**ISO 14496-3:2019 AMD1(X)**

ISO/IEC JTC1/SC 29/WG 6

Date: 2025-07-04

**Information technology — Coding of audio-visual objects — Part 3: Audio**

**Amendment 1: Media authenticity and immersive interchange format**

TuC on DAM stage

**Warning for WDs and CDs**

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

*A model document of an International Standard (the Model International Standard) is available at:*[*https://www.iso.org/drafting-standards.html*](https://www.iso.org/drafting-standards.html)

© ISO 20XX

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO’s member body in the country of the requester.

ISO copyright office

CP 401 • Ch. de Blandonnet 8

CH-1214 Vernier, Geneva

Phone: +41 22 749 01 11

Email: copyright@iso.org

Website: www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

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

ISO draws attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO takes no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO *[had/had not]* received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at [www.iso.org/patents](http://www.iso.org/patents). ISO shall not be held responsible for identifying any or all such patent rights.

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

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

This document was prepared by Technical Committee *[or Project Committee]* ISO/TC *[or ISO/PC]* ###, *[name of committee]*, Subcommittee SC ##, *[name of subcommittee]*.

This second/third/… edition cancels and replaces the first/second/… edition (ISO #####:####), which has been technically revised.

The main changes are as follows:

— xxx xxxxxxx xxx xxxx

A list of all parts in the ISO ##### series can be found on the ISO website.

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

Add the following at the end of Section 1.2:

IETF RFC 3986, Uniform Resource Identifier (URI): Generic Syntax

IETF RFC 9562, Universally Unique IDentifiers (UUIDs)

ISO/IEC 23091-3, Information technology — Coding-independent code-points — Part 3: Audio

Replace the last 3 rows of Table 1.1 with the following

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 46 | Audio Sync |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 47-63 | Reserved for 14496-3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 64-95 | Reserved for 23003-3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Replace the last 10 rows of Table 1.17 with the following:

|  |  |  |
| --- | --- | --- |
| 0x5E | AAC Profile | L9 |
| 0x5F | ALS Simple Profile | L5 |
| 0x60 | ALS Simple Profile | L6 |
| 0x61 | ALS Simple Profile | L7 |
| 0x62 - 0x6F | reserved for ISO/IEC 14496-3 | - |
| 0x70 - 0x7F | reserved for ISO/IEC 23003-3 | - |
| 0x80 - 0xFD | user private | - |
| 0xFE | no audio profile specified | - |
| 0xFF | no audio capability required | - |
| NOTE Usage of the value 0xFE indicates that the content described by this InitialObjectDescriptor does not comply to any audio profile specified in ISO/IEC 14496-3. Usage of the value 0xFF indicates that none of the audio profile capabilities are required for this content. | | |

After “case 46” in Table 1.19 add the following:

Table 1.19 – Syntax of AudioSpecificConfig()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| AudioSpecificConfig () |  |  |
| { |  |  |
| … |  |  |
| case 46: |  |  |
| AudioSyncFeatureSpecificConfig(): |  |  |
| break; |  |  |
| case 47: |  |  |
| iif\_specific\_config(): |  |  |
| break; |  |  |
| default: |  |  |
| /\* reserved \*/ |  |  |
| } |  |  |
| … |  |  |
| a In the Baseline USAC profile defined in ISO/IEC 23003-3, the backwards compatible signaling of SBR, PS, MPS, or SAOC at the end of the AudioSpecificConfig() (i.e., using the extensionIdentifier bitstream element) is not permitted. | | |

After 1.6.2.1.22, add the following:

1.6.2.1.23 **iif\_specific\_config**

Defined in subpart 4.

Add the following at the end of Table 1.21:

|  |  |  |  |
| --- | --- | --- | --- |
| 47 | AAC IIF | ISO/IEC 14496‑3 Subpart 4 | see subclause 1.6.2.2.2.6 |

After 1.6.2.2.2.5, add the following:

**1.6.2.2.2.6 AAC IIF**

One top level payload (iif\_access\_unit()) is mapped into one access unit. A sequence of access units forms one elementary stream.

In 4.4.2.7, replace Table 4.59 with the following:

Table 4.59 – Syntax of extension\_payload()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| extension\_payload(cnt) |  |  |
| { |  |  |
| **extension\_type**; | **4** | **uimsbf** |
| align = 4; |  |  |
| switch( extension\_type ) { |  |  |
| case EXT\_DYNAMIC\_RANGE: |  |  |
| return dynamic\_range\_info(); |  |  |
| case EXT\_UNI\_DRC: |  |  |
| return uniDrc(); |  |  |
| case EXT\_AUTH\_DATA: |  |  |
| **fill\_nibble**; /\* shall be ‘0000’ \*/ | **4** | **uimsbf** |
| return auth\_extension(); |  |  |
| case EXT\_ADD\_TYPES: |  |  |
| **extension\_type\_add**; | **4** | **uimsbf** |
| extension\_type = 0xF + extension\_type\_add; |  |  |
| switch( extension\_type ) { |  |  |
| default: |  |  |
| return 1; |  |  |
| } |  |  |
| case EXT\_SAC\_DATA: |  |  |
| return sac\_extension\_data(cnt); |  |  |
| case EXT\_SAOC\_DATA: |  |  |
| return saoc\_extension\_data(cnt); |  |  |
| case EXT\_LDSAC\_DATA: |  |  |
| return ldsac\_extension\_data(cnt); |  |  |
| case EXT\_SBR\_DATA: |  |  |
| return sbr\_extension\_data(id\_aac, 0); |  | Note 1 |
| case EXT\_SBR\_DATA\_CRC: |  |  |
| return sbr\_extension\_data(id\_aac, 1); |  | Note 1 |
| case EXT\_SAOC\_DE\_DATA: |  |  |
| return saoc\_de\_extension\_data(cnt); |  |  |
| case EXT\_DATA\_LENGTH: |  |  |
| hlp = 1; |  |  |
| **len**; | **4** | **uimsbf** |
| if (len==15) { |  |  |
| len += **add\_len;** | **8** | **uimsbf** |
| hlp += 1; |  |  |
| If (add\_len==255) { |  |  |
| len += **add\_add\_len**; | **16** | **uimsbf** |
| hlp += 2; |  |  |
| }  } |  |  |
| return hlp+extension\_payload(len); |  | **Note 2** |
| case EXT\_FILL\_DATA: |  |  |
| **fill\_nibble**; /\* shall be ‘0000’ \*/ | **4** | **uimsbf** |
| for (i=0; i<cnt-1; i++) { |  |  |
| **fill\_byte[i]**; /\* shall be ‘10100101’ \*/ | **8** | **uimsbf** |
| } |  |  |
| return cnt; |  |  |
| case EXT\_DATA\_ELEMENT: |  |  |
| **data\_element\_version**; | **4** | **uimsbf** |
| switch( data\_element\_version ) { |  |  |
| case ANC\_DATA: |  |  |
| loopCounter = 0; |  |  |
| dataElementLength = 0; |  |  |
| do { |  |  |
| **dataElementLengthPart**; | **8** | **uimsbf** |
| dataElementLength += dataElementLengthPart; |  |  |
| loopCounter++; |  |  |
| } while (dataElementLengthPart == 255); |  |  |
| for (i=0; i<dataElementLength; i++) { |  |  |
| **data\_element\_byte[i]**; | **8** | **uimsbf** |
| } |  |  |
| return (dataElementLength+loopCounter+1); |  |  |
| default: |  |  |
| align = 0; |  |  |
| } |  |  |
| case EXT\_FIL: |  |  |
| default: |  |  |
| for (i=0; i<8\*(cnt-1)+align; i++) { |  |  |
| **other\_bits[i]**; | **1** | **uimsbf** |
| } |  |  |
| return cnt; |  |  |
| } |  |  |
| } |  |  |
| Note 1: id\_aac is the id\_syn\_ele of the corresponding AAC element (ID\_SCE or ID\_CPE) or ID\_SCE in case of CCE.  Note 2: The extension\_payload() included here shall not have extension\_type == EXT\_DATA\_LENGTH. | | |

Add the following after 4.4.2.9:

* + - 1. **Payloads for the audio object type AAC IIF**

**Table 4.87 – Syntax of iif\_access\_unit()**

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| iif\_acess\_unit() |  |  |
| { |  |  |
| while (bits\_to\_decode() > 23) { |  |  |
| iif\_block(); |  |  |
| } |  |  |
| } |  |  |

**Table 4.88 – Syntax of iif\_block()**

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| iif\_block() |  |  |
| { |  |  |
| iif\_block\_header(); |  |  |
| iif\_block\_payload(); |  |  |
| iif\_block\_footer(); |  |  |
| } |  |  |

**Table 4.89 – Syntax of iif\_block\_header()**

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| iif\_block\_header() |  |  |
| { |  |  |
| **frame\_counter;** | **4** | **uimsbf** |
| **block\_id;** | **5** | **uimsbf** |
| **block\_priority;** | **2** | **uimsbf** |
| **block\_size;** | **11** | **uimsbf** |
| **b\_block\_protected;** | **1** | **uimsbf** |
| **reserved;** | **1** | **uimsbf** |
| block\_bytes\_left = block\_size; |  |  |
| if (block\_size == 0x7ff) { |  |  |
| ext\_block\_size = 0; |  |  |
| ext\_blk\_size\_loop\_cnt = 0; |  |  |
| b\_more\_bits = 1; |  |  |
| while (0 != b\_more\_bits) { |  |  |
| if (ext\_blk\_size\_loop\_cnt < 2) { |  |  |
| **block\_size\_extension**; | **7** | **uimbsf** |
| **b\_more\_bits**; | **1** | **uimsbf** |
| ext\_block\_size <<= 7; |  |  |
| } else { |  |  |
| **block\_size\_extension**; | **8** | **uimsbf** |
| b\_more\_bits = 0; |  |  |
| ext\_block\_size <<= 8; |  |  |
| } |  |  |
| ext\_block\_size += block\_size\_extension; |  |  |
| ext\_blk\_size\_loop\_cnt += 1; |  |  |
| } |  |  |
| block\_size += ext\_block\_size + ext\_blk\_size\_loop\_cnt; |  |  |
| block\_bytes\_left = block\_size – ext\_blk\_size\_loop\_cnt; |  |  |
| } |  |  |
| if (b\_block\_protected) { |  |  |
| block\_bytes\_left = block\_bytes\_left - 2; |  |  |
| } |  |  |
| } |  |  |

**Table 4.90 – Syntax of iif\_block\_payload()**

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| iif\_block\_payload() |  |  |
| { |  |  |
| samplingFrequency = block\_sampling\_rate[block\_id]; |  |  |
|  |  |  |
| /\* Config Update \*/ |  |  |
| **b\_config\_update\_inband;** | **1** | **uimsbf** |
| if (b\_config\_update\_inband) { |  |  |
| iif\_config\_update(0); |  |  |
| } |  |  |
| else { |  |  |
| **dynamic\_config\_upd\_idx;** | **3** | **uimsbf** |
| } |  |  |
|  |  |  |
| iif\_metadata\_payload(); |  |  |
|  |  |  |
| /\* Audio Data \*/ |  |  |
| if (audio\_signals\_in\_source\_block[block\_id] > 0) { |  |  |
| iif\_audio\_payload(); |  |  |
| } |  |  |
|  |  |  |
| byte\_alignment(); |  |  |
| } |  |  |

**Table 4.91 – Syntax of iif\_audio\_payload()**

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| iif\_audio\_payload() |  |  |
| { |  |  |
| b\_more\_elements = 1; |  |  |
| while (b\_more\_elements) |  |  |
| { |  |  |
| **b\_more\_elements;** | **1** | **uimsbf** |
| **b\_element\_is\_cpe;** | **1** | **uimsbf** |
| if (b\_element\_is\_cpe == 0) |  |  |
| { |  |  |
| single\_channel\_element(); |  |  |
| } |  |  |
| else |  |  |
| { |  |  |
| channel\_pair\_element(); |  |  |
| } |  |  |
| } |  |  |
| } |  |  |

**Table 4.90 – Syntax of iif\_metadata\_payload()**

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| iif\_metadata\_payload() |  |  |
| { |  |  |
| num\_metadata\_sections = 0; |  |  |
| metadata\_section\_type[num\_metadata\_sections] = -1; |  |  |
|  |  |  |
| if (use\_explicit\_metadata\_signaling[block\_id]) { |  |  |
| b\_more\_md\_sections = 1; |  |  |
| while (b\_more\_md\_sections) { |  |  |
| **md\_type\_idx;** | **4** | **uimsbf** |
| metadata\_section\_type[num\_metadata\_sections] =  inband\_metadata\_type[block\_id][md\_type\_idx]; |  |  |
| num\_metadata\_sections += 1; |  |  |
| metadata\_section\_type[num\_metadata\_sections] = -1; |  |  |
| **b\_more\_md\_sections;** | **1** | **uimsbf** |
| } |  |  |
| } |  |  |
| else { |  |  |
| for (m=0; m<num\_metadata\_types[block\_id]; m++) { |  |  |
| **b\_metadata\_present;** | **1** | **uimsbf** |
| if (b\_metadata\_present) { |  |  |
| metadata\_section\_type[num\_metadata\_sections] =  inband\_metadata\_type[block\_id][m]; |  |  |
| num\_metadata\_sections += 1; |  |  |
| metadata\_section\_type[num\_metadata\_sections] = -1; |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
|  |  |  |
| if (num\_metadata\_sections > 0) { |  |  |
| num\_metadata\_sections = num\_metadata\_sections-1; |  |  |
| for (s=0; s<num\_metadata\_sections; s++) { |  |  |
| n\_metadata\_bits = md\_size(); |  |  |
| inband\_metadata(metadata\_section\_type[s], n\_metadata\_bits); |  |  |
| } |  |  |
|  |  |  |
| if (audio\_signals\_in\_source\_block[block\_id] > 0) { |  |  |
| n\_metadata\_bits = md\_size(); |  |  |
| } |  |  |
| else { |  |  |
| n\_metadata\_bits = 8\*block\_bytes\_left; |  |  |
| } |  |  |
| inband\_metadata  (metadata\_section\_type[num\_metadata\_sections],  n\_metadata\_bits); |  |  |
| } |  |  |
| } |  |  |

**Table 4.91 – Syntax of md\_size()**

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| md\_size() |  |  |
| { |  |  |
| **md\_size\_in\_bits;** | **10** | **uimsbf** |
| if (md\_size\_in\_bits == 0) { |  |  |
| **ext\_md\_size\_in\_bits;** | **18** | **uimsbf** |
| md\_size\_in\_bits = (1<<10) + ext\_md\_size\_in\_bits; |  |  |
| } |  |  |
| return md\_size\_in\_bits; |  |  |
| } |  |  |

**Table 4.91 – Syntax of inband\_metadata()**

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| inband\_metadata(\_md\_type, \_nbits) |  |  |
| { |  |  |
| switch (\_md\_type) { |  |  |
| case INBAND\_METADATA\_TYPE\_AUTH:  auth\_extension(); |  |  |
| break; |  |  |
| default: |  |  |
| **unknown\_metadata;** | **\_nbits** | **uimsbf** |
| break; |  |  |
| } |  |  |
|  |  |  |
| } |  |  |

**Table 4.91 – Syntax of iif\_config\_update()**

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| iif\_config\_update(\_cfg) |  |  |
| { |  |  |
| if (audio\_presentation\_type ==  AUDIO\_PRESENTATION\_TYPE\_CHANNELS) { |  |  |
| **active\_channel\_config\_idx[\_cfg];** | **3** | **uimsbf** |
| active\_channel\_mode[\_cfg] =  channel\_config\_set\_channel\_mode[active\_channel\_config\_idx[\_cfg]]; |  |  |
| active\_channel\_mask[\_cfg] =  channel\_config\_set\_channel\_mask[active\_channel\_config\_idx[\_cfg]]; |  |  |
|  |  |  |
| **b\_active\_channel\_config\_change[\_cfg];** | **1** | **uimsbf** |
| if (b\_active\_channel\_config\_change[\_cfg]) { |  |  |
| **next\_channel\_config\_sample\_offset\_minus1;** | **10** | **uimsbf** |
| next\_channel\_config\_sample\_offset[\_cfg] =  next\_channel\_config\_sample\_offset\_minus1 + 1; |  |  |
| **next\_channel\_config\_idx[\_cfg];** | **3** | **uimsbf** |
| next\_channel\_mode[\_cfg] =  channel\_config\_set\_channel\_mode  [next\_channel\_config\_idx[\_cfg]]; |  |  |
| next\_channel\_mask[\_cfg] =  channel\_config\_set\_channel\_mask  [next\_channel\_config\_idx[\_cfg]]; |  |  |
| } |  |  |
| } |  |  |
|  |  |  |
| if (num\_target\_devices > 0) { |  |  |
| /\* init with defaults \*/ |  |  |
| for (d=0; d<num\_target\_devices; d++) { |  |  |
| for (s=0; s<num\_signals\_for\_device\_routing; s++) { |  |  |
| audio\_signal\_role[\_cfg][d][s] = default\_audio\_signal\_role[d][s]; |  |  |
| audio\_signal\_gain[\_cfg][d][s] = default\_audio\_signal\_gain[d][s]; |  |  |
| audio\_signal\_delay[\_cfg][d][s] =  default\_audio\_signal\_delay[d][s]; |  |  |
| } |  |  |
| } |  |  |
| /\* override \*/ |  |  |
| **b\_override\_defaults;** | **1** | **uimsbf** |
| if (b\_override\_defaults == 1) { |  |  |
| for (d=0; d<num\_target\_devices; d++) { |  |  |
| if (device\_address\_mask[block\_id]&(1<<d)) { |  |  |
| /\* only if parent block is addressed to this device! \*/ |  |  |
| **b\_override\_default\_role\_for\_device;** | **1** | **uimsbf** |
| if (b\_override\_default\_role\_for\_device) { |  |  |
| for (s=0; s<num\_signals\_for\_device\_routing; s++) { |  |  |
| **b\_signal\_connected;** | **1** | **uimsbf** |
| if (b\_signal\_connected == 1) { |  |  |
| **audio\_signal\_role[\_cfg][d][s];** | **3** | **uimsbf** |
| } |  |  |
| else { |  |  |
| audio\_signal\_role[\_cfg][d][s] = -1; |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| **b\_override\_default\_params\_for\_device;** | **1** | **uimsbf** |
| if (b\_override\_default\_params\_for\_device) { |  |  |
| for (s=0; s<num\_signals\_for\_device\_routing; s++) { |  |  |
| if (audio\_signal\_role[\_cfg][d][s] >= 0){ |  |  |
| **b\_override\_default\_gain;** | **1** | **uimsbf** |
| if (b\_override\_default\_gain) { |  |  |
| **gain\_code;** | **5** | **uimsbf** |
| **b\_override\_default\_delay;** | **1** | **uimsbf** |
| if (b\_override\_default\_delay) { |  |  |
| **b\_relative\_delay;** | **1** | **uimsbf** |
| if (b\_relative\_delay == 1) { |  |  |
| **rel\_delay\_code;** | **7** | **uimsbf** |
| } |  |  |
| else { |  |  |
| **delay\_code;** | **12** | **uimsbf** |
| audio\_signal\_delay[\_cfg][d][s] =  4\*delay\_code; |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| } |  |  |

**Table 4.91 – Syntax of iif\_block\_footer()**

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| iif\_block\_footer() |  |  |
| { |  |  |
| **block\_fill;** | **block\_bytes\_left \* 8** |  |
| if (b\_block\_protected) { |  |  |
| **block\_crc;** | **16** | **uimsbf** |
| } |  |  |
| } |  |  |

At the end of 4.5.2.9.1, add:

**extension\_type\_add** Four bit field indicating the value to be added to extension\_type to identify the type of fill element content according to Table 4.127.

Replace Table 4.127 with the following:

Table 4.127 – Values of the extension\_type field

|  |  |  |
| --- | --- | --- |
| Symbol | Value of extension\_type | Purpose |
| EXT\_FILL | ‘0000’ | bitstream payload filler |
| EXT\_FILL\_DATA | ‘0001’ | bitstream payload data as filler |
| EXT\_DATA\_ELEMENT | ’0010‘ | data element |
| EXT\_DATA\_LENGTH | ‘0011’ | container with explicit length for extension\_payload() |
| EXT\_UNI\_DRC | ’0100‘ | Unified dynamic range control |
| EXT\_AUTH\_DATA | ’0101‘ | Authentication Data |
| - | ’0110‘ | reserved |
| - | ’0111‘ | reserved |
| EXT\_ADD\_TYPES | ‘1000’ | Allows signalling of additional extension\_type values |
| EXT\_LDSAC\_DATA | ‘1001’ | LD MPEG Surround |
| EXT\_SAOC\_DATA | ‘1010’ | SAOC |
| EXT\_DYNAMIC\_RANGE | ‘1011’ | dynamic range control |
| EXT\_SAC\_DATA | ‘1100’ | MPEG Surround |
| EXT\_SBR\_DATA | ‘1101’ | SBR enhancement |
| EXT\_SBR\_DATA\_CRC | ‘1110’ | SBR enhancement with CRC |
| EXT\_SAOC\_DE\_DATA | ‘1111’ | SAOC-DE |
| Note: Extension payloads of the type EXT\_FILL or EXT\_FILL\_DATA have to be added to the bitstream payload if the total bits for all audio data together with all additional data are lower than the minimum allowed number of bits in this frame necessary to reach the target bitrate. Those extension payloads are avoided under normal conditions and free bits are used to fill up the bit reservoir. Those extension payloads are written only if the bit reservoir is full.  ‘reserved’ values can be used for a further extension of the syntax in a compatible way. | | |

After 4.5.2.16, add:

**4.5.2.17 Media Authentication**

MPEG-4 audio supports media authentication. The corresponding data is carried in an extension payload with the type EXT\_AUTH\_DATA. The syntax element auth\_extension() shall be used to embed media authentication information, as defined in Table 4.XXX.

**4.5.2.17.1 Media Authentication Syntax**

**Table 4.XXX — Syntax of auth\_extension**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| auth\_extension() |  |  |
| { |  |  |
| **numAuthExtMinus1** = escapedValue(4,4,4); | **4, 8, 12** | **uimsbf** |
| for (i=0; i <= numAuthExtMinus1; i++) { |  |  |
| **authSequence** | **1** | **uimsbf** |
| **authExtType** | **3** | **uimsbf** |
| switch (authExtType) { |  |  |
| case AUTH\_START: |  |  |
| authExtConfigAAC(); |  |  |
| break; |  |  |
| case AUTH\_SIG: |  |  |
| authExtSig(); |  |  |
| break; |  |  |
| case AUTH\_UUID: |  |  |
| authUUID(); |  |  |
| break; |  |  |
| case AUTH\_TIME: |  |  |
| authTimestamp(); |  |  |
| break; |  |  |
| default: |  |  |
| break; |  |  |
| } |  |  |
| } |  |  |
| } |  |  |

**Table 4.XXY — Syntax of authExtConfigAAC**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| authExtConfigAAC() |  |  |
| { |  |  |
| **authID** | **8** | **uimsbf** |
| **authHashType** = escapedValue(4,8,8); | **4, 12, 20** | **uimsbf** |
| **authKeyID** = escapedValue(3,8,8); | **3, 11, 19** | **uimsbf** |
| **authAddExtTypes** | **1** | **uimsbf** |
| **authProvID** = escapedValue(8,8,16); | **8, 16, 32** | **uimsbf** |
| if (authProvID == 0x00) { |  |  |
| **authSourceURILengthMinus1** | **8** | **uimsbf** |
| for (i=0; i<= authProvIDLengthMinus1; i++) { |  |  |
| **authSourceURI[i]** | **8** | **bslbf** |
| } |  |  |
| } |  |  |
| if (authAddExtTypes){ |  |  |
| **authAddExtInclusion** | **1** | **uimsbf** |
| **authAddExtTypeListLengthMinus1** | **3** | **uimsbf** |
| for (i=0; i <= authExtTypeListLengthMinus1; i++) { |  |  |
| **authAddExtType[i]** | **4** | **uimsbf** |
| } |  |  |
| } |  |  |
| } |  |  |

**Table XXY — Syntax of authExtSig**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| authExtSig() |  |  |
| { |  |  |
| **authID** | **8** | **uimsbf** |
| **authPartialSig** | **1** | **uimsbf** |
| if (authPartialSig) { |  |  |
| **sigSegmentStart** | **1** | **uimsbf** |
| **sigSegmentStop** | **1** | **uimsbf** |
| **sigSegmentLengthMinus1** | **5** | **uimsbf** |
| **sigPartial** | **(sigSegmentLengthMinus1+1)\*8** | **uimsbf** |
| } else { |  |  |
| **sigLengthMinus1** | **7** | **uimsbf** |
| **sigComplete** | **(sigLengthMinus1+1)\*8** | **uimsbf** |
| } |  |  |
| } |  |  |

**Table 3 — Syntax of authUUID**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| authUUID() |  |  |
| { |  |  |
| **uuidSegmentStart** | **1** | **uimsbf** |
| **uuidSegmentStop** | **1** | **uimsbf** |
| **uuidSegmentLengthMinus1** | **4** | **uimsbf** |
| **uuid** | **(uuidSegmentLengthMinus1+1)\*8** | **uimsbf** |
| } |  |  |

**Table 4 — Syntax of authTimestamp**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| authTimestamp() |  |  |
| { |  |  |
| **authID** | **8** | **uimsbf** |
| **authTimeType** | **7** | **uimsbf** |
| **authTimeOffsetType** | **1** | **uimsbf** |
| switch (authTimeType ) { |  |  |
| case authTimeLong: |  |  |
| **authTime** = escapedValue(28,32,0); | **28, 60** | **uimsbf** |
| **authTimeOffset** | **12** | **uimsbf** |
| break; |  |  |
| case authTimeShort: |  |  |
| **authTimeS** = escapedValue(4,8,8); | **4, 12, 20** | **uimsbf** |
| **authTimeOffsetS** = escapedValue(4,8,8); | **4, 12, 20** | **uimsbf** |
| break; |  |  |
| case authTimeTAI: { /\* acc. ISO/IEC 23001-17 \*/ |  |  |
| **TAI\_timestamp;** | **64** | **uimsbf** |
| **status\_bits;** | **8** | **uimsbf** |
| break; |  |  |
| default: |  |  |
| break; |  |  |
| } |  |  |
| } |  |  |

**4.5.2.17.2 Media Authentication Semantics**

|  |  |
| --- | --- |
| **numAuthExtMinus1** | Plus 1 indicates the number of signalled authExtType elements. |
| **authSequence** | Indicates the authentication sequence to which the related authentication information belongs to. |
| **authExtType** | Indicates the type of authentication information signalled. |
| |  |  | | --- | --- | | **authExtType** | **Value** | | AUTH\_START | 0 | | AUTH\_SIG | 1 | | AUTH\_UUID | 2 | | AUTH\_TIME | 3 | | Reserved for ISO use | 4-7 | | |
|  | |
| **authID** | Shall be used to identify the combination of authHashType, authProvID and authKeyID to which the related authentication information belongs to. This may be used to enable authentication of one authentication sequence with different authentication configurations. |
| **authHashType** | Indicates the hashing algorithm used. |
| |  |  | | --- | --- | | **authHashType Value** | **Hashing algorithm** | | 0 | SHA-1 | | 1 | SHA-224 | | 2 | SHA-256 | | 3 | SHA-384 | | 4 | SHA-512 | | All other values | /\* reserved for ISO use \*/ | | |
|  | |
| **escapedValue()** | See ISO/IEC 23003-3. |
| **authKeyID** | Identifies the authentication key used to calculate the value of the signature in authExtSig(). The values of authKeyID are dependent on the authentication provider and are not defined in the present document. This value shall be set to 0 in case there is no key needed for the underlying hashing function. |
| **authAddExtTypes** | Indicates if authentication information for additional extension types is signalled. |
| **authProvID** | Identifies the provider of the authentication system. In case authProvID equals to 1, there is no provider. This mode can be used to create a message digest only by using the method identified by authHashType. |
| |  |  | | --- | --- | | authProvID Value | Authentication Provider URI | | 0 | See authSourceURI. | | 1 | Message Digest only | | 2-… | /\* Registration Authority \*/ | | |
|  | |
| **authSourceURILengthMinus1** | Plus 1 indicates the length of the authSourceURI-field in bytes. |
| **authSourceURI** | Contains a URI with syntax and semantics as defined in as defined in IETF RFC 3986. |
| **authAddExtInclusion** | If set to 1, all extension types signalled in authAddExtType shall be included into the calculation of the authentication information, according to 4.5.2.17.3. If set to 0, all extension types signalled in authAddExtType shall be excluded from the calculation of the authentication information, according to 4.5.2.17.3. |
| **authAddExtTypeListLengthMinus1** | Plus 1 indicates the length of the list of authAddExtType. | |
| **authAddExtType** | Indicates the extension type to be included or excluded for the calculation of the authentication information. |
| **authPartialSig** | If set to ‘1’, indicates that the signature is transmitted partially. |
| **sigSegmentStart** | Indicates that the following sigPartial is the first segment of a signature. |
| **sigSegmentStop** | Indicates that the following sigPartial is the last segment of a signature.  NOTE: If both sigSegmentStartand and sigSegmentStop are equal to ‘1’, sigPartial contains a signature which is complete, but shorter than a full signature resulting from the related hashing algorithm. |
| **sigSegmentLengthMinus1** | Plus 1 indicates the length of the sigPartial field. |
| **sigPartial** | This field carries a segment of the signature resulting from the used verification mechanism.  Note: Verification may happen comparing only a subset of the bits resulting from the hashing algorithm. |
| **sigLengthMinus1** | Plus 1 indicates the length of the sigComplete field in bytes. |
| **sigComplete** | This field carries the signature resulting from the used verification mechanism.  Note: Verification may happen comparing only a subset of the bits resulting from the hashing algorithm. |
| **uuidSegmentStart** | If set to ‘1’, indicates if the bytes in the uuid field are the first bytes of a UUID-segment. |
| **uuidSegmentStop** | If set to ‘1’, indicates if the bytes in the uuid field are the last bytes of a UUID-segment.  Note: If both uuidSegmentStart and uuidSegmentStop are set to ‘1’ the uuid field contains the full UUID. |
| **uuidSegmentLengthMinus1** | Plus 1 indicates the length of the uuid-field in bytes. |
| **uuid** | This field contains the UUID of the related (sub-)stream. It may be used to map the audio stream to other media types, such as a related video stream. |
| **authTimeType** | Indicates the type of the time-signalling. |
| |  |  | | --- | --- | | **authTimeType Value** | **Timing Scheme** | | 0 | authTimeLong | | 1 | authTimeShort | | 2 | authTimeTAI | | 3-127 | /\* reserved \*/ | | |
|  | |
| **authTimeOffsetType** | Indicates the unit of *authTimeOffset* value. This shall be set to ‘0’ if the unit is milliseconds, and shall be set to ‘1’ for using the sampling rate configured for the underlying signal type as base time. |
| **authTime** | Indicates the base time. This is counted in seconds and the count starts on January 1st, 2025 at 00.00.01 UTC. |
| **authTimeOffset** | Indicates the time offset compared to the value indicated in authTime. The time shall be set in a way that authTime + authTimeOffset indicates the time when the first sample of following related access unit has been recorded. |
| **authTimeS** | Indicates the base time elapsed in seconds since the last time update of type ‘authTimeLong’. |
| **authTimeOffsetS** | Indicates the time offset compared to the value indicated in authTimeS. The time shall be set in a way that authTime + authTimeOffset + authTimeS + authTimeOffsetS indicates the time when the first sample of following related access unit has been recorded. |
| **TAI\_timestamp** | Indicates the TAI\_timestamp according to ISO/IEC 23001-17. |
| **status\_bits** | Indicates the bits synchronization\_state, timestamp\_generation\_failure, timestamp\_is\_modified and reserved according to ISO/IEC 23001-17. |

**4.5.2.17.3 Media Authentication Interface**

For applications which require media authentication, related data for verification of the authenticity of a bitstream and mapping to other media types shall be provided to the system by using the syntax element mpeg4audio\_GetAuthData().

**4.5.2.17.3.1 Syntax**

**Table 5 — Syntax of mpeg4audio\_GetAuthData**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| mpeg4audio\_GetAuthData() |  |  |
| { |  |  |
| authExtConfigAAC() |  |  |
| authExtSig() |  |  |
| authUUID() |  |  |
| authTimestamp() |  |  |
| **gad\_bytesLengthMinus1** | **64** | **uimsbf** |
| **gad\_bytes** | **(gad\_bytesLengthMinus1+1)\*8** | **uimsbf** |
| } |  |  |

**4.5.2.17.3.2 Semantics**

|  |  |  |
| --- | --- | --- |
| **gad\_bytesLengthMinus1** | Plus 1 indicates the length of the gad\_bytes field. | |
| **gad\_bytes** | Includes all bytes relevant for the creation of the authentication information according to the configuration described in authExtConfigAAC(). |

**4.5.2.17.3.3 Processing**

The decoder shall extract all data related to media authentication for every authentication sequence as indicated by authID and authSequence, and populate the respective fields in mpeg4audio\_GetAuthData().

authExtConfigAAC(), authExtSig(), authUUID() and authTimestamp() for the related authentication sequence shall be copied from the respective bitstream elements. The gad\_bytes field shall be populated by concatenating all bytes of all access units starting from the access unit including the related authExtConfigAAC() syntax element up until and including the access unit with the related authExtSig() syntax element where either sigSegmentStop==1 or sigComplete==1.

The following extension types indicated by extension\_type in Table 4.127 shall be excluded by default:

* EXT\_FILL
* EXT\_FILL\_DATA
* EXT\_DATA\_ELEMENT
* EXT\_DATA\_LENGTH

All other extension types shall be included, unless they signalled to be excluded via authAddExtType and authAddExtInclusion equal to ‘0’. In the case an extension type is excluded, the respective bits shall be replaced with ‘0’s before population in the interface. The extension types excluded by default may be signalled to be included using authAddExtInclusion equal to ‘1’ and setting the respective authAddExtType value.

Note: An access unit including an authExtConfigAAC() syntax element from the previous authentication sequence (as indicated by a different value of authSequence) may be included into the calculation of the current authentication sequence, which allows verification of temporal consistency of the data.

**4.5.2.18 Payloads for the audio object type IIF**

**4.5.2.18.1 Semantics**

|  |  |
| --- | --- |
| **frame\_counter** | Indicates the IIF frame to which this IIF block belongs to. The value shall be counted from 0 to 15 and wrap around to 0. |
| **block\_id** | Indicates the current IIF block. The block ID links a block to the corresponding block stream. Values shall be strictly contiguous and in ascending order. |
| **block\_priority** | Indicates the priority of the IIF block, 0 being the highest priority. A value of ‘0’ shall only be used for the IIF blocks belonging to the current IIF frame in an access unit. There shall only be one IIF block of each block\_id with a block\_priority value of ‘0’ belonging to one IIF frame.  A value of ‘1’ shall be used to signal the first layer of redundant IIF blocks. There may be multiple IIF blocks with the same block\_id and a block\_priority value of ‘1’ belonging to one IIF frame.  A value of ‘2’ shall be used to signal the second layer of redundant IIF blocks. There may be multiple IIF blocks with the same block\_id and a block\_priority value of ‘2’ belonging to one IIF frame.  A value of ‘3’ shall be used to signal the third layer of redundant IIF blocks. There may be multiple IIF blocks with the same block\_id and a block\_priority value of ‘3’ belonging to one IIF frame. |
| **block\_size** | Indicates the size of the IIF block in bytes, including the first element after the reserved element and all elements of iif\_block\_footer() element. The total size of a block is therefore block\_size+3 bytes. |
| **b\_block\_protected** | Indicates if the block is protected by a CRC. The value shall be set to ‘0’ in the case of a CRC not being present. The value shall be set to ‘1’ in the case of a CRC being present. |
| **block\_size\_extension** | Indicates the extension size of the IIF block in bytes. |
| **b\_more\_bits** | Indicates if there’s further block size extension in the bitstream. |
| **b\_config\_update\_inband** | Indicates if the config update is included in-band. |
| **dynamic\_config\_upd\_idx** | Indicates the index of the configuration update. A value of ‘0’ is reserved for inband metadata and shall not to be used. |
| **md\_type\_idx** | Indicates the index of the metadata type. |
| **b\_more\_md\_sections** | Indicates if there are more sections of metadata present. |
| **b\_metadata\_present** | Indicates if metadata are present. |
| **b\_more\_elements** | Indicates continuation of the related while-loop. |
| **b\_element\_is\_cpe** | Indicates if the element is a CPE or SCE. |
| **md\_size\_in\_bits** | Indicates the size of the metadata in bits. |
| **ext\_md\_size\_in\_bits** | Indicates the size-extension of the metadata in bits. |
| **unknown\_metadata** | Contains bits for unknown types of metadata. |
| **active\_channel\_config\_idx** | Indicates the active channel configuration for the current IIF frame as index into the channel\_config\_set\_channel\_mode and channel\_config\_set\_channel\_mask array. |
| **b\_active\_channel\_config\_change** | Indicates if there is a change of the active channel configuration present in the current IIF frame.  A value of ‘0’ indicates that there is no channel configuration change in the current IIF frame. The channel configuration set by active\_channel\_config\_idx shall be valid for the duration of the entire frame.  A value of ‘1’ indicates that there is a channel config change within the current IIF frame. |
| **next\_channel\_config\_sample\_**  **offset\_minus1** | Indicates after which sample, counting from the first sample of the associated IIF Block, the new channel configuration identified by the next\_channel\_config\_idx field shall be active. |
| **next\_channel\_config\_idx** | Indicates the active channel configuration for the current IIF frame, that shall be valid after the configuration change. |
| **b\_override\_defaults** | Indicates that all default values of the device specific routing are overridden for this IIF frame. |
| **b\_signal\_connected** | Indicates if a signal is connected and a role is assigned to a specific device. |
| **audio\_signal\_role** | Indicates the role of the audio signal according to Table 4.ASR. |
| **Table 4.ASR – Values of the audio\_signal\_role field**   |  |  |  | | --- | --- | --- | | Symbol | Value of audio\_signal\_role | Purpose | | AUDIO\_SIGNAL\_ROLE\_MAIN | ‘000’ | Main audio. | | AUDIO\_SIGNAL\_ROLE\_AUX1 | ‘001’ | Auxiliary audio. | | AUDIO\_SIGNAL\_ROLE\_AUX2 | ‘010’ | Auxiliary audio. | | AUDIO\_SIGNAL\_ROLE\_AUX3 | ‘011’ | Auxiliary audio. | | AUDIO\_SIGNAL\_ROLE\_AUX4 | ‘100’ | Auxiliary audio. | | AUDIO\_SIGNAL\_ROLE\_AUX5 | ‘101’ | Auxiliary audio. | | AUDIO\_SIGNAL\_ROLE\_AUX6 | ‘110’ | Auxiliary audio. | | AUDIO\_SIGNAL\_ROLE\_AUX7 | ‘111’ | Auxiliary audio. | | |
|  |  |
| **b\_override\_default\_params\_for\_device** | Indicates if gain- or delay-values are signaled for the related signal and device in this IIF frame.  A value of ‘0’ indicates that all default gain- and delay-values shall be applied for the related device in this IIF frame.  A value of ‘1’ indicates that gain- and/or delay-values are signaled. |
| **b\_override\_default\_gain** | Indicates if a gain is signaled for the related signal and device in this IIF frame.  A value of ‘0’ indicates that the default gain value shall be applied for the related device in this IIF frame.  A value of ‘1’ indicates that a gain value is signaled. |
| **gain\_code** | Indicates the gain code to derive the related audio signal gain. The audio signal gain value shall be calculated as follows: |
| **b\_override\_default\_delay** | Indicates if the default gain shall be applied to the audio signal s on device d.  A value of ‘0’ indicates that the default delay shall be applied.  A value of ‘1’ indicates that a delay is signaled for the audio signal s on the device d. |
| **b\_relative\_delay** | Indicates if the signaled delay is relative to the default delay.  A value of ‘0’ indicates that a new absolute delay is signaled that shall replace the default delay.  A value of ‘1’ indicates that a relative delay is signaled which shall be added to the default delay to determine the delay to be applied to the current IIF frame. |
| **rel\_delay\_code** | Indicates the relative delay code. The relative delay in samples shall be calculated as follows:  The delay in samples of the signal for device shall be calculated as follows: |
| **delay\_code** | Indicates the absolute delay code. The delay in samples of the signal s for device d shall be calculated as follows: |
| **block\_fill** | Indicates fill bits which shall be ignored. |
| **block\_crc** | Indicates a 16-bit CRC which shall be calculated over the concatenated data of iif\_block\_header(), iif\_block\_payload(), and the block\_fill element, as defined in 1.8.4.5 (CRC16). |

In 4.5.4 Tables add before Table 4.166 – AAC error sensitivity category assignment for main payload:

**Table 4.150 – scalefactor bands for a window length of 512 for LONG\_WINDOW**

|  |  |
| --- | --- |
| num\_swb\_long\_window | 20 |
| swb | swb\_offset\_lon  g\_window |
| 0 | 0 |
| 1 | 4 |
| 2 | 8 |
| 3 | 12 |
| 4 | 16 |
| 5 | 20 |
| 6 | 24 |
| 7 | 28 |
| 8 | 36 |
| 9 | 44 |
| 10 | 52 |
| 11 | 64 |
| 12 | 76 |
| 13 | 92 |
| 14 | 108 |
| 15 | 128 |
| 16 | 148 |
| 17 | 172 |
| 18 | 196 |
| 19 | 244 |
|  | 256 |

**Table 4.150 – scalefactor bands for a window length of 480 for LONG\_WINDOW**

|  |  |
| --- | --- |
| num\_swb\_long\_window | 20 |
| swb | swb\_offset\_lon  g\_window |
| 0 | 0 |
| 1 | 4 |
| 2 | 8 |
| 3 | 12 |
| 4 | 16 |
| 5 | 20 |
| 6 | 24 |
| 7 | 28 |
| 8 | 32 |
| 9 | 40 |
| 10 | 48 |
| 11 | 56 |
| 12 | 68 |
| 13 | 80 |
| 14 | 96 |
| 15 | 112 |
| 16 | 132 |
| 17 | 156 |
| 18 | 180 |
| 19 | 208 |
|  | 240 |

For the AAC Immersive Interchange Format, the scalefactor bands of Table 4.148 are reused for LONG\_WINDOW (256/240 window size).

After 4.6.20, add the following:

* + 1. ***AAC Immersive Interchange Format***
       1. **Introduction**

The AAC Immersive Interchange Format is an audio codec that enables reliable immersive audio transmission via transport links at low latency. It enables carriage of different types of metadata, which can be associated with the audio data. Further, it includes signaling which allows flexible handling of various use cases.

* + - 1. **Coder description**

The core codec of the AAC Immersive Interchange Format is defined by the following modifications with respect to the Low delay codec algorithm as described in 4.6.17.

* + - 1. **Frame size/window length**

In addition to 1024 and 960 windows (512 and 480 spectral values), the following window and frame sizes shall be supported:

* 2048/1920 resulting in 1024/960 spectral values
* 512/480 resulting in 256/240 spectral values
* 256/240 resulting in 128/120 spectral values
  + - 1. **Window shape**

The sine window (window\_shape==0) as defined in 4.6.11.3.2 and low-overlap window (window\_shape==1) as defined in 4.6.17.2.3 shall be reused. Additional values of N for the calculation of the sine window shall be 1024, 960, 512 and 480. Additional values of N for the calculation of the low-overlap window shall be 2048, 1920, 512, 480, 256 and 240.

* + - 1. **Tables for temporal noise shaping (TNS)**

The following tables specify additional value of TNS\_MAX\_BANDS for the core codec of the AAC Immersive Interchange Format:

**Table xxx – TNS\_MAX\_BANDS in case of 128 and 120 samples per frame**

|  |  |
| --- | --- |
| Sampling rate | TNS\_MAX\_BANDS |
| 48000 | 14 |
| 44100 | 14 |

**Table xxx – TNS\_MAX\_BANDS in case of 256 and 240 samples per frame**

|  |  |
| --- | --- |
| Sampling rate | TNS\_MAX\_BANDS |
| 48000 | 17 |
| 44100 | 17 |

**Table xxx – TNS\_MAX\_BANDS in case of 1024 and 960 samples per frame**

|  |  |
| --- | --- |
| Sampling rate | TNS\_MAX\_BANDS |
| 48000 | 40 |
| 44100 | 40 |

* + - 1. **Definitions**

|  |  |
| --- | --- |
| **IIF Session** | A IIF Session starts with setting up an IIF Stream with a configuration and lasts until the session is stopped or needs to be reconfigured. |
| **IIF Frame** | A IIF Frame is a portion of the signal belonging to a specific period of time. An IIF Frame has a start time, and end time, and a duration (the difference of end time and start time). The start time of the coded IIF Frame shall be aligned with the first sample of the decoded PCM signal resulting from decoding that IIF Frame. The end time of the coded IIF Frame shall be aligned with the last sample of the decoded PCM signal resulting from decoding that IIF Frame. An IIF Frame is a concept which is not reflected in the syntax. |
| **IIF Block** | A IIF Block is the coded representation of one complete IIF Frame or a part of an IIF Frame and the basic building block of the IIF bitstream. If the coded representation of the audio signals in one IIF Frame is split over multiple IIF Blocks, then those IIF Blocks shall have different values of block\_id. IIF Blocks of subsequent IIF Frames with the same value of block\_id shall carry the same signal (e.g. coded using different bitrates). The decoder can traverse the IIF bitstream block-wise and therefore only process relevant (e.g., for a device) blocks. The block header contains all relevant information about the block, e.g., the block ID and size. |
| **IIF Block Stream** | A IIF Block Stream is comprised of all blocks of the same block ID across all IIF access units. |
| **IIF Access Unit** | An IIF Access Unit is a piece of binary data consisting of one or more IIF Blocks carrying the coded representation of the signal belonging to the current or a previous IIF Frame. One IIF Access Unit contains the full signal presentation of the current IIF Frame, it may contain redundant blocks belonging to a previous IIF Frame.  The presentation time of an IIF Access Unit shall be the presentation time of the current IIF Frame in that IIF Access Unit, which is identified by the IIF Blocks with a block\_priority value of ‘0’ (highest priority). |
| **IIF Packet** | A IIF Packet is a concept defining the unit of transmission over a packet-based network. In the context of this codec, a packet contains either one or multiple complete IIF Access Units, or one or multiple IIF Blocks all belonging to the same IIF Access Unit. |
| **IIF Audio Bus** | The IIF Audio Bus is a concept that allows mapping of incoming audio signals to either a predefined audio presentation (i.e., channels, objects or HOA) and/or to be used by devices in selected roles (e.g. as a driver feed or echo reference).  The signals of all block streams are routed to the IIF Audio Bus ordered by block ID and respective channel order (if blocks carry more than one channel). The IIF Audio Bus supports two sections:   1. Predefined audio presentation represents a full presentation, i.e., channels or objects or HOA. 2. Device specific routing assigns signals to devices and drivers via a routing matrix and assigns roles (i.e., main or auxiliary).   A IIF Session may be configured to only contain a predefined audio presentation, device specific routing or a combination of both.  The flexibility in mapping incoming signals onto the IIF Audio Bus allows for continuous audio signals even if the active channel configuration changes. Therefore, the assignment of IIF Blocks and channel elements to the IIF Audio Bus shall be stable over the duration of a IIF Session, while some audio elements (e.g., channels, objects) or IIF Blocks may drop in or out intentionally, because of channel mode changes, or unintentionally because of packet losses. |

* + - 1. **iif\_specific\_config**

**Table 4.XXY — Syntax of iif\_specific\_config**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| iif\_specific\_config(base\_sampling\_rate) |  |  |
| { |  |  |
| iif\_frame\_properties\_config(base\_sampling\_rate); |  |  |
| iif\_audio\_bus\_source\_config(base\_sampling\_rate); |  |  |
| iif\_predef\_audio\_presentation\_config(); |  |  |
| iif\_block\_properties\_config(); |  |  |
| iif\_device\_specific\_routing\_config(); |  |  |
|  |  |  |
| /\* LATENCY INFO \*/ |  |  |
| **max\_interleaving\_depth;** | **3** | **uimsbf** |
|  |  |  |
| **n\_config\_updates;** | **3** | **uimsbf** |
| for (u=0; u<n\_config\_updates; u++) { |  |  |
| **dynamic\_config\_upd\_idx;** | **3** | **uimsbf** |
| iif\_config\_update (dynamic\_config\_upd\_idx); |  |  |
| } |  |  |
| } |  |  |

**Table 4.XXY — Syntax of iif\_frame\_properties\_config**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| iif\_frame\_properties\_config(base\_sampling\_rate) |  |  |
| { |  |  |
| **frame\_length\_idx;** | **5** | **uimsbf** |
| **out\_sampling\_rate\_multiplier\_exponent;** | **2** | **uimsbf** |
| out\_sampling\_rate\_multiplier = (1 <<  out\_sampling\_rate\_multiplier\_exponent); |  |  |
| out\_sampling\_rate = base\_sampling\_rate \* out\_sampling\_rate\_multiplier; |  |  |
| } |  |  |

**Table 4.XXY — Syntax of iif\_audio\_bus\_source\_config**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| iif\_audio\_bus\_source\_config(base\_sampling\_rate) |  |  |
| { |  |  |
| /\* AUDIO BUS SOURCE INFO \*/ |  |  |
| **audio\_bus\_width\_minus1;** | **8** | **uimsbf** |
| audio\_bus\_width = audio\_bus\_width\_minus1 + 1; |  |  |
| idx = 0; |  |  |
| offs = 0; |  |  |
| audio\_bus\_input\_block[0] = 0; |  |  |
| b\_add\_metadata\_block = 0; |  |  |
| while (offs < audio\_bus\_width || b\_add\_metadata\_block != 0) |  |  |
| { |  |  |
| /\* the number of signals per block is limited by the capacity of the |  |  |
| audio\_payload() \*/ |  |  |
| **audio\_signals\_in\_source\_block[idx];** | **5** | **uimsbf** |
| if (audio\_signals\_in\_source\_block[idx] > 0) { |  |  |
| **block\_sampling\_rate\_multiplier\_exponent;** | **2** | **uimsbf** |
| block\_sampling\_rate\_multiplier = (1 <<  block\_sampling\_rate\_multiplier\_exponent); |  |  |
| block\_sampling\_rate[idx] = base\_sampling\_rate \*  block\_sampling\_rate\_multiplier; |  |  |
| for (i=0; i<audio\_signals\_in\_source\_block[idx]; i++) |  |  |
| { |  |  |
| audio\_bus\_input\_block[i+offs] = idx; |  |  |
| } |  |  |
| } |  |  |
|  |  |  |
| **num\_metadata\_types[idx];** | **4** | **uimsbf** |
| for (m=0; m<num\_metadata\_types[idx]; m++){ |  |  |
| **inband\_metadata\_type[idx][m];** | **8** | **uimsbf** |
| if (inband\_metadata\_type[idx][m] == 0xFF){ |  |  |
| **inband\_metadata\_type\_ext** | **8** | **uimsbf** |
| inband\_metadata\_type[idx][m] +=  inband\_metadata\_type\_ext; |  |  |
| } |  |  |
| } |  |  |
| **use\_explicit\_metadata\_signaling[idx];** | **1** | **uimsbf** |
|  |  |  |
|  |  |  |
| audio\_bus\_offset\_for\_block[idx] = offs; |  |  |
| offs += audio\_signals\_in\_source\_block[idx]; |  |  |
| idx += 1; |  |  |
| if (offs == audio\_bus\_width) { |  |  |
| **b\_add\_metadata\_block**; | **1** | **uimsbf** |
| } |  |  |
| } |  |  |
| num\_block\_ids = idx; |  |  |
| } |  |  |

**Table 4.XXY — Syntax of iif\_predef\_audio\_presentation\_config**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| iif\_predef\_audio\_presentation\_config() |  |  |
| { |  |  |
| CHANNEL\_MASK\_BITS = [15,32,43,128]; |  |  |
|  |  |  |
| num\_signals\_for\_audio\_presentation = 0; |  |  |
| channel\_mask\_superset = 0; |  |  |
| num\_hoa\_signals = 0; |  |  |
| object\_metadata\_id = 0; |  |  |
| num\_audio\_objects = 0; |  |  |
|  |  |  |
| **audio\_presentation\_type;** | **2** | **uimsbf** |
| switch (audio\_presentation\_type) { |  |  |
| case AUDIO\_PRESENTATION\_TYPE\_CHANNELS: |  |  |
| **channel\_mode\_superset;**  /\* according to ISO/IEC 23091-3:2018, Table 3 \*/ | **5** | **uimsbf** |
| if (channel\_mode\_superset == 0) { |  |  |
| **channel\_mask\_width;** | **2** | **uimsbf** |
| **channel\_mask\_superset;**  /\* according to ISO/IEC 23091-3:2018, Table-2 \*/ | **CHANNEL\_MASK\_BITS[channel\_mask\_width]** | **uimsbf** |
| } |  |  |
| else { |  |  |
| channel\_mask\_superset =  get\_channel\_mask\_for\_channel\_config  (channel\_mode\_superset); |  |  |
| } |  |  |
| num\_audio\_channels =  count\_audio\_channels(channel\_mask\_superset); |  |  |
| num\_signals\_for\_audio\_presentation += num\_audio\_channels; |  |  |
|  |  |  |
| channel\_config\_set\_channel\_mode[0] = channel\_mode\_superset; |  |  |
| channel\_config\_set\_channel\_mask[0] = channel\_mask\_superset; |  |  |
| channel\_config\_set\_mix\_matrix[0] =  init\_mix\_matrix(channel\_mask\_superset); |  |  |
|  |  |  |
| **num\_channel\_config\_subsets;** | **3** | **uimsbf** |
| for (i=0; i<num\_channel\_config\_subsets; i++) { |  |  |
| **channel\_config\_set\_channel\_mode[i+1];**  /\* according to ISO/IEC 23091-3:2018, Table 3 \*/ | **5** | **uimsbf** |
| if (channel\_config\_set\_channel\_mode[i+1] == 0) { |  |  |
| **channel\_mask\_width;** | **2** | **uimsbf** |
| **channel\_config\_set\_channel\_mask[i+1];**  /\* according to ISO/IEC 23091-3:2018, Table-2 \*/ | **CHANNEL\_MASK\_BITS[channel\_mask\_width]** | **uimsbf** |
| } |  |  |
| else { |  |  |
| channel\_config\_set\_channel\_mask[i+1] = |  |  |
| get\_channel\_mask\_for\_channel\_config  (channel\_config\_set\_channel\_mode[i+1]); |  |  |
| } |  |  |
|  |  |  |
| channel\_config\_set\_mix\_matrix[i+1] =  init\_mix\_matrix(channel\_mask\_superset); |  |  |
| channels\_not\_in\_super =  channel\_config\_set\_channel\_mask[i+1] -  (channel\_config\_set\_channel\_mask[i+1] &  channel\_mask\_superset); |  |  |
| if (channels\_not\_in\_super > 0) { |  |  |
| channel\_subset\_mix\_info(channels\_not\_in\_super, i+1); |  |  |
| } |  |  |
| } |  |  |
| break; |  |  |
|  |  |  |
| case AUDIO\_PRESENTATION\_TYPE\_HOA: |  |  |
| **hoa\_order\_minus\_one;** | **3** | **uimsbf** |
| num\_hoa\_signals = 1 << (hoa\_order\_minus\_one + 2); |  |  |
| num\_signals\_for\_audio\_presentation += num\_hoa\_signals; |  |  |
| break; |  |  |
|  |  |  |
| case AUDIO\_PRESENTATION\_TYPE\_OBJECTS: |  |  |
| **object\_metadata\_id;** | **8** | **uimsbf** |
| **num\_audio\_objects\_minus1;** | **8** | **uimsbf** |
| num\_audio\_objects = num\_audio\_objects\_minus\_one + 1; |  |  |
| num\_signals\_for\_audio\_presentation += num\_audio\_objects; |  |  |
| break; |  |  |
|  |  |  |
| case AUDIO\_PRESENTATION\_TYPE\_NONE: |  |  |
| break; |  |  |
| } |  |  |
| } |  |  |

**Table 4.XXY — Syntax of iif\_block\_properties\_config**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| iif\_block\_properties\_config() |  |  |
| { |  |  |
| num\_signals\_for\_device\_routing = audio\_bus\_width –  num\_signals\_for\_audio\_presentation; |  |  |
| if (num\_signals\_for\_device\_routing > 0) { |  |  |
| **num\_target\_devices;** | **4** | **uimsbf** |
|  |  |  |
| if (num\_target\_devices > 0) { |  |  |
| **b\_enable\_device\_addressing;** | **1** | **uimsbf** |
| /\* num\_blocks\_ids is already defined at Audio Bus Source Info \*/ |  |  |
| for (i=0; i<num\_block\_ids; i++) { |  |  |
| if (b\_enable\_device\_addressing == 1) { |  |  |
| **b\_address\_to\_all\_devices[i];**  /\* add a target adressing mask, 0=all targets \*/ | **1** | **uimsbf** |
| if (b\_address\_to\_all\_devices[i] == 0) { |  |  |
| **device\_address\_mask[i];** | **num\_target\_devices bits** |  |
| } |  |  |
| else { |  |  |
| device\_address\_mask[i] = (1<<num\_target\_devices)  – 1; |  |  |
| } |  |  |
| } |  |  |
| else { |  |  |
| b\_address\_to\_all\_devices[i] = 1; |  |  |
| device\_address\_mask[i] = (1<<num\_target\_devices) - 1; |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| else { |  |  |
| b\_enable\_device\_addressing = 0; |  |  |
| } |  |  |
| } |  |  |
| else { |  |  |
| num\_target\_devices = 0; /\* this implies broadcast mode \*/ |  |  |
| } |  |  |
|  |  |  |
| } |  |  |

**Table 4.XXY — Syntax of iif\_device\_specific\_routing\_config**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| iif\_device\_specific\_routing\_config() |  |  |
| { |  |  |
| for (d=0; d<num\_target\_devices; d++) { |  |  |
| for (s=0; s<num\_signals\_for\_device\_routing; s++) { |  |  |
| \_offs = num\_signals\_for\_audio\_presentation + s; |  |  |
| \_block = audio\_bus\_input\_block[\_offs]; |  |  |
| audio\_signal\_bus\_offs[d][s] = \_offs; |  |  |
| if (device\_address\_mask[\_block] & (1<<d)) { |  |  |
| /\* data are only present for audio signals on the bus |  |  |
| delivered by blocks addressed to device d \*/ |  |  |
| bus\_to\_device\_mapping\_defaults(d,s); |  |  |
| } |  |  |
| else { |  |  |
| default\_audio\_signal\_role[d][s] =  AUDIO\_SIGNAL\_NO\_ROLE; |  |  |
| default\_audio\_signal\_gain[d][s] = 0; |  |  |
| default\_audio\_signal\_delay[d][s] = 0; |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| } |  |  |

**Table 4.XXY — Syntax of bus\_to\_device\_mapping\_defaults**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| bus\_to\_device\_mapping\_defaults(device, bus) |  |  |
| { |  |  |
| **b\_default\_signal\_connected;** | **1** | **uimsbf** |
| if (b\_default\_signal\_connected == 1) { |  |  |
| **default\_audio\_signal\_role[device][bus];** | **3** | **uimsbf** |
| } |  |  |
| else { |  |  |
| default\_audio\_signal\_role[device][bus] = AUDIO\_SIGNAL\_NO\_ROLE; |  |  |
| } |  |  |
| /\* allow setting defaults for gain and delay for all signals, |  |  |
| even if not connected by default \*/ |  |  |
| default\_audio\_signal\_gain[device][bus] = 0; |  |  |
| **b\_set\_default\_custom\_gain;** | **1** | **uimsbf** |
| if (b\_set\_default\_custom\_gain) { |  |  |
| **gain\_code;** | **5** | **uimsbf** |
| if (gain\_code < 24) { |  |  |
| default\_audio\_signal\_gain[device][bus] = gain\_code - 24; |  |  |
| } |  |  |
| else { |  |  |
| default\_audio\_signal\_gain[device][bus] = gain\_code - 23; |  |  |
| } |  |  |
| } |  |  |
| default\_audio\_signal\_delay[device][bus] = 0; |  |  |
| **b\_set\_default\_custom\_delay;** | **1** | **uimsbf** |
| if (b\_set\_default\_custom\_delay) { |  |  |
| **delay\_code;** | **12** | **uimsbf** |
| default\_audio\_signal\_delay[device][bus] = 4\*delay\_code; |  |  |
| } |  |  |
| } |  |  |

**Table 4.XXY — Syntax of channel\_subset\_mix\_info**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| channel\_subset\_mix\_info(\_channels\_not\_in\_super\_arg, \_set) |  |  |
| { |  |  |
| src\_ch\_idx = 0; |  |  |
| \_channels\_not\_in\_super = \_channels\_not\_in\_super\_arg; |  |  |
| while (\_channels\_not\_in\_super > 0) { |  |  |
| if ((\_channels\_not\_in\_super & 1) == 0x1) { |  |  |
| **map\_or\_mix;** | **1** | **uimsbf** |
| **is\_channel\_pair;** | **1** | **uimsbf** |
| n\_mixing\_coeffs = map\_or\_mix + 1; |  |  |
| for (m=0; m<n\_mixing\_coeffs; m++) { |  |  |
| **upmix\_target\_ch\_idx;** | **7** | **uimsbf** |
| **upmix\_gain;** | **3** | **uimsbf** |
|  |  |  |
| ch\_offs = 0; |  |  |
| \_ch\_mask = channel\_mask\_superset; |  |  |
| \_ch\_idx = upmix\_target\_ch\_idx; |  |  |
| while (\_ch\_idx > 0) |  |  |
| { |  |  |
| if ((\_ch\_mask & 0x1) == 0x1) { |  |  |
| ch\_offs += 1; |  |  |
| } |  |  |
| \_ch\_mask = \_ch\_mask >> 1; |  |  |
| \_ch\_idx = \_ch\_idx - 1; |  |  |
| } |  |  |
|  |  |  |
|  |  |  |
| channel\_config\_set\_mix\_matrix[\_set][ch\_offs][3\*m+0] =  src\_ch\_idx; |  |  |
| channel\_config\_set\_mix\_matrix[\_set][ch\_offs][3\*m+1] =  upmix\_target\_ch\_idx; |  |  |
| channel\_config\_set\_mix\_matrix[\_set][ch\_offs][3\*m+2] =  upmix\_gain; |  |  |
| if (is\_channel\_pair == 1) { |  |  |
| ch\_offs += 1; |  |  |
| channel\_config\_set\_mix\_matrix[\_set][ch\_offs][3\*m+0] =  src\_ch\_idx+1; |  |  |
| channel\_config\_set\_mix\_matrix[\_set][ch\_offs][3\*m+1] =  upmix\_target\_ch\_idx+1; |  |  |
| channel\_config\_set\_mix\_matrix[\_set][ch\_offs][3\*m+2] =  upmix\_gain; |  |  |
| } |  |  |
| } |  |  |
| if (is\_channel\_pair == 1) { |  |  |
| \_channels\_not\_in\_super = \_channels\_not\_in\_super >> 1; |  |  |
| src\_ch\_idx += 1; |  |  |
| } |  |  |
| } |  |  |
| \_channels\_not\_in\_super = \_channels\_not\_in\_super >> 1; |  |  |
| src\_ch\_idx += 1; |  |  |
| } |  |  |
| } |  |  |

**Table 4.XXY — Syntax of init\_mix\_matrix**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| init\_mix\_matrix(\_channel\_mask\_superset\_arg) |  |  |
| { |  |  |
| \_ch\_cnt = 0; |  |  |
| \_ch\_idx = 0; |  |  |
| \_mix\_mtrx = []; |  |  |
| \_channel\_mask\_superset = \_channel\_mask\_superset\_arg; |  |  |
| while (\_channel\_mask\_superset > 0) { |  |  |
| if ((\_channel\_mask\_superset & 0x1) == 0x1) { |  |  |
| \_mix\_mtrx[\_ch\_cnt] = [\_ch\_idx, \_ch\_idx, 0]; |  |  |
| \_ch\_cnt += 1; |  |  |
| } |  |  |
| \_channel\_mask\_superset = \_channel\_mask\_superset >> 1; |  |  |
| \_ch\_idx += 1; |  |  |
| } |  |  |
| return \_mix\_mtrx; |  |  |
| } |  |  |

**Table 4.XXY — Syntax of count\_audio\_channels**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| count\_audio\_channels(\_channel\_mask) |  |  |
| { |  |  |
| \_audio\_channels = 0; |  |  |
| \_ch\_mask = \_channel\_mask; |  |  |
| while (\_ch\_mask > 0) { |  |  |
| \_audio\_channels += (\_ch\_mask & 0x1); |  |  |
| \_ch\_mask = \_ch\_mask >> 1; |  |  |
| } |  |  |
| return \_audio\_channels; |  |  |
| } |  |  |

**Table 4.XXY — Syntax of get\_channel\_mask\_for\_channel\_config**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| get\_channel\_mask\_for\_channel\_config(\_channel\_config) |  |  |
| { |  |  |
| channel\_mask\_for\_channel\_mode = [ |  |  |
| 0, /\* 0: \*/ |  |  |
| 0b100, /\* 1: C (1.0) \*/ |  |  |
| 0b11, /\* 2: L,R (2.0) \*/ |  |  |
| 0b111, /\* 3: L,R,C (3.0) \*/ |  |  |
| 0b10000000111, /\* 4: L,R,C,Cs \*/ |  |  |
| 0b110111, /\* 5: L,R,C,Ls,Rs (5.0) \*/ |  |  |
| 0b111111, /\* 6: L,R,C,LFE,Ls,Rs (5.1) \*/ |  |  |
| 0b11111111, /\* 7: L,R,C,LFE,Ls,Rs,Lc,Rc \*/ |  |  |
| 0, /\* 8: Dual-mono is unsupported! \*/ |  |  |
| 0b10000000011, /\* 9: L,R,Cs \*/ |  |  |
| 0b110011, /\* 10: L,R,Ls,Rs \*/ |  |  |
| 0b10000111111, /\* 11: L,R,C,LFE,Ls,Rs,Cs \*/ |  |  |
| 0b1100111111, /\* 12: L,R,C,LFE,Ls,Rs,Lsr,Rsr (7.1) \*/ |  |  |
| 0b00111111111111100110011111001111, /\* 13: L,R,C,LFE,Lc,Rc,Lsr,Rsr,Cs,Lss,Rss,Lv,Rv,Cv,Lvr,Rvr,Cvr,Lvss,Rvss,Ts,LFE2,Lb,Rb,Cb (22.2) \*/ |  |  |
| 0b1100000000000111111, /\* 14: L,R,C,LFE,Ls,Rs,Lv,Rv \*/ |  |  |
| 0b100010001100110000000111111, /\* 15: L,R,C,LFE,Ls,Rs,Lss,Rss,Lv,Rv,Cvr,LFE2 \*/ |  |  |
| 0b11000000000001100000000000111111, /\* 16: L,R,C,LFE,Ls,Rs,Lv,Rv,Lvs,Rvs (5.1.4) \*/ |  |  |
| 0b11000010000011100000000000111111, /\* 17: L,R,C,LFE,Ls,Rs,Lv,Rv,Cv,Ts,Lvs,Rvs \*/ |  |  |
| 0b1100000000011000010000011100000000000111111, /\* 18: L,R,C,LFE,Ls,Rs,Lv,Rv,Cv,Ts,Lvs,Rvs,Lbs,Rbs \*/ |  |  |
| 0b1101100110001100001111, /\* 19: L,R,C,LFE,Lsr,Rsr,Lss,Rss,Lv,Rv,Lvr,Rvr (7.1.4) \*/ |  |  |
| 0b110000011000000000001100110001100001111, /\* 20: L,R,C,LFE,Lsr,Rsr,Lss,Rss,Lv,Rv,Lvs,Rvs,Leos,Reos \*/ |  |  |
| 0, /\* 21: Reserved \*/ |  |  |
| 0, /\* 22: Reserved \*/ |  |  |
| 0, /\* 23: Reserved \*/ |  |  |
| 0, /\* 24: Reserved \*/ |  |  |
| 0, /\* 25: Reserved \*/ |  |  |
| 0, /\* 26: Reserved \*/ |  |  |
| 0, /\* 27: Reserved \*/ |  |  |
| 0, /\* 28: Reserved \*/ |  |  |
| 0, /\* 29: Reserved \*/ |  |  |
| 0, /\* 30: Reserved \*/ |  |  |
| 0 /\* 31: Reserved \*/ |  |  |
| ]; |  |  |
|  |  |  |
| \_ch\_mask2 = 0; |  |  |
| if (\_channel\_config < 32) { |  |  |
| \_ch\_mask2 = channel\_mask\_for\_channel\_mode[\_channel\_config]; |  |  |
| } |  |  |
| return \_ch\_mask2; |  |  |
| } |  |  |

* + - 1. **Semantics of iif\_specific\_config**

|  |  |
| --- | --- |
| **frame\_length\_idx** | Indicates the frame length in audio samples for the IIF stream, according to Table 4.xxx. |
| **Table 4.xxx – Values of the frame\_length\_idx field**   |  |  | | --- | --- | | Value of frame\_length\_idx | IIF frame length in samples | | 0 | 120 | | 1 | 240 | | 2 | 480 | | 3 | 960 | | 4 | 128 | | 5 | 256 | | 6 | 512 | | 7 | 1024 | | 8 | 192 | | 9 | 384 | | 10 | 768 | | 11-31 | Reserved for ISO use | | |
|  | |
| **out\_sampling\_rate\_multiplier\_exponent** | Indicates the exponent to the base of 2 of the out\_sampling\_rate\_multiplier to be applied to the base\_sampling\_rate. For values of ‘0’-‘2’, the output sampling rate shall be calculated as follows:  The value of 3 shall not be used. |
| **audio\_bus\_width\_minus1** | Plus 1 indicates the width of the IIF Audio Bus. |
| **audio\_signals\_in\_source\_block** | Indicates the number of audio signals delivered by the IIF block with the block\_id matching the counting index. A value of ‘0’ (zero) indicates that a block does not carry any audio data. |
| **block\_sampling\_rate\_multiplier\_exponent** | Indicates the exponent to the base of 2 of the block\_sampling\_rate\_multiplier to be applied to the base\_sampling\_rate. This value shall not exceed the value of the out\_sampling\_rate\_multiplier\_exponent element.  For values of ‘0’-‘2’, the output sampling rate shall be calculated as follows:  The value of 3 shall not be used. |
| **num\_metadata\_types** | Indicates the number of metadata types. |
| **inband\_metadata\_type** | Indicates the type of the metadata as according to Table 4.xxx. |
| **Table 4.xxx – Values of the inband\_metadata\_type field**   |  |  | | --- | --- | | Value of inband\_metadata\_type | Metadata type | | 0 | INBAND\_METADATA\_TYPE\_AUTH | | 1-127 | Reserved for ISO use. | | 128-510 | Reserved for use outside of ISO scope | | |
|  |  |
| **inband\_metadata\_type\_ext** | Indicates an extension to the value of inband\_metadata\_type. |
| **use\_explicit\_metadata\_signaling** | Indicates if metadata are signaled explicitly. |
| **b\_add\_metadata\_block** | Indicates that another block stream is configured for metadata only. |
| **audio\_presentation\_type** | Indicates the type of the audio presentation according to the Table 4.xxx. |
| **Table 4.xxx – Values of the audio\_presentation\_type field**   |  |  |  | | --- | --- | --- | | Symbol | Value of audio\_presentation\_type | Purpose | | AUDIO\_PRESENTATION\_TYPE\_NONE | ‘00’ | No signals are assigned to a predefined audio presentation. | | AUDIO\_PRESENTATION\_TYPE\_CHANNELS | ‘01’ | Audio presentation type channels. | | AUDIO\_PRESENTATION\_TYPE\_HOA | ’10‘ | Audio presentation type HOA. | | AUDIO\_PRESENTATION\_TYPE\_OBJECTS | ‘11’ | Audio presentation type objects. | | |
|  |  |
| **channel\_mode\_superset** | Indicates the channel mode superset according to ISO/IEC 23091-3:2018, Table 3. |
| **channel\_mask\_width** | Identifies the number of least significant bits signaled for the channel mask according to Table 4.xxx. All bits with higher significance shall be ‘0’. |
| **Table 4.xxx – Values of the channel\_mask\_width field**   |  |  | | --- | --- | | Value of channel\_mask\_width | Number of least significant bits | | 0 | 15 | | 1 | 32 | | 2 | 43 | | 3 | 128 | | |
|  |  |
| **channel\_mask\_superset** | Indicates selected channels via a bitmask. Selected channels are indicated by setting the respective bit of the bitmask to a value of ‘1’ in accordance with ISO/IEC 23091-3:2018, Table-2, where the significance of the bit represents the “Value” of the respective channel in ISO/IEC 23091-3:2018, Table-2. |
| **num\_channel\_config\_subsets** | Indicates the number of channel configuration subsets. |
| **channel\_config\_set\_channel\_mode** | Indicates the channel mode for the respective channel config set, according to ISO/IEC 23091-3:2018, Table 3. |
| **channel\_config\_set\_channel\_mask** | Indicates the channel mask for the respective channel config set, according to ISO/IEC 23091-3:2018, Table 3. |
| **hoa\_order\_minus1** | Plus 1 indicates the order of the higher order ambisonics (HOA) signal. |
| **object\_metadata\_id** | Indicates an identifier according to Table 4.xxx that can be used to map metadata (e.g., from the inband\_metadata syntax element) to the related object. |
| **num\_audio\_objects\_minus\_one** | Plus 1 indicates the number of audio objects. |
| **num\_target\_devices** | Indicates the number of target devices. |
| **b\_enable\_device\_addressing** | Indicates if device addressing is enabled. |
| **b\_address\_to\_all\_devices** | Indicates if the the block stream with the block\_id=i is addressed to all devices. |
| **device\_address\_mask** | Indicated the device address mask for the respective block\_id. Bit 0 of this mask refers to the least significant bit of that mask and is associated with the device\_id=’0’, bit 1 with device\_id=’1’, and other bits respectively. |
| **max\_interleaving\_depth** | Indicates the maximum interleaving depth of redundant blocks in the stream of blocks. |
| **b\_default\_signal\_connected** | Indicates if a signal is connected and a role is assigned on a specific device.  A value of ‘0’ indicated that the audio signal on default\_audio\_signal\_bus\_offs[d][s] is not connected on the device d. Its role shall be set to AUDIO\_SIGNAL\_NO\_ROLE.  A value of ‘1’ indicates that the audio signal on default\_audio\_signal\_bus\_offs[d][s] is connected on the device d. Its role shall be transmitted explicitly. |
| **default\_audio\_signal\_role** | Indicates the default role of the audio signal according to Table 4.ASR. |
| **b\_set\_default\_custom\_gain** | Indicates if a default custom gain is signaled. |
| **gain\_code** | See 4.5.2.18.1. |
| **b\_set\_default\_custom\_delay** | Indicates if a default custom delay is signaled. |
| **delay\_code** | See 4.5.2.18.1. |
| **map\_or\_mix** | Indicates if a source channel indicated by a bit in the respective channel\_config\_set\_channel\_mask that is not present in the channel\_mask\_superset is mapped at another channel in the superset (0) or upmixed into two channels of the superset. |
| **is\_channel\_pair** | Indicates if the following mapping or mixing parameters are applied to a single channel (0) or to two consecutive channels (1). |
| **upmix\_target\_ch\_idx** | Indicates the channel offset in the superset the source channel has been mapped or mixed into before encoding. This corresponds to the channel offset on the audio bus. |
| **upmix\_gain** | Indicates the gain that has been applied to the source signal before mapping or mixing it into the target channel of the superset according to Table 4.MIXGAIN. |
| **Table 4.MIXGAIN – Values of the upmix\_gain field**   |  |  | | --- | --- | | Value of upmix\_gain | Mixing gain | | 0 | 1.0 (0 dB) | | 1 | (~-1.5 dB) | | 2 | (~-3 dB) | | 3 | (~-4.5 dB) | | 4 | (~-6 dB) | | 5 | (~-9 dB) | | 6 | (~-12 dB) | | 7 | reserved | | |
|  |  |

* + - 1. **Decoding process**
         1. **Variables**

|  |  |
| --- | --- |
| block\_bytes\_left | Indicates the number of bytes remaining for the current block, excluding **block\_crc (**if **b\_block\_protected** == 1). When used in the syntax, it is assumed that this value is always up to date. |
| metadata\_section\_type | An array populated with the metadata types transmitted in the current block. Metadata types are defined in Table 4.xxx – Values of the inband\_metadata\_type field |
| num\_metadata\_sections | Indicates the number of metadata sections transmitted in the current block (corresponds to the size of the array metadata\_section\_type). |
| n\_metadata\_bits | Indicates the number of bits of one metadata section. The size is either transmitted in the bitstream (**b\_md\_size\_signaled** == 1) or derived from block\_bytes\_left (i.e., until the end of the block). |
| active\_channel\_mode | Indicates the currently active channel mode subset. |
| active\_channel\_mask | Indicates the currently active channel mask subset. |
|  |  |

* + - * 1. **Decoding of iif\_specific\_config**

As an extension to the abstract class AudioSpecificConfig(), the iif\_specific\_config() carries IIF specific info enabling signaling for specific use cases, e.g., broadcasting. It provides information such as signals to devices mapping enabling a particular device to skip over irrelevant parts and only decode relevant IIF Blocks.

Besides the static parameters such as the IIF Frame length and the intended output sampling rate, iif\_specific\_config() also carries information on the IIF Audio Bus Source Info and Target Info. The signaled frame length and sampling rate shall be used to configure the underlying tools of the low delay codec. The IIF Audio Bus Source Info provides the mapping of an IIF Block containing audio signals into the IIF Audio Bus and the Target Info provides the mapping of IIF Audio Bus into a pre-defined audio presentation layout, i.e., channels, objects, or HOA, and additionally, the routing to a list of target devices.

If required, gain parameters shall be retrieved and applied for the two Target Info functionalities, i.e., upmix\_gain in channel\_subset\_mix\_info() and gain\_code in bus\_to\_device\_mapping\_defaults(), for the pre-defined audio presentation layout mapping and device routing, respectively. Furthermore, if required, other parameters in bus\_to\_device\_mapping\_defaults(), such as the audio signal roles configuration (e.g., for the echo reference signaling) and delay parameters shall additionally be retrieved to handle a dynamic interaction between the listener and devices position.

Finally, an IIF decoder shall decode information on the latency (max\_interleaving\_depth) and the number of config updates (n\_config\_updates) with their respective indices (dynamic\_config\_upd\_idx > 0), included in iif\_specific\_config(). Note that iif\_specific\_config() specifies the configuration for dynamic\_config\_upd\_idx = 0. A configuration update shall be signaled via iif\_config\_update(dynamic\_config\_upd\_idx), for other out of band configuration indices (dynamic\_config\_upd\_idx > 0).

Some parameters specified in iif\_specific\_config() can be dynamically updated through the iif\_config\_update(dynamic\_config\_upd\_idx = 0) inband signaling mechanism (per block) or through the decoding of a specific out of band configuration index (dynamic\_config\_upd\_idx > 0), as specified in the iif\_block\_payload().

Configuration updates shall apply to the current frame only, i.e., updates shall not persist between frames. For each frame, the decoder shall reset to the initial parameters transmitted in iif\_specific\_config() before applying any configuration updates.

For AUDIO\_PRESENTATION\_TYPE\_CHANNELS, both channel superset and any channel subset may be selected during runtime. If an output interface is available, the decoder shall report the channel configuration change via the interface.

If the audio signal role for device d is updated (b\_override\_default\_role), the renderer shall adapt the signal routing from the IIF Audio Bus to the device according to the flag b\_signal\_connected; if set (1), the signal shall be routed to device d, otherwise the signal shall not be routed to device d. If available, the decoder shall report the role change for device d on the output interface.

Gain updates (b\_override\_default\_gain) shall be applied in the decoder based on the value iif\_apply\_gains. If the input interface is not available, gains shall be applied in the decoder. ~~The decoder shall apply linear interpolation to reach the target gain within half the current frame size.~~ If available, the decoder shall report gain updates on the output interface.

Delay updates (b\_override\_default\_delay) shall be applied in the decoder based on the value iif\_apply\_delay. If the input interface is not available, delays shall always be applied in the decoder. If available, the decoder shall report delay updates on the output interface.

* + - * 1. **Decoding of IIF Block**

At least one IIF Block shall be decoded from an iif\_access\_unit provided that the block is addressed to the decoder. Considering the latency info specified by max\_interleaving\_depth, the presence of redundancy blocks to cope with packet loss shall be verified. If no packet loss is detected, the extracted blocks shall be arranged based on the extracted information such as frame\_counter and block\_id. In the presence of multiple blocks having different block priority values, the decoder shall select the block with the lowest block priority.

An IIF block is divided into three parts, i.e., header, payload and footer. The header shall decode information such as frame\_counter (indicating the corresponding audio frame of the block), block\_id, block\_priority, block\_size, b\_block\_protected (indicating the presence of CRC stored in the footer), reserved and block size extension (indicating a block\_size greater than 2047 bytes).

The iif\_block\_payload shall then decode information regarding the config update, audio data and metadata. When an inband config update is triggered (b\_config\_upidate\_inband = 1), the dynamic parameters such as the audio signal roles configuration, gains and delays, can be updated. Additionally, a relative delay (rel\_delay\_code) may be used to signal the delay relative to the default delay.

Next, iif\_block\_payload shall verify whether an audio payload is present for the specified block\_id or not, and act accordingly. An audio payload consists of one or more channel elements of type Single Channel Element (SCE) or Channel Pair Element (CPE).

* + - * 1. The last part of iif\_block\_payload decoding is to retrieve information on the presence of inband metadata, which can either be signaled explicitly or not. If present, depending on the amount of metadata sections carried in the payload, it is then verified whether the corresponding metadata size is derived from the remaining payload bytes or specified in the payload (md\_size()). The inband\_metadata shall then specify the content based on the specified metadata type, such as INBAND\_METADATA\_TYPE\_AUTH for media authenticity metadata. In case the inband\_metadata\_type is “Reserved for use outside of ISO scope”, the IIF decoder shall convey the binary data to the calling layer.**Decoder interfaces**

**Table 4.XXY — Immersive interchange format decoder interface**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| iif\_decoder\_interface() |  |  |
| { |  |  |
| if (**iif\_input\_interface\_present**) | **1** | **uimsbf** |
| if (**iif\_device\_id\_present**) { | **1** | **uimsbf** |
| **iif\_device\_id**; | **8** | **uimsbf** |
| } |  |  |
| if (**iif\_gain\_control\_present**) { | **1** | **uimsbf** |
| **iif\_device\_id**; | **8** | **uimsbf** |
| **iif\_signal\_index**; | **8** | **uimsbf** |
| **iif\_apply\_gains**; | **1** | **uimsbf** |
| } |  |  |
| if (**iif\_delay\_control\_present**) { | **1** | **uimsbf** |
| **iif\_device\_id**; | **8** | **uimsbf** |
| **iif\_signal\_index**; | **8** | **uimsbf** |
| **iif\_apply\_delay**; | **1** | **uimsbf** |
| } |  |  |
| } else if (**iif\_output\_interface\_present**) { | **1** | **uimsbf** |
| if(**iif\_channel\_config\_changed**) { | **1** | **uimsbf** |
| **iif\_channel\_config\_mask**; | **32** | **uimsbf** |
| } |  |  |
| if(**iif\_audio\_signal\_role\_changed**) { | **1** | **uimsbf** |
| for(s = 0; s < **iif\_num\_signals\_device\_routing**; ++s) { | **8** | **uimsbf** |
| if(**iif\_audio\_signal\_role\_present**) { | **1** | **uimsbf** |
| **iif\_audio\_signal\_role;** | **8** | **uimsbf** |
| } |  |  |
| } |  |  |
| } |  |  |
| if(**iif\_gain\_changed**) { | **1** | **uimsbf** |
| for(s = 0; s < **iif\_num\_signals\_device\_routing**; ++s) { | **8** | **uimsbf** |
| if(**iif\_gain\_present**) { | **1** | **uimsbf** |
| **iif\_gain\_code;** | **8** | **uimsbf** |
| } |  |  |
| } |  |  |
| } |  |  |
| if(**iif\_delay\_changed**) { | **1** | **uimsbf** |
| for(s = 0; s < **iif\_num\_signals\_device\_routing**; ++s) { | **8** | **uimsbf** |
| if(**iif\_delay\_present**) { | **1** | **uimsbf** |
| **iif\_delay\_code;** | **8** | **uimsbf** |
| } |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| } |  |  |

* + - * 1. **Semantics of iif\_decoder\_interface()**

|  |  |
| --- | --- |
| **iif\_input\_interface\_present** | Indicates whether the input interface is selected. |
| **iif\_device\_id\_present** | Indicates whether a device ID is present. |
| **iif\_device\_id** | The IIF device ID, 0.. **num\_target\_devices**-1. |
| **iif\_gain\_control\_present** | Indicates whether gain control information is present. |
| **iif\_signal\_index** | The index of the signal under device routing. |
| **iif\_apply\_gains** | Gains shall be applied in the decoder if set (1), else gains shall not be applied in the decoder. |
| **iif\_output\_interface\_present** | Indicates whether the output interface is selected. |
| **iif\_channel\_config\_changed** | Indicates whether the channel configuration has changed. |
| **iif\_channel\_config\_mask** | The channel configuration mask, see **channel\_config\_set\_channel\_mode** |
| **iif\_audio\_signal\_role\_changed** | Indicates whether an audio signal role has changed. |
| **iif\_audio\_signal\_role** | The audio signal role, according to Table 4.ASR – Values of the audio\_signal\_role field. |
| **iif\_num\_signals\_device\_routing** | The number of signals assigned to device routing. |
| **iif\_gain\_changed** | Indicates whether an audio signal gain has changed. |
| **iif\_gain\_present** | Indicates whether an audio signal gain is present. |
| **iif\_gain\_code** | The gain code, see **gain\_code**. |
| **iif\_delay\_changed** | Indicates whether an audio signal delay has changed. |
| **iif\_delay\_present** | Indicates whether an audio signal delay is present. |
| **iif\_delay\_code** | The delay code, see **delay\_code**. |