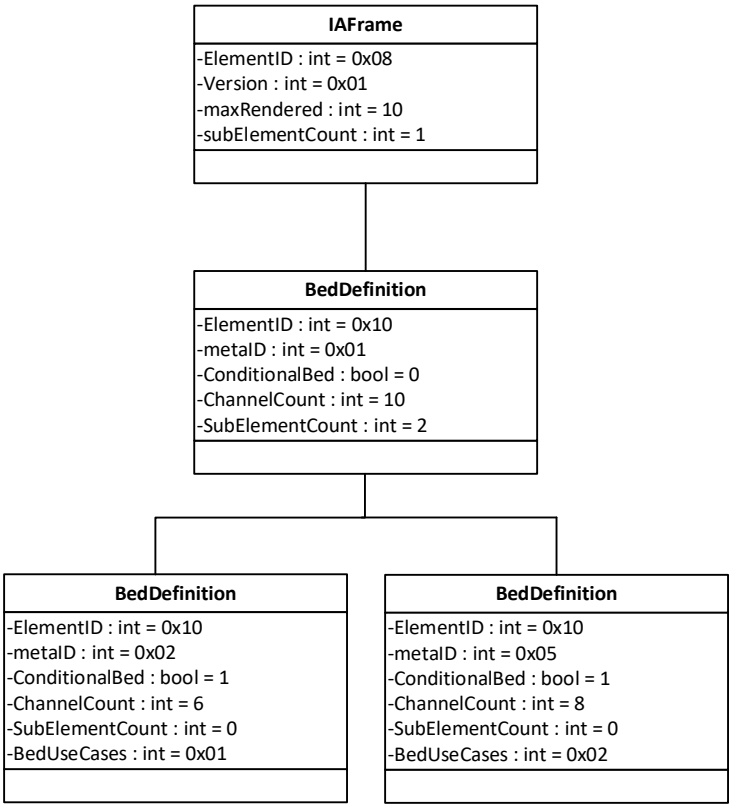

Figure A.2 — Bitstream organization for replacement Beds.

A.3 Example Bitstream Organization for Bed Remapping

Figure A.3 shows an example bitstream organization with a BedRemap element. In the case shown, a 10 channel Bed is included in the bitstream, and a BedRemap element tells how to remap this 10 channel bed to 6 channels if necessary. In this example, the Bed that will be rendered for all playback configurations is specified by the 10 channel BedDefinition element that is the Child of the IAFrame element – it has the ConditionalBed bit set to 0 – unless the Target Environment corresponds to the BedUseCase code specified in the BedRemap element.

If the Target Environment matches the BedUseCase of the BedRemap element, the BedRemap is applied to the Parent 10 channel BedDefinition yielding 6 channels.

As the maximum audio assets that can be rendered at any time and under all possible configurations is 10, the MaxRendered field in the IAFrame element is set to 10.

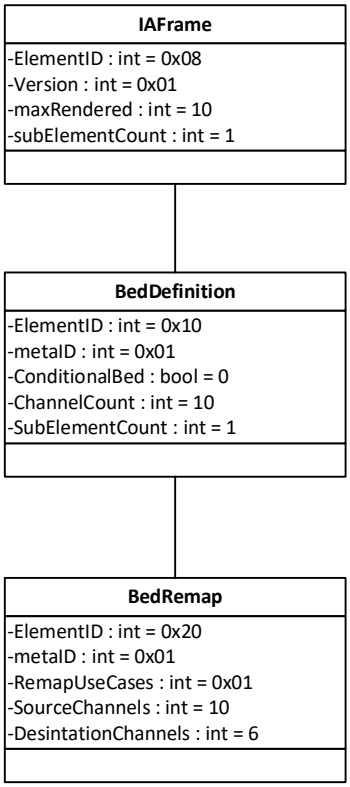

Figure A.3 — Bitstream organization for Bed remapping.

A.4 Example Bitstream Organization for Simultaneous Bed and Object Rendering

Figure A.4 shows an example bitstream organization for simultaneous Bed and object rendering. In the case shown, both the 10 channel *BedDefinition* element and the *ObjectDefinition* element are Children of the *IAFrame* element. As there will be a total of 11 audio assets rendered, the **MaxRendered** field of the *IAFrame* element is set to 11.

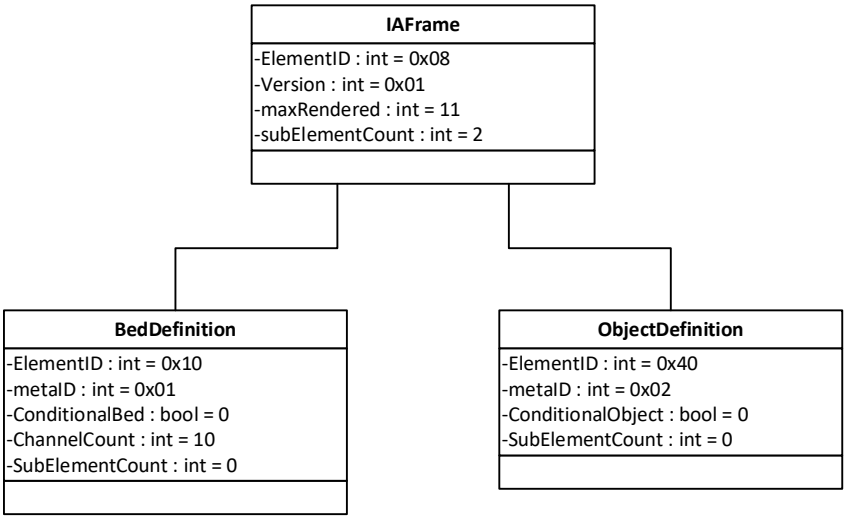

Figure A.4 — Bitstream organization for the simultaneous Bed and object rendering.

A.5 Example Bitstream Organization for Replacement Objects

Figure A.5 shows an example bitstream organization for replacements objects. In the case shown, the object that will be rendered for all playback configurations is specified by the Child **ObjectDefinition** Sub-Element of the *IAFrame* element - it has the **ConditionalObject** bit set to 0 - unless the Target Environment corresponds to the **ObjectUseCase** code specified in the second **ObjectDefinition** element that has the **ConditionalObject** bit set to 1 and the **ObjectUseCase** code set to specify 6 channel support. As there will be a total of 1 audio asset rendered, the **MaxRendered** field of the *IAFrame* element is set to 1.

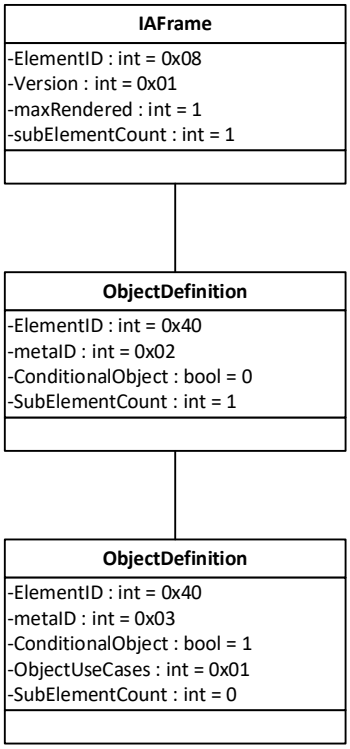

Figure A.5 — Bitstream organization for replacement objects.

Annex B Sample Rate Scalable Lossless Audio Coding (AudioDataDLC)

B.1 Sample rate scalable lossless coding, Overview

This specification supports carriage of a losslessly-coded audio format. Audio encoded in this format is carried in the bitstream using the AudioDataDLC element. Each AudioDataDLC element carries one frame of one monaural audio waveform. The AudioDataDLC element includes an AudioDataID field that allows channel or object elements to identify an AudioDataDLC element.

The frame size (and rate) is variable to support multiple video edit rates ensuring synchronization between the audio and video presentations. The audio data stored in each AudioDataDLC element is losslessly coded using a time-domain, linear prediction audio compression system specifically designed for cinema applications.

The compression system supports both 48 kHz and 96 kHz sample rates. Soundtracks recorded at 96 kHz are losslessly packaged in two frequency bands: a high quality 48 kHz sample rate base layer and an extension layer to losslessly reproduce the original 96 kHz sample rate audio. The sample rate scalable feature provides the following benefits:

- Soundtracks can be efficiently and losslessly distributed with extremely high quality: up to 96 kHz, and 24 bits per sample.
- A cinema processor with lower computational capability can receive 96 kHz audio programs and easily unpack audio streams at 48 kHz for low complexity processing. (Decoding the base layer is much lower complexity process than a high quality sample rate conversion process);
- The 48 kHz stream is generated at encode time, ensuring reliable, verifiable audio that is not dependent on the cinema processor used to playback the soundtrack;
- A single DCP can be delivered to all cinemas, including those capable of full 96 kHz playback and those capable of only 48 kHz processing.

NOTE The AudioDataDLC syntax does not support fractional frame rates.

B.2 Encoder

The encoding process shall accept PCM audio and produce an encoded bitstream that is compliant with the AudioDataDLC element and can be losslessly decoded by the decoder defined in this specification. Authoring tools capable of creating AudioDataDLC elements compliant with this specification shall be capable of this AudioDataDLC encoding process.

B.3 Encoding Process Overview (Informative)

The encoding process must produce a bitstream that can be losslessly decoded by the decoder. The encoder operates on frames of PCM data and each frame is independently decodable (that is, the encoder is stateless beyond the frame boundary).

The encoding process has 2 different modes of operation depending on the sample rate of the input PCM. The encoding process for 48 kHz sample rate PCM is shown in Figure B.1. The encoding process for 96 kHz is shown in Figure B.2. In Figure B.1, the input 48 kHz is first passed through a linear prediction stage.

NOTE For details on the predictor, see John G. Proakis and Dimitris G. Manolakis, *Digital Signal Processing*, Upper Saddle River, NJ, USA: Prentice-Hall International, 1996, pp. 511-514.

The prediction stage is forward-adaptive -- the adaptive prediction coefficients are transmitted in the bitstream to the decoder. Sending the coefficients explicitly helps minimize decoder complexity. The adaptive prediction stage is computed with fixed, finite precision math to ensure bit-exact inverse prediction in the decoder. Following the linear prediction, the residual signal is further compressed using entropy coding. When possible, the lossless coding system exploits non-uniform signal statistics of the residual signal using Rice/Golomb coding. Otherwise, the encoder transmits the residual directly as PCM. The residual signal and the prediction coefficients are then packaged into the bitstream.

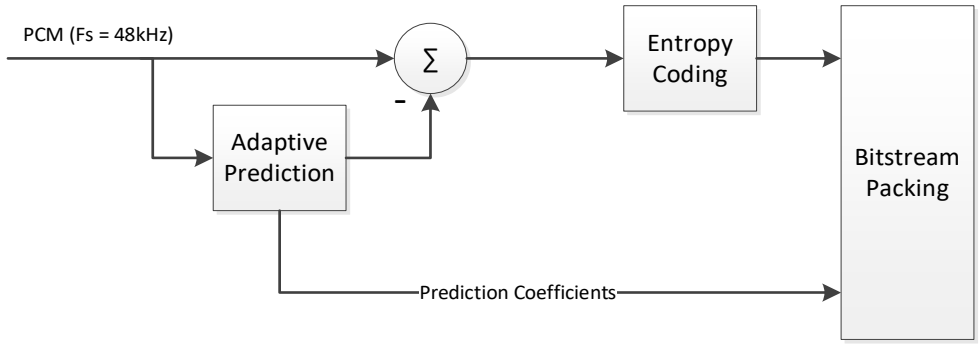

Figure B.1 — Simplified diagram of the encode process for 48 kHz PCM input.

The encoding process for 96 kHz is shown in Figure B.2. The input 96 kHz PCM is split into 2 paths; the 48 kHz base layer (which is independently decodable) and the extension layer that allows the reproduction of the full 96 kHz sample rate audio. The first process in the generation of the base layer is a low pass filter and the downsampling operation. Any low pass filter may be employed; however, the best overall system performance is provided when the filter is designed with linear phase and has sufficiently high order to get the cut off frequency close to 24 kHz while ensuring sufficient stop band rejection to prevent audible aliasing distortion.

NOTE The structure of the sample rate scalable lossless coding system ensures the decoded 96 kHz PCM output is lossless regardless of the low pass filter design and the arithmetic precision employed. In parallel to the low pass and downsampling operation, the extension is delayed to account for the delay of the low pass filter and the fixed precision interpolation process. The next steps are to upsample and interpolate the base layer and subtract it from the extension layer. The interpolation is computed with fixed, finite-precision math such that the result can be exactly reproduced in the decoder ensuring lossless output of the 96 kHz sample rate audio. Following the subtraction of the interpolated base layer from the extension layer, both layers are passed through the adaptive linear prediction and entropy coded prior to bitstream packaging as described previously in this clause for the 48 kHz PCM encoder.

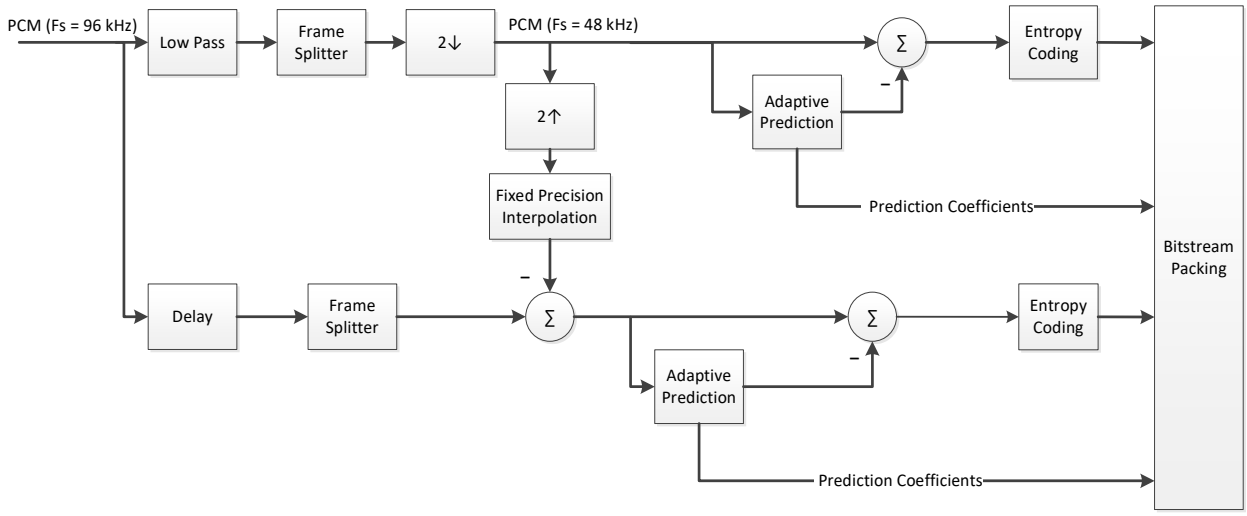

Figure B.2 — Simplified diagram of the encode process for 96 kHz PCM.

B.4 Decoder

The decoding process shall take a compliant AudioDataDLC element as input and produce PCM audio as the output.

B.5 Decoding Process Overview

Unlike the encoding process, the decoding process is normative. Similar to the encoding process, the decoding process shall have 2 distinct modes of operation: one for 48 kHz sample rate audio and one for 96 kHz sample rate audio.

The decoding process for 48 kHz sample rate audio is shown in Figure B.3. The AudioDataDLC bitstream is unpacked to obtain the entropy encoded residual signal and the adaptive prediction filter coefficients. The residual signal is then inverse entropy coded prior to the adaptive prediction process, which is the bit-exact inverse of the encoding predictive stage. The output of the predictor is the lossless 48 kHz sample rate PCM.

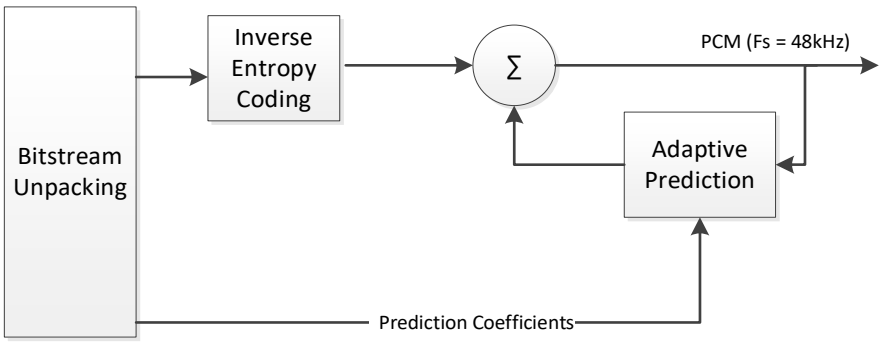

Figure B.3 — Simplified diagram of the decoding process for 48 kHz PCM.

The decoding process for 96 kHz sample rate audio is shown in Figure B.4. The AudioDataDLC bitstream is unpacked to obtain the entropy encoded residual and the prediction coefficients for both the 48 kHz sample rate base layer audio and the extension layer. Both layers have the residual signal inverse entropy coded prior to the application of the inverse prediction stages. The 48 kHz base layer is then upsampled, interpolated, and added to the extension layer to produce the 96 kHz PCM lossless output. Systems that only support a 48 kHz sample rate, or those with limited processing, may produce a 48 kHz signal without decoding the extension layer.

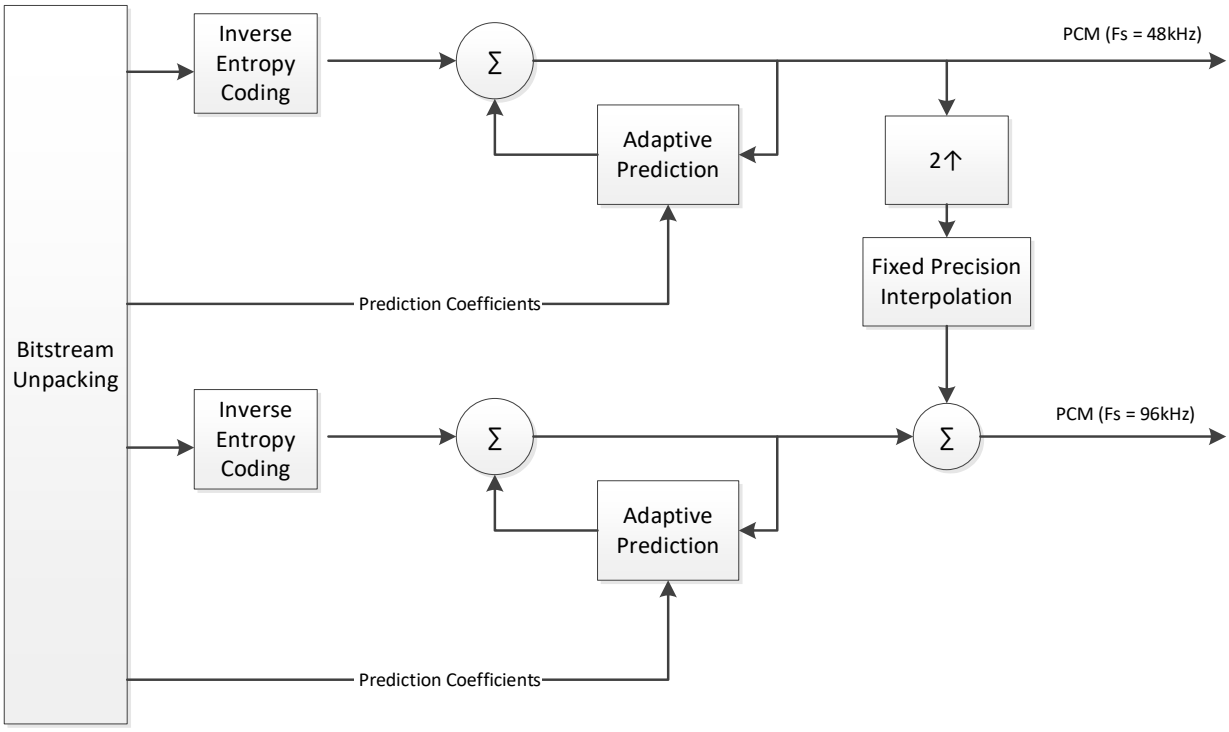

Figure B.4 — Simplified diagram of the decoding process for 96 kHz PCM.

B.6 Decoding the AudioDataDLC Audio Data

B.6 specifies the normative steps required to decode the AudioDataDLC bitstream in order to reconstruct a frame of lossless PCM data. The processing in each step is defined using short segments of pseudocode. All scalar values shall be implemented as 32-bit integers (INT32) unless otherwise specified. The variable “Accum64”, when it appears, shall be implemented as a 64-bit integer (INT64).

The normative decoding steps shall be as follows:

1. Initialize predictors and sample rate converter sample values to 0 at start of frame.
2. Unpack bitstream.
3. Convert lattice prediction coefficients to direct form filter coefficients.
4. Apply the adaptive predictor.
5. Upsample and interpolate the 48 kHz base layer audio and add it to the extension layer as necessary.
6. Shift audio back to the original bit depth.

B.7 Converting the Lattice Prediction Coefficients to Direct Form Coefficients

B.7 describes the conversion of the lattice prediction coefficients to direct form prediction coefficients for both the 48 kHz base layer and the 96 kHz extension layer if present. The output of this function is the direct form coefficients ACoeff48[ ][ ] and ACoeff96[ ][ ] (if present in stream).

NOTE KCoeff48[ ][ ] and KCoeff96[ ][ ] are read from the bitstream as 10 bit unsigned integers. In the following pseudocode the values are stored in arrays of INT32 values.

```
for(n = 0; n < NumPredRegions48; n ++){
  KCoeff48[n][0] = 1048576; /*unused state*/
  ACoeff48[n][0] = 1048576;
  ACoeffTemp[0] = 1048576; /*unused state*/

  for(j = 1; j <= Order48[n]; j ++){
    KCoeff48[n][j] -= 512;
    KCoeff48[n][j] <<= 11;
    ACoeff48[n][j] = 0;
    for(k = 1; k <= j; k ++){
      Accum64 = (INT64)KCoeff48[n][j]*(INT64)ACoeff48[n][j-k];
      ACoeffTemp[k] = ACoeff48[n][k] + (INT32)(Accum64>>20);
    }
    for(k = 1; k <= j; k ++){
      ACoeff48[n][k] = ACoeffTemp[k];
    }
  }
}

/* Convert 96 kHz Extension Layer Predictors as Necessary */
if(DLCSampleRate == 0x1){
  for(n = 0; n < NumPredRegions96; n ++){
    KCoeff96[n][0] = 1048576;
    ACoeff96[n][0] = 1048576;
    ACoeffTemp[0] = 1048576;

    for(j = 1; j <= Order96[n]; j ++){
      KCoeff96[n][j] -= 512;
      KCoeff96[n][j] <<= 11;
      ACoeff96[n][j] = 0;
      for(k = 1; k <= j; k ++){
        Accum64 =
        (INT64)KCoeff96[n][j]*(INT64)ACoeff96[n][j-k];

        ACoeffTemp[k] =
        ACoeff96[n][k] + (INT32)(Accum64>>20);
      }
      for(k = 1; k <= j; k ++){
        ACoeff96[n][k] = ACoeffTemp[k];
      }
    }
}
}
```

B.8 Applying the Predictors

B.8 describes the application of the direct form predictors to the residual data to obtain the PCM output data.

```
if (NumPredRegions48 == 0) {
  for(n = 0; n < SampleCount48; n ++) {
    PCM48[n] = Residual48[n];
  }
}
else {
  for(n = 0; n < 64; n ++) {
    Buffer[n] = 0;
  }

  Index1 = 0;
  n = 0;
  for(i = 0; i < NumPredRegions48; i ++) {
    for(j= 0; j < RegionLength48[i]; j ++) {
      for (k = 0; k < DLCSubBlockSize48; k++) {
        Index2 = Index1;
        Accum64 = 0;
        for (p = 1; p <= Order48[i]; p++) {
          Accum64 -= ((INT64)Buffer[Index2] *
              (INT64)ACoeff48[i][p]);

          Index2--;
          Index2 &= 63;
        }

        Output = (INT32)(Accum64 >> 20);
        Output = Residual48[n] + Output;
        Index1++;
        Index1 &= 63;
        Buffer[Index1] = Output;

        PCM48[n] = Output;
        n++;
      }
    }
  }
}
if(DLCSampleRate == 0x1){
  if (NumPredRegions96 == 0) {
    for(n = 0; n < SampleCount96; n ++) {
      PCM96[n] = Residual96[n];
    }
  }
  else {

    for(n = 0; n < 64; n ++) {
        Buffer[n] = 0;
    }
```

```
    Index1 = 0;
    n = 0;

    for(i = 0; i < NumPredRegions96; i ++) {
      for(j= 0; j < RegionLength96[i]; j ++) {
        for (k = 0; k < DLCSubBlockSize96; k++) {
          Index2 = Index1;
          Accum64 = 0;
          for (p = 1; p <= Order96[i]; p++) {
            Accum64 -= ((INT64)Buffer[Index2] *
               (INT64)ACoeff96[i][p]);
            Index2--;
            Index2 &= 63;
          }

          Output = (INT32)(Accum64 >> 20);
          Output = Residual96[n] + Output;
          Index1++;
          Index1 &= 63;
          Buffer[Index1] = Output;

          PCM96[n] = Output;
          n ++;
        }
      }
    }
  }
}
/*
  DLCSubBlockSize48 and DLCSubBlockSize96 can be found in Table 30. SampleCount48
                    and SampleCount96 can be found in Table 18
*/
```

B.9 Upsampling and Interpolation of the 48 kHz Base Layer

If the AudioDataDLC element contains a 96 kHz extension layer, the 48 kHz base layer needs to be upsampled and interpolated prior to the summation with the extension layer. The C-style pseudocode below defines the upsampling and interpolation operations necessary. The output of the upsampling and interpolation function is stored in PCM48to96[ ] for later use. The interp[i] coefficients are given by Table B.1. The interpolation filter is half band so even index coefficients are zero.

```
for(n = 0; n < 64; n ++){
  Buffer[n] = 0;
}
Index1 = 0;
for(n = 0, k = 0; n < SampleCount96; n += 2, k ++){
      /* Feed buffer */
      Buffer[Index1] = PCM48[k];


      Index2 = Index1;
      Index2 -= 8;
      Index2 &= 63;
      PCM48to96[n] = Buffer[Index2];
```

```
      Index2 = Index1;
      Accum64 = 0;
      for(i = 1; i < 33; i += 2){
            Accum64 += ((INT64)Buffer[Index2] *
                        (INT64)Interp[i]);
            Index2 --;
            Index2 &= 63;
      }
      PCM48to96[n + 1] = (INT32)(Accum64 >> 15);

      Index1 ++;
      Index1 &= 63;
}
/*
      SampleCount96 is found in Table 18
*/
```

Table B.1 — Coefficients for the interpolation filter (Interp).

Index	Interp[Index]	Index	Interp[Index]
0	0	17	20688
1	-138	18	0
2	0	19	-6450
3	305	20	0
4	0	21	3377
5	-618	22	0
6	0	23	-1952
7	1128	24	0
8	0	25	1128
9	-1952	26	0
10	0	27	-618
11	3377	28	0
12	0	29	305
13	-6450	30	0
14	0	31	-138
15	20688	32	0
16	32767		

B.10 Adding Base and Extension Layer and Perform Final LSB Shifting

If the DLCElement contains a 96 kHz extension layer, then the upsampled base layer (PCM48to96[]) must be added to the extension layer (PCM96[]) prior to shifting left by ShiftBits to produce 96 kHz audio output (PCM96[]). Alternatively, the 48 kHz audio for the base layer (PCM48[]) is produced by shifting left by ShiftBits.

After shifting, the bit depth of the output PCM is 32 bits. E.g., to recover a 24-bit sample, starting from the least significant bit (LSB), 8 LSBs would be discarded, and the next 24 bits would be used. To recover a 16-bit sample, starting from the LSB, 16 LSBs would be discarded, and the next 16 bits would be used.

```
if(DLCSampleRate == 0x1){
  for(n = 0; n < SampleCount96; n ++){
    /* Add 96 kHz Extension and Upsampled Base Layer */
    PCM96[n] += PCM48to96[n];
    PCM96[n] <<= ShiftBits;
  }
}
else{
  for(n = 0; n < SampleCount48; n ++){
    PCM48[n] <<= ShiftBits;
  }
}
/*
  SampleCount48 and SampleCount96 are found in Table 18
*/
```

B.11 Minimal DLC Encoder (Informative)

Since it is possible to disable adaptive prediction by setting **NumPredRegions48** and **NumPredRegions96** to zero, a very simple DLC encoder can be implemented. As shown in Figure B.1, 48 kHz audio can be encoded by entropy encoding the 48 kHz PCM. With no Adaptive Prediction, the "residual" (PCM minus Predicted) is the original PCM. The residual can be entropy encoded as a minimum width signed magnitude PCM or can be Rice/Golomb coded. In the very simple DLC encoder pseudocode below in this clause, the residual is encoded using PCM.

For 48 kHz audio, the minimum number of bits required to encode the absolute value of all samples in a sub block is determined after ShiftBits are applied. ShiftBits are set so that during decoding the final output will use 32 bits. For example, with 24-bit audio, the maximum positive sample (0 dB FS) would have a value of 0x7FFFFF which can be encoded in 23 bits. If the maximum level in a sub block were -6dB FS (0x3FFFFF), the sub block would be encoded with 22 bits of magnitude for each sample. The number of bits used to encode the magnitude of the samples in a sub block is in **BitDepth**. The magnitude of each sample is in **Residual48**. If the magnitude of the sample is non-zero, the sign of the sample is in **sign**. See 10.7.11 et seq. Note that a negative full scale (0x800000 in two's complement) magnitude would be encoded in 24 bits.

In the pseudocode below in this clause, NumDLCSubBlocks, DLCSubBlockSize48, and DLCSubBlockSize96 depend on the IAFrame Rate as defined in 10.7.9. PCM48 is the 48 kHz audio to be encoded.

When 96 kHz audio is to be encoded, it is sample rate converted to 48 kHz to be a "base layer," after being split into frames, as shown in Figure B.2. In addition, the 48 kHz base layer is upsampled to 96 kHz (PCM48to96), after ShiftBits are applied, as described in B.9 to yield an approximation of the original 96 kHz audio. The difference between the original 96 kHz audio and this approximation is saved as Residual96. As with Residual48, Residual96 is encoded in the minimum number of bits required for the block and in sign-magnitude (as opposed to two's complement) form. The decoder also generates the identical 96 kHz audio approximation through upsampling of the base layer using PCM48to96, then adds the Residual96 to yield the original 96 kHz audio.

All field names are in AudioDataDLC unless otherwise indicated. Comments clarify duplicate field names.

```
DLCSampleRate = IAFrame.SampleRate /* DLC uses same sample rate as IAFrame */
ShiftBits = 32 – IAFrame.BitDepth;
NumPredRegions48 = 0; /* No linear prediction. */
for (n = 0; n < NumDLCSubBlocks; n++) {
 CodeType = 0; /* 48k Residual CodeType is PCM, not Rice/Golomb */
 maxAbs = 0; /* Get ready to compute max abs(sample[])*/
 for (i = 0; i < DLCSubBlockSize48; i++) { /* For each sample in sub block */
   /* Convert from 32bit and transfer to Residual48. (The right shift
      operation is assumed to perform sign-extension.) */
   Residual48[(n * DLCSubBlockSize48) + i]
     = PCM48[(n * DLCSubBlockSize48) + i] >> ShiftBits;
   maxAbs = max(maxAbs, abs(Residual48[n * DLCSubBlockSize48 + i]));
 }
 /* Count how many bits we need for this residual block. */
 BitDepth = 0; /* 48k residual sub block bit depth */
 while ((1 << BitDepth) <= maxAbs){
   BitDepth++;
 }
} /* next sub block */
/* Finished computing base layer. If 96 kHz, compute extension layer. */
```

```
if (DLCSampleRate == 0x1) { /* 96 kHz. We have Residual96 */
 NumPredRegions96 = 0; /* No linear prediction. */
 for (n = 0; n < NumDLCSubBlocks; n++) {
   CodeType = 0; /* 96 kHz code type is Residual PCM, not Rice/Golomb */
   maxAbs = 0; /* Get ready to compute max abs(residual[])*/
   for (i = 0; i < DLCSubBlockSize96; i++) { /*for each sample in sub block */
     Residual96[(n * DLCSubBlockSize96) + i] =
       (PCM96[(n * DLCSubBlockSize96) + i] >> ShiftBits)-
        PCM48to96[(n * DLCSubBlockSize96) + i];
      /* Difference between actual PCM96 and upconverted 48k. */
     maxAbs = max(maxAbs, abs(Residual96[n * DLCSubBlockSize96 + i]));
   }
   /* Count how many bits we need for this residual block. */
   BitDepth = 0; /* How many bits required for abs(Residual96) */
   while ((1 << BitDepth) <= maxAbs) {
     BitDepth++;
   }
 } /* next sub block */
}  /* endif 96 kHz */
```

Annex C Constraints

C.1 General Bitstream Constraints

Bitstreams complying to this specification shall be constrained as follows:

1. MaxRendered (as defined in 10.2.5) shall be limited as follows:
    - Shall be equal to or less than 128 if SampleRate = 48 kHz
    - Shall be equal to or less than 64 if SampleRate = 96 kHz
2. The AudioDescriptionText field of the BedDefinition and ObjectDefinition elements shall not exceed 64 bytes (including the null terminator).
3. Element hierarchy shall be limited as stated below:
    a. No element which is a Child of a BedDefinition element shall have a Child BedDefinition element.
    b. No element which is a Child of an ObjectDefinition element shall have a Child ObjectDefinition element.
4. If BedRemap elements share the same Parent, the UseCases shall be constrained such that only one may be Activated.
5. If a BedDefinition element and BedRemap element share the same Parent, the UseCases shall be constrained such that both cannot be Activated at the same time.