MXF : DOLBY ATMOS : LE SON SPATIALISÉ

Ce chapitre est toujours en cours d'écriture. Certains passages peuvent manquer d'informations.
Notez également qu'il est (finalement) très orienté sur la norme Immersive Audio Bitstream au lieu de la norme Dolby Atmos Bitstream. Ce chapitre sera probablement renommé Immersive Audio Bitstream. Atmos utilisant cette norme, cela ne changera pas à la compréhension finale d'un MXF Dolby Atmos.

Références	SMPTE 429-18 - Immersive Audio Track File (description générale, avec les KLV métadonnées descriptors) SMPTE 429-19 - DCP Operational Constraints for Immersive Audio SMPTE 2098-1 - Immersive Audio Metadata SMPTE 2098-2 - Immersive Audio Bitstream Specification SMPTE 2098-5 - D-Cinema Immersive Audio Channels and Soundfield Groups SMPTE RDD 57:2021 - Immersive Audio Bitstream and Packaging Constraints - IAB Application Profile 1 SMPTE RDD 29:2019 - Dolby Atmos Bitstream Specification Dolby Atmos IMF IAB Interoperability Guidelines (hors scope DCP) ISDCF Doc 15: SMPTE ST 2098-2 Immersive Audio Bitstream Constraints - IAB Application Profile 1
Modèle KLV	Pour les **KLV headers** : Local Sets Pour les **KLV body** : Data Items
Universal Label	**Essence:** `060e2b34.01020105.0e090601.00000001` - Immersive Audio Data Element **Metadata Header :** `060e2b34.02530105.0e090603.00000000` - Immersive Audio Data Essence Descriptor `060e2b34.02530105.0e090606.00000000` - Immersive Audio Data Essence Sub-Descriptor **Identifiant Item** : `060e2b34.04010105.0e090605.00000000` - Immersive Audio Data Essence Container `060e2b34.04010105.0e090604.00000000` - Immersive Audio Data Essence Coding

PRÉFACE

La technologie **Dolby Atmos** va utiliser la norme SMPTE **Immersive Audio** (SMPTE 2098-2) comme base commune et normer sa structure de données et ses règles avec la norme **Dolby Atmos Bitstream** (SMPTE RDD 29:2019) pour former la technologie Dolby Atmos dans un DCP compatible IAB.

Pour être très clair, la norme **Dolby Atmos Bitstream** ne va pas ajouter des éléments supplémentaires par rapport à la norme **Immersive Audio Bitstream** (sinon on sortirait de la norme IAB). La norme Atmos va juste figer la norme IAB pour ses propres besoins et avec ses propres paramètres. L'IAB étant plus large, Atmos va juste restreindre certains paramètres ou données à sa propre utilisation. C'est pour cela qu'étudier le **Dolby Atmos Bitstream** (plus restreint) permet de comprendre assez rapidement l'**Immersive Audio Bitstream** (plus vaste) vu qu'il ne va utiliser une petite partie de la norme IAB.

Voyez cela comme un de ces énormes hangars de stockage qu'on peut louer : c'est structuré, il y a des règles et n'importe qui peut accéder quand il veut. C'est votre IAB. Vous allez stocker vos objets dans une partie du hangar en respectant les règles du hangar et vous ne pourrez pas imposer vos propres règles (par exemple utiliser le toit pour y faire du stockage), vous n'utiliserez peut-être pas l'autre côté du hangar, ses bureaux ou les toilettes visiteurs : c'est cela la norme Atmos, c'est de dire "Mes objets sont dans le coin gauche du hangar, et je n'utilise pas les toilettes" :)

A l'aide de la norme IAB, chaque constructeur peut faire son propre système **Immersive Audio** personnel : il suffit de respecter les règles de la norme IAB et tout ira bien.

Quelles sont les principales différences entre IAB et Atmos ?

La norme **Dolby Atmos Bitstream** va utiliser la norme **Immersive Audio Bistream**, elle va définir ses paramètres et ses valeurs dans les structures définies par la norme Immersive Audio Bitstream.

Nous allons donc avoir, dans la norme Atmos, beaucoup d'éléments dans les structures de données avec des valeurs à 0 et simplement nommées "*Reserved*" sans plus d'informations : ce sont des sous-structures qui ne seront pas utilisées par Atmos. Pour savoir ce qu'il se cache derrière (par curiosité), il faut donc se reporter à la norme IAB pour comprendre ce qu'il se cache derrière.

Par exemple, dans la norme Atmos, au début de la structure **BedDefinition**, nous aurons un simple et énigmatique *Reserved* avec une valeur nulle :

```
MetaID ..... Plex(8)
Reserved (set to 0x0) ..... 1 bit
ChannelCount ..... Plex(4)
(etc...)
```

Selon la norme Atmos, il faudra donc définir `0x0` dans notre jeu de données, quoiqu'il arrive.

Mais si vous voulez comprendre ce qu'il se cache derrière ce *Reserved* et pourquoi il faut une valeur nulle, il suffit de jeter un coup d'oeil à la norme IAB et la définition de la structure **BedDefinition** :

```
    MetaID ..... Plex(8)
    ConditionalBed ..... 1 bit
    if (ConditionalBed == 0x1) {
        BedUseCase ..... 8 bits
}
ChannelCount ..... Plex(4)
(etc...)
```

Nous comprenons que notre `0x0` est un booléen. Si nous définissons une valeur nulle, c'est pour ne pas utiliser la sous-structure **BedUseCase**, il ne sera pas utilisé par Atmos.

L'autre différence majeure entre les deux normes se situe dans le **renommage de certains termes**. Par exemple, IAFrame est nommée ATMOSFrame, Version en AtmosVersion, le nommages des SubBlocks, etc...

Pour des raisons de compréhension, j'essayerai d'utiliser que les termes venant de la norme IAB et non celle de la norme Atmos, car si vous travaillez sur d'autres IAB, ils n'utiliseront pas les termes Atmos

Également, pour des raisons de simplicité, je vais devoir légèrement jouer avec certains termes pour ne pas vous perdre. Les plus experts tiqueront probablement mais c'est dans un but pédagogique :-)

LES MÉTADONNÉES

L'intégration des pistes Immersive Audio s'accompagne de deux KLV dans l'entête :

Immersive Audio Data Essence Descriptor et **Immersive Audio Data Essence Sub Descriptor** :

```
060e2b34.02530105.0e090603.00000000   |   Immersive Audio Data Essence Descriptor
╔═══════════════════════════════════════╦═══════════════════════════════════════════════════════════════════════════════════════
║   3C0A - Instance UID                 ║  179c9769.caef4770.958fbc57.406f4bf9
║   FFFF - Descriptors & Sub-Descriptors (SMPTE)  ║  1 item(s): 2f13c88b.3498498f.89a150ad.fd2c94fd
║   3006 - Linked Track ID              ║  2
║   3001 - Sample Rate                  ║  24/1
║   3002 - Container Duration           ║  1440
║   3004 - Essence Container            ║  060e2b34.04010105.0e090605.00000000  (Immersive Audio Data Essence Container)
║   3E01 - Immersive Audio              ║  060e2b34.04010105.0e090604.00000000
╚═══════════════════════════════════════╩═══════════════════════════════════════════════════════════════════════════════════════
060e2b34.02530105.0e090606.00000000   |   Immersive Audio Data Essence Sub Descriptor
╔═══════════════════════════════════════╦═══════════════════════════════════════════════════════════════════════════════════════
║   3C0A - Instance UID                 ║  2f13c88b.3498498f.89a150ad.fd2c94fd
║   FFFE - Immersive Audio - Immersive Audio ID  ║  d24917c83bbe4018aeb061af72a64327
║   FFFD - Immersive Audio - First Frame  ║  00000000
║   FFFC - Immersive Audio - Max Channel Count  ║  10
║   FFFB - Immersive Audio - Max Object Count  ║  118
║   FFFA - Dolby Atmos Sub-Descriptor - Atmos Version  ║  01
╚═══════════════════════════════════════╩═══════════════════════════════════════════════════════════════════════════════════════
```

Comme pour d'autres descripteurs et sous-descripteurs, le descripteur va faire un lien avec le sous-descriptor. Ainsi, dans **Immersive Audio Data Essence Descriptor**, l'item **Descriptors & Sub-Descriptors** va faire lien avec le sous-descripteurs **Immersive Audio Data Essence Sub-Descriptor**.

Il faut savoir que la quasi-totalité des items - comme Immersive Audio Version, Max Channel Count, Max Object Count, Immersive Audio ID, First Frame et IAB Sample Rate - sont totalement optionnels, pire la norme **SMPTE 429 - DCP - Immersive Audio Track File** indique que la plupart doivent être ignorés par le décodeur et d'autres sont obsolètes.

Voici quelques descriptions sur certains items :

- **Max Channel Count** (optionnel) définit le nombre maximum de channels dans le bitsream (nous aurons toujours 10 pour Atmos)
- **Max Object Count** (optionnel) définit le nombre maximum d'objets dans le bitstream (nous aurons toujours 118 pour Atmos)
- **Immersive Audio ID** (optionnel) est juste un UUID aléatoire pour définir le projet audio.
- **First Frame** (optionnel) va être utilisé pour s'aligner avec la première frame appelée "First Frame of Action" [1] ou "FFOA". En règle général, vous aurez soit 0, soit 24, soit 48 (pour du 48 fps) ou soit 192, soit 384 (pour du 48 fps). 192 et 384 correspondent à un FFOA à la position 8 secondes en début du film.
- **IAB Sample Rate** (optionnel) définit simplement le SampleRate des essences.

LES DONNÉES

Dans Partition Body, nous avons un ou plusieurs **Immersive Audio Data Element** qui intègreront chacune une petite partie des données et des métadonnées audios :

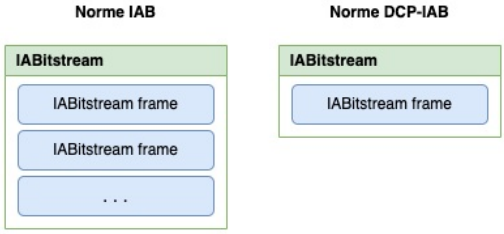
```
(...)
060e2b34.01020105.0e090601.00000001   |   Immersive Audio Data Element  (element n°1)
060e2b34.01020105.0e090601.00000001   |   Immersive Audio Data Element  (element n°2)
060e2b34.01020105.0e090601.00000001   |   Immersive Audio Data Element  (element n°3)
060e2b34.01020105.0e090601.00000001   |   Immersive Audio Data Element  (element n°x)
(...)
```

Dans un **Immersive Audio Data Element**, vous aurez une (seule) structure appelée **IABistream**.

Une **IABitstream** correspond à une unité d'image (`image edit unit`). Par exemple, si votre framerate est de 24 images par seconde, alors une **IABitstream** va contenir 1/24eme d'audio.

A L'INTÉRIEUR DU IABITSTREAM : LE CONTAINER MAÎTRE DE BASE

Un **IABitstream** va contenir **une ou plusieurs IABistream frames** :



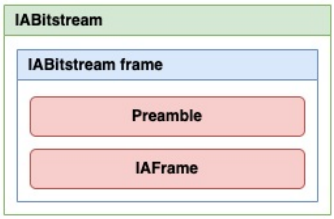
La norme IAB permet d'insérer plusieurs **IABistream frame** dans un seul container **IABitstream** -cependant- la norme DCP sur l'IAB SMPTE-429-19 impose **une seule IABistream frame** par container **IABitstream**.

Nous n'aurons donc qu'une seule **IABitstream frame** par KLV.

> **IABitstream et IABitstream frame**
>
> Dans la suite du document, je ne parlerai plus du tout du container IABistream car cela n'a aucun intérêt, ce n'est qu'un concept pour indiquer qu'il va contenir plusieurs IABitstream frames. Vu que nous n'en avons qu'une seule, IABitstream n'apporte rien de plus à la compréhension.

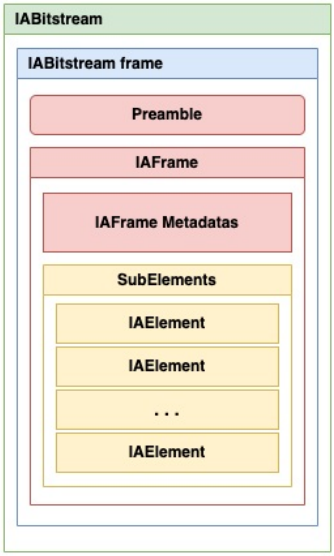
A L'INTÉRIEUR D'UNE IABITSTREAM FRAME : L'UNITÉ DE BASE

La structure d'une **IABitstrea frame** possèdera deux principaux éléments : un Preamble et une IAFrame :



Le Preamble est un ... préambule de données. La norme IAB indique simplement qu'elle stocke un payload de données - et que c'est en dehors de son scope, nous pouvons donc passer pour l'instant.

L'IAFrame est importante car c'est notre container principal qui va contenir une quantité de données, si nous déroulons IAFrame, nous avons plusieurs éléments intégrés :



En tout début, nous avons les métadonnées IAFrame (*IAFrame Metadata*) qui va intégrer plusieurs items comme Version, SampleRate, BitDepth, FrameRate, MaxRendered (voire paragraphe **IA Frame** pour avoir la structure)

Ce qui va être déterminant, ce sont les IAElement dans SubElements qui vont intégrer les données de spatialisations et surtout les données brutes audios.

LES IAELEMENTS : LES BLOCS DE MÉTADONNÉES ET DE DONNÉES UTILES

En fin de l'IAFrame, nous avons le bloc **SubElements** qui va intégrer plusieurs éléments appelés **IAElements**.

Un **IAElement** est une simple structure de données et, suivant le type, la structure sera différente.

Voici la liste complète de tous les IAElement normés par la norme **Immersive Audio Bistream** :

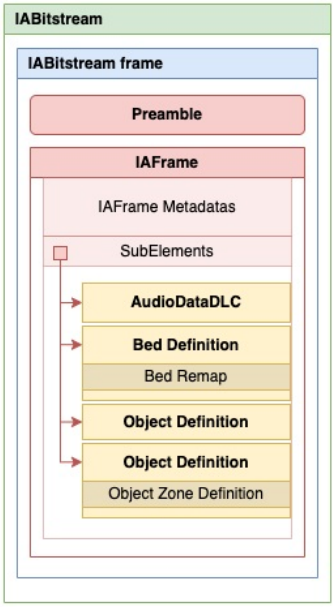
- **IA Frame**
  - Bed Definition
    - Bed Remap
  - Object Definition
    - Object Zone Definition
  - Audio Data DLC
  - Audio Data PCM
  - Authoring Tool Info
  - User Data

En vert, les seuls IAElements acceptés par l'IAB-DCP [2].

Oui, une **IA Frame** - que nous avons vu en tout premier - est aussi un IAElement ! À l'exception qu'il est le parent de tous les IAElements et ne peut être inclus dans les sous-éléments.

Le tout premier IAElement sera donc notre IAFrame et pourra avoir des sous-éléments IAElements. Mais ces mêmes sous-éléments IAElements peuvent aussi avoir des sous-éléments IAElements : c'est une cascade d'IAElements.

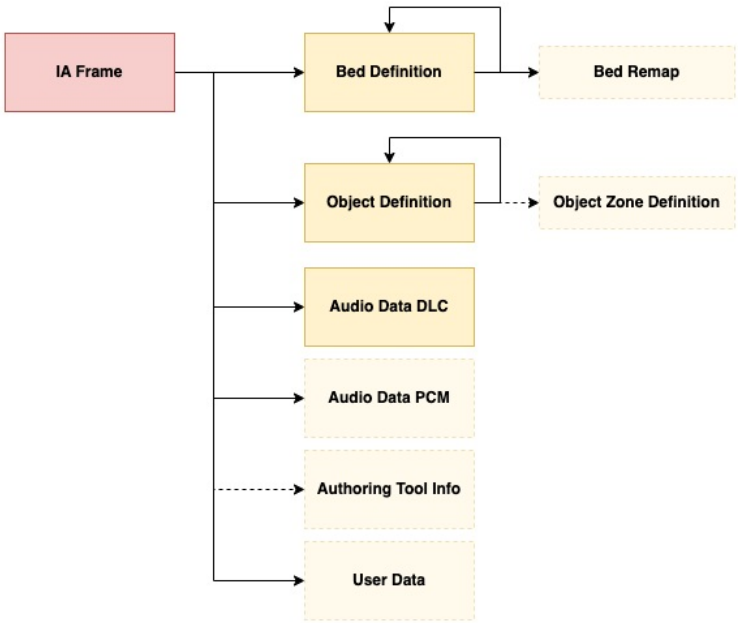
Un tout petit exemple avec seulement quelques éléments et une récursivité réduite :



Il existe une restriction sur les sous-éléments possibles :

- Une IAFrame peut intégrer n'importe quels autres IAElements.
- Les autres IAElements ne peuvent pas intégrer une IAFrame.
- Certains IAElements n'ont tout simple pas de bloc SubElements, ils ne pourront donc jamais intégrer des sous-éléments.

Il existe une hiérarchie dans les IAElements, voici un schéma montrant les liens entre sous-éléments :



Ainsi, en suivant le schéma, la norme IAB permet :

- **IA Frame** est le parent et peut avoir plusieurs sous-éléments **Bed Definition**, **Object Definition**, **Audio Data DLC**, **Audio Data PCM** et **User Data** (à l'exception d'**Authoring Tool Info** qui sera toujours unique ou absent)
- **Bed Definition** peut avoir plusieurs **Bed Remap** mais également d'autres **Bed Definition** comme sous-éléments.
- **Object Definition** peut également avoir plusieurs **Object Definition** mais seulement un seul **Object Zone Definition**.
- Les cases avec des flèches avec des pointillés doivent être uniques.
- Les cases opaques sont les éléments écartés de la norme DCP et ne seront donc jamais utilisés dans un MXF DCP.

Maintenant que nous avons vu cet aspect hiérarchique, voyons à l'intérieur de chaque élément.

LES DIFFÉRENTS ÉLÉMENTS : LEURS STRUCTURES INTERNES

IABITSTREAM FRAME

Voici enfin l'intérieur d'une **IABitstream frame** avec une **IAFrame** complète :

`<Element>` spécifie des bits qu'on va devoir lire. Ces bits stockent des valeurs pouvant avoir un impact sur le reste de la structure. Les autres éléments (en gris) sont de simples indications indiquant des blocs conceptuels.

Nous avons vu rapidement qu'une **IABitstream frame** comprend un **Preamble** et une **IAFrame**, mais dans les faits, c'est un peu plus compliqué :-)

Au tout début, vous avez un **Preamble** (que j'ai renommé **Preamble Header** dans le schéma des raisons de compréhension). Lui-même respecte le principe du KLV (identifiant + taille + valeur).

Puis, c'est là qu'arrive la subtilité car dans les précédents schémas, je n'ai pas indiqué ce header afin d'éviter une complexité malvenue.

L'**IAFrame** indiquée dans les précédents schémas n'intégraient pas ce petit header car le véritable IAFrame débute juste après et c'est lui qui est un **IAElement** comme les autres.

Dans le **IAFrame Header**, vous aurez aussi un identifiant puis la taille complète de la partie **IAFrame**. Enfin, notre **IAFrame** qui est notre véritable IAElement.

Et c'est à partir de là que la magie opère.

Comment analyser une structure IAElement ?

Contrairement aux autres chapitres d'analyses de binaire, notamment ceux sur les KLV, la structure d'un IAElement est un peu plus complexe. Des blocs de données peuvent être absents ou présents suivant certaines conditions et notamment suivant les valeurs de certains items.

Les structures doivent être lues comme du pseudo-code.

Les **items** seront exprimés avec du <jaune> et doivent être lus bits à bits.

Les **mots-clefs** sont des concepts et ne doivent pas pris pour des données mais pour des instructions, et sont exprimés avec du violet :

- foreach : lire pour chaque élément dans l'item ou - dans le cadre d'un nombre comme valeur - une simple boucle de 0 à la valeur dans l'item
- while : lire en boucle et/ou jusqu'à ce que l'expression accolée soit valide
- if : si et seulement si l'item valide l'expression accolée.
- Les noms ne commençant pas par un mot-clef (foreach, while, if, ...) et finissant avec des accolades ne sont que des indications d'un nouveau bloc de structure. Ils ne sont pas réellement dans les données. Considérez-les comme une aide visuelle pour comprendre les différentes structures qui s'imbriquent.

Si vous avez des notions en programmation, vous serez comme un poisson dans l'eau :-)

Autre élément important, les pseudo-codes ont été (très) simplifiés afin d'aider à la lecture et à la compréhension. Normalement, tout est présent mais il peut arriver que des subtilités soient un peu trop lisées avec le pseudo-code. Si vous cherchez à écrire un parser ou un encoder d'IAB, je vous conseille tout de même de vous reporter à la norme IAB, à la fin de la lecture de ce chapitre afin d'affiner votre compréhension.

IAELEMENT

Tous les **IAElements** respectent une structure commune qui ressemble à un KLV.

La structure d'un **IAElement** intègre toujours ces items dans cet ordre :

```
IAElement {                              # --Taille:--
   <ElementID>                           #   Plex(8)
   <ElementSize>                         #   Plex(8)
   <ElementData>                         #   Variable (ElementSize)
}
```

En résumé :

- **ElementID** est un identifiant qui va définir le type de l'IAElement
- **ElementSize** définit la taille de **ElementData**
- **ElementData** stocke une structure spécifique au type de l'IAElement (via son **ElementID**)

Voici les différents codes d'identification **ElementID** par type d'**IAElement**, ainsi que leurs noms normés entre IAB et Atmos :

ElementID	Nom IAB	Nom Atmos	Description
`0x08`	`IA_FRAME`	`ATMOS_FRAME`	Frame informations
`0x10`	`BED_DEFINITION`	`BED_DEFINITION`	Bed Definition (Type 1)
`0x20`	`BED_REMAP`	-	Bed Remap
`0x40`	`OBJECT_DEFINITION`	`OBJECT_DEFINITION`	Object Definition (Type 1)
`0x80`	`OBJECT_ZONE_DEFINITION`	-	Extended Object Zone Definition
`0x100`	`AUTHORING_TOOL_INFO`	-	Authoring Tool Information
`0x101`	`USER_DATA`	-	User Defined Data
`0x200`	`AUDIO_DATA_DLC`	`AUDIO_DATA_DLC`	Audio Data DLC (données audios encodées)
`0x400`	`AUDIO_DATA_PCM`	-	Audio Data PCM (données audios brutes)

Nous constatons assez rapidement que Atmos - qui respecte donc la norme DCP-IAB - ne va utiliser que les 4 types de containeurs sur l'ensemble des containeurs IAB possibles (et au passage en profite pour renommer quelques noms et variables ;-)

Si nous prenons un peu de point de vue, voici comment s'articule un **IAElement** avec une structure **ElementData** complète (le tout dans une **IABistream frame**) :

```
IABitstream {
      IABitstreamFrame {
            <PreambleTag>  ------------------------------------.
            <PreambleLength>                                   |
            <PreambleValue>                                    |
            <IAFrameTag>                                       |
            <IAFrameLength>                                    |
             IAFrame (IAElement) {                             |
                  <ElementID>    ------------------------.     |
                  <ElementSize>                         | I   |
                   ElementData {      ------------------+ A   |
                        <Version>                       |     |
                        <SampleRate>                    | E   |___ IABitstream frame
                        <BitDepth>                      | l   |
                        <FrameRate>                     | e   |
                        <MaxRendered>                   | m   |
                        <AlignBits>                     | e   |
                        <SubElementCount>               | n   |
                         SubElements {                  | t   |
                              IAElement {}              |     |
                         }                              |     |
                  }     -------------------------------+     |
            } ------------------------------------------------'
      }
}
```

Commençons l'analyse de l'IAElement IAFrame.

IA FRAME

IAFrame est l'élément de base d'un **IABistream frame**, comme nous l'avons vu, il se placera juste en dessous du Preamble et du **IAFrame Header** et après notre couple **ElementID** (qui sera à `0x08`) et un **ElementSize** correctement renseigné, notre structure **ElementData** sera de la sorte :

```
ElementData "IAFrame" {                             # --Taille:--
    <Version>                                       #   8 bits
    <SampleRate>                                    #   2 bits
    <BitDepth>                                      #   2 bits
    <FrameRate>                                     #   4 bits
    <MaxRendered>                                   #   Plex(8)
    <AlignBits>                                     #   Variable
    <SubElementCount>                               #   Plex(8)
    foreach SubElementCount {
        <IAElement>                                 #   Variable
    }
}
```

Rien de bien extravagant, il suffit de lire, par bloc de bits, les différents <items>.

La seule subtilité se trouvant dans la partie foreach. On va lire les différents IAElements selon le nombre qu'on trouvera dans **SubElementCount**. Ainsi, si notre **SubElementCount** est à 0, nous passerons jamais dans notre foreach. Dans l'autre cas, on va lire et analyser chaque **IAElement** suivant le nombre **SubElementCount**. Par exemple, si **SubElementCount** vaut 10, alors nous savons que nous aurons 10 **IAElements**.

SampleRate est un code sous 2 bits :

Code	Binaire	Description
`0x0`	`00`	48000 samples / seconde
`0x1`	`01`	96000 samples / seconde
`0x2`	`10`	Réservé
`0x3`	`11`	Réservé

BitDepth est un code sous 2 bits :

Code	Binaire	Description
`0x0`	`00`	16 bits
`0x1`	`01`	24 bits
`0x2`	`10`	Réservé
`0x3`	`11`	Réservé

Dans notre cas, nous serons toujours en 24 bits.

FrameRate est un code sous 4 bits :

Code	Binaire	Description
`0x0`	`0000`	24 fps
`0x1`	`0001`	25 fps
`0x2`	`0010`	30 fps
`0x3`	`0011`	48 fps
`0x4`	`0100`	50 fps
`0x5`	`0101`	60 fps
`0x6`	`0110`	96 fps
`0x7`	`0111`	100 fps
`0x8`	`1000`	120 fps
`0x9`	`1001`	Réservé
...		...
`0xF`	`1111`	Réservé

AlignBits seront des bits de bourrage - tous à zéro - pour aligner sur l'octet. Par exemple, l'item Version va prendre 1 octet (8 bits), les 3 autres items (SampleRate, BitDepth et FrameRate) vont aussi prendre 1 octet. Mais MaxRendered, qui a une taille variable, nous ne pouvons savoir combien de bits il va occuper. S'il prend un 1 octet, alors AlignBits sera inexistant. Si MaxRendered prend - par exemple - 4 bits, alors nous aurons un AlignBits à 4 bits de zéro. Nous aurons le principe de l'AlignBits un peu partout dans les autres IAElements.

Avec cette petite structure simple, passons à des structures plus complexes...

BED DEFINITION : LE CÂBLAGE DES PISTES (BED) AVEC LES ASSETS AUDIOS

Les **Bed Definitions** contiennent les métadonnées et des indicateurs vers les assets audios pour créer une frame pour un seul *Bed*.

Les **Bed Definitions** indiquent quels assets audios doivent être acheminés vers quel sortie(s) sonore(s) physique(s) (un haut-parleur en particulier ou bien plusieurs).

Les **Bed Definitions** pointent vers les éléments **Audio Data DLC** ou **Audio Data PCM** via l'identifiant stocké dans **AudioDataID** - et vers un haut-parleur via l'identifiant stocké dans **ChannelID**.

Les **Bed** sont un concept un particulier mais relativement simple : ce sont juste les pistes de base (ex. L, R, C, LFE, Ls, Rs, ...). Par exemple, dans un système Atmos, vous pouvez avoir un 9.1. Nous aurons donc 10 Bed Channels [3]. Imaginé, les Bed Channels sont comme de longues bandes de papier où dessus se trouve les mixages sonores principaux. Et par dessus, vous pouvez poser des petits objects qui peuvent se déplacer...

Voici la structure d'un IAElement **Bed Definition** sous forme de pseudo-code comprenant if et des while en supplément :

```
ElementData "BedDefinition" {                          # --Taille:--
    <MetaID>                                           #   Plex(8)
    <ConditionalBed>                                   #   1 bit
    if ConditionalBed == 1 {
        <BedUseCase>                                   #   8 bits
    }
    <ChannelCount>                                     #   Plex(4)
    foreach ChannelCount {
        <ChannelID>                                    #   Plex(4)     - Table 19 - ChannelID Codes
        <AudioDataID>                                  #   Plex(8)
        <ChannelGainPrefix>                            #   2 bits
        if ChannelGainPrefix > 1 {
            <ChannelGain>                              #   10 bits
        }
        <ChannelDecorInfoExists>                       #   1 bit
        if ChannelDecorInfoExists == 1 {
            <Reserved>                                 #   4 bits
            <ChannelDecorCoefPrefix>                   #   2 bits
            if ChannelDecorCoefPrefix > 1 {
                <ChannelDecorCoef>                     #   8 bits
            }
        }
    }
    <Reserved>                                         #   10 bits
    <AlignBits>                                        #   Variable
    <AudioDescription>                                 #   8 bits
    if AudioDescription >= 0x80 {
        while {
            <AudioDescriptionText>                     #   8 bits
        }
    }
    <SubElementCount>                                  #   Plex(8)
    foreach SubElementCount {
        <IAElement>                                    #   Variable
    }
}
```

MetaID est un identifiant qui permet de suivre les métadonnées entre chaque IAFrame. Selon les restrictions DCP-IAB, le numéro de **MetaID** ne peut excéder 118.

Les **Conditionals** sont de simples booléns indiquant si on aura un **UseCase** juste derrière. A partir de là, soit vous devrez passer directement à la lecture de <ChannelCount>, soit vous devrez lire <BedUseCase> avant de passer à <ChannelCount>. (Avec Atmos, ConditionalBed sera toujours à 0, donc aucun UseCase)

ChannelCount va indiquer le nombre de channel qui sera à traiter dans la suite. (Avec Atmos, la valeur est fixée à 10).

ChannelID est un identifiant qui va définir le type du channel. Nous aurons un code allant de `0x0` à `0x17`. Par exemple, `0x0` représente le canal LEFT. Reportez-vous au paragraphe **Tables d'informations : Table 19 : Channel ID codes**.

AudioDataID est un identifiant qui permet d'identifier l'audio associé. Si l'identifiant utilisé est 0 ou NULL, alors soit l'audio est absent, soit il est considéré comme silencieux. Vous retrouverez les **AudioDataID** dans l'entête **Audio Data DLC** ou **Audio Data PCM**.

ChannelGainPrefix est un code qui va indiquer le type de gain qu'on va appliquer.

Code	Binaire	Description
`0x0`	`00`	Le gain est à 1.0
`0x1`	`01`	Le gain est à 0.0
`0x2`	`10`	Le gain est défini juste en dessous
`0x3`	`11`	Réservé

Si le code est `0x2`, alors les prochains bits (ChannelGain) vont définir le numéro exact du gain allant de 0.0 à 1.0 avec une méthode de calcul [4]

Dans le bloc **ChannelDecor(relation)**, nous avons toute la partie sur les **valeurs de décorrélations audios**.

N'étant pas spécialiste du principe de décorrélation, je propose une traduction légèrement retravaillée de la norme IAB : *« Lors du rendu d'objets et des canaux, il est possible qu'un seul asset audio soit acheminé ou panoramique vers plusieurs haut-parleurs. Le paramètre de décorrélation indique le degré de décorrélation souhaité entre plusieurs signaux de sortie dérivés d'une seule forme d'onde audio. Le code ChannelDecorCoef doit spécifier le degré de décorrélation à appliquer à l'asset audio du Bed Channel. »*

Son préfixe est un code :

Code	Binaire	Description
0x0	`00`	Aucune décorrelation
0x1	`01`	Decorrelation maximum
0x2	`10`	Décorrelation définie juste en dessous

Si le code est `0x2`, la valeur de décorrélation se trouve dans les 8 prochains bits et son amplitude sera de 0 à 255 (`0x00` - `0xFF`)

(Note: l'item Reserved n'est pas spécifié dans la documentation et les nombreux IAB testés n'ont donné qu'une série de 4 bits à zéro.)

Reserved aura toujours la valeur `0x180` (`0110000000` en binaire). Aucune description du pourquoi dans la norme IAB. Faire du reverse

AudioDescription est un bloc de 8 bits où chaque bit représente une signification particulière. Combiné, vous pouvez avoir une description audio avec plusieurs significations :

Bitmask	Binaire	Signification
`0x01`	`00000001`	Non indiqué
`0x02`	`00000010`	Dialogue
`0x04`	`00000100`	Musique
`0x08`	`00001000`	Effets
`0x10`	`00010000`	Bruitage
`0x20`	`00100000`	Ambiance
`0x40`	`01000000`	Réservé
`0x80`	`10000000`	Texte

Un exemple, si nous avons la valeur `0x54` qui nous donne `01010100` en binaire, avec notre tableau, nous savons que nous avons affaire à de la **Musique**, du **Bruitage**, et **Réservé**.

Si **AudioDescription** est supérieur ou égale à `0x80` (donc que le bit de poids fort, le premier, est à 1), alors nous aurons du texte dans **AudioDescriptionText** qu'il faudra lire en boucle jusqu'à tomber sur un caractère nul (`NULL` / `0x00`). Notez que nous pouvons combiner "*Non indiqué*" et "*Texte*", mais nous ne pouvons pas combiner "*Non indiqué*" avec une autre signification.

Les **SubElements** possibles pour BedDefinitions sont uniquement **BedDefinition** et **BedRemap** - sauf dans le cas d'un DCP-IAB / Atmos, nous aurons aucun sous-élément [5].

Les paramètres et valeurs spécifiques à DCP-IAB / Atmos :

Item	Valeur
ConditionalBed	`0`
ChannelCount	`10`
ChannelGainPrefix	`0x00`
ChannelDecorInfoExists	`0x0`
Reserved (10 bits)	`0x180`
AudioDescription	`0x5`
SubElementCount	`0`

OBJECT DEFINITION : LES MÉTADONNÉES DE SPATIALISATION

Le concept d'objet sonore est différent des principes classiques qu'on a pu voir auparavant. Un objet sonore est un asset sonore mais qui n'a pas de canal en particulier. Il est libre de circuler d'un canal à l'autre. Un objet doit donc posséder des métadonnées pour sa position, sa durée, sa direction, son intensité, etc...

Les **Object Definitions** permettent de définir l'emplacement sonore et sa spacialisation à un certain espace-temps.

Un **ObjectDefinition** va pointer vers un IAElement **Audio Data DLC** ou **Audio Data PCM** (via son AudioDataID) et va permettre de le disposer grâce aux items ObjectPosX, ObjectPosY et ObjectPosZ.

Un **ObjectDefinition** est subdivisé en petit morceau via la variable statique `NUM_PAN_SUB_BLOCKS` qui permet d'avoir une granularité pour les déplacements des objets dans l'espace. `NUM_PAN_SUB_BLOCKS` est dépendant du FrameRate. Selon ce dernier, `NUM_PAN_SUB_BLOCKS` sera soit de 2, de 4 ou de 8.

Selon la documentation, un seul sous-bloc permet d'avoir une granularité entre 4-5 ms et 20 ms [6].
(voir **Tables d'informations : Tableau 23 : Number Of SubBlocks per IAFrame**)

```
ElementData "ObjectDefinition" {                                    # --Taille:--
    <MetaID>                                                        #   Plex(8)
    <AudioDataID>                                                   #   Plex(8)
    <ConditionalObject>                                             #   1 bit
    if ConditionalObject == 1 {
        <Reserved>                                                  #   1 bit (toujours à 1)
        <ObjectUseCase>                                             #   8 bits
    }
    <Reserved>                                                      #   1 bit (toujours à 0)
    foreach NUM_PAN_SUB_BLOCKS {                                    #   Table 23 - Number Of SubBlocks per IAFrame
        if index == 0 {                                             #      (ex. 24fps: NUM_PAN_SUB_BLOCKS=8)
            PanInfoExists = 1
        } else {
            <PanInfoExists>                                         #   1 bit
        }
        if PanInfoExists == 1 {
            <ObjectGainPrefix>                                      #   2 bits
            if ObjectGainPrefix > 1 {
                <ObjectGain>                                        #   10 bits
            }
            <Reserved>                                              #   3 bits
            <ObjectPosX>                                            #   16 bits
            <ObjectPosY>                                            #   16 bits
            <ObjectPosZ>                                            #   16 bits

            <ObjectSnap>                                            #   1 bit
            if ObjectSnap == 1 {
                <ObjectSnapToleranceExists>                         #   1 bit
                if ObjectSnapToleranceExists == 1 {
                    <ObjectSnapTolerance>                           #   12 bits
                }
                <Reserved>                                          #   1 bit (toujours à 0)
            }
            <ObjectZoneControl>                                     #   1 bit
            if ObjectZoneControl == 1 {
                foreach 0...8 {                                     #   Table 24 - Zone Definition Names
                    <ZoneGainPrefix>                                #   2 bits
                    if ZoneGainPrefix > 0x1 {
                        <ZoneGain>                                  #   10 bits
                    }
                }
            }

            <ObjectSpreadMode>                                      #   2 bits
            if ObjectSpreadMode == OBJECT_SPREAD_LOWREZ {           #   Table 26 - Object Spread Mode Code
                <ObjectSpread>                                      #   8 bits
            }
            if ObjectSpreadMode == OBJECT_SPREAD_1D {               #   Table 26 - Object Spread Mode Code
                <ObjectSpread>                                      #   12 bits
            }
            if ObjectSpreadMode == OBJECT_SPREAD_3D {               #   Table 26 - Object Spread Mode Code
                <ObjectSpreadX>                                     #   12 bits
                <ObjectSpreadY>                                     #   12 bits
                <ObjectSpreadZ>                                     #   12 bits
            }
            <Reserved>                                              #   4 bits
            <ObjectDecorCoefPrefix>                                 #   2 bits
            if ObjectDecorCoefPrefix > 1 {
                <ObjectDecordCoef>                                  #   8 bits
            }
        }  #  PanInfoExits
    }  #  foreach

    <AlignBits>                                                     #   Variable
    <AudioDescription>                                              #   8 bits
    if AudioDescription {
        while {
            <AudioDescriptionText>                                  #   8 bits
        }
    }

    <SubElementCount>                                               #   Plex(8)
    foreach SubElementCount {
        <IAElement>                                                 #   Variable
    }
}
```

ObjectPosX, ObjectPosY et ObjectPosZ auront une valeur entre 0.0 et 0.1 qui permet de "positionner" le son dans la salle.

ObjectSnap indique une certaine tolérance à partir du moment où un objet sonore "saute" d'un haut-parleur à un autre. Si ObjectSnap est à 1, alors l'objet sonore doit sauter sur le haut-parleur le plus proche.

ObjectZoneControl permet de définir un gain pour un objet sonore selon certaines zones. Les zones sont des collections de haut-

parleurs dispatchés dans la salle.

> Pourquoi seulement 9 éléments dans le `foreach 0...8` ?
> Car il n'existe que 9 zones, voir Table 24 **Noms des Zone Definition** pour comprendre :-)

ObjectSpread définit la propagation d'un objet sonore. [7]

Les **SubElements** possibles pour ObjectDefinitions sont uniquement ObjectDefinition et ObjectZoneDefinition - sauf dans le cas d'un DCP-IAB / Atmos, nous aurons aucun sous-élément [2b].

Les valeurs obligatoires pour DCP-IAB / Atmos :

Item	Valeurs
ConditionalObject	`0b1`
Reserved	`0b1`
ObjectUseCase	`0xff`
Reserved	`0b0`
ObjectGainPrefix	`0x0`
ObjectSpreadMode	`OBJECT_SPREAD_LOWREZ` ou `OBJECT_SPREAD_1D`
ObjectSnapTolExists	`0x0`
ZoneGain	`0x0` ou `0x1`

Quelques notes :

- Atmos Specifications définit que **ConditionalObject** + **Reserved** + **ObjectUseCase** + **Reserved** est égal à `0x7FE` (sur 11 bits).
- Les contraintes DCP-IAB indiquent que le **ConditionalObject** peut être soit à `0x0` ou `0x1` ; mais s'il est à `0x1`, alors ObjectUseCase doit être à `0xFF`.

BED REMAP : CONFIGURATION POUR DU SOUS-MIXAGE

Ce container est interdit dans un DCP-IAB

Bed Remap permet de définir un autre type de routage suivant le type de système de lecture. Par exemple, si on doit faire du sous-mixage d'un DCP en 9.1 vers un système en 7.1.

```
ElementData "BedRemap" {                          # --Taille:--
    <MetaID>                                      #   Plex(8)
    <RemapUseCase>                                #   8 bits
    <SourceChannels>                              #   Plex(4)
    <DestinationChannels>                         #   Plex(4)
    foreach NUM_PAN_SUB_BLOCKS {                  #   Table 23 - Number Of SubBlocks per IAFrame
        if index == 0 {                           #         (ex. 24fps: NUM_PAN_SUB_BLOCKS=8)
            RemapInfoExists = 1
        } else {
            <RemapInfoExists>                     #   1 bit
        }
        if RemapInfoExists == 1 {
            foreach DestinationChannels {
                <DestinationChannelID>            #   Plex(4)
                foreach SourceChannels {
                    <RemapGainPrefix>             #   2 bits
                    if RemapGainPrefix > 1 {
                        <RemapGain>               #   10 bits
                    }
                }
            }
        }
    }
    <AlignBits>                                   #   Variable
    <Reserved>                                    #   Plex(8)
}
```

OBJECT ZONE DEFINITION : LES ZONES ALTERNATIVES

Ce container est interdit dans un DCP-IAB

Object Zone Definition permet de définir des zones alternatives définites dans la table 28 "**Object Zone Definition Names**".

```
ElementData "ObjectZoneDefinition" {                    # --Taille:--
    foreach NUM_PAN_SUB_BLOCKS {                        #   Table 23 - Number Of SubBlocks per IAFrame
        if index == 0 {                                 #       (ex. 24fps: NUM_PAN_SUB_BLOCKS=8)
            ZoneInfoExists = 1
        } else {
            <ZoneInfoExists>                            #   1 bit
        }
        if ZoneInfoExists == 1 {
            foreach 0...18 {                            #   Table 28 - Object Zone Definition Names
                <ZoneGainPrefix>                        #   2 bits
                if ZoneGainPrefix > 0x1 {
                    <ZoneGain19>                        #   10 bits
                }
            }
        }
    }
    <AlignBits>                                         #   Variable
}
```

AUTHORING TOOL INFO : LES INFORMATIONS CRÉATEURS

Ce container est interdit dans un DCP-IAB

Ce container donne juste quelques informations sur le créateur de cet IAB.

```
ElementData "AuthoringToolInfo" {                       # --Taille:--
    while {
        <AuthoringToolURI>                              #   8 bits
    }
}
```

USER DATA : LES MÉTADONNÉES PERSONNELLES

Ce container est interdit dans un DCP-IAB

Ce container stocke des données utilisateurs non définis et qui seront identifiés via un UL SMPTE.

```
ElementData "UserData" {                                # --Taille:--
    <UserID>                                            #   128 bits
    <UserDataBytes>                                     #   Variable
}
```

AUDIO DATA PCM & DLC : LES CONTAINERS AUDIO

Le **DolbyAtmos** utilise des MXF IAB pour stocker ses données. Il faudra donc se baser sur la doc IAB dans un premier temps pour comprendre le container IAB, puis la doc Atmos pour ses spécificités (des paramètres ou des champs activés ou désactivés)

Les IAB permettent d'utiliser soit des sous-containers **AudioDataPCM** ou **AudioDataDLC** :

- **AudioDataPCM** : containeur stockant de la donnée audio au format PCM (comme le WAV quoi), d'une frame en mono. C'est le containeur le plus simple, les données audio se trouve juste après l'entête contenant un simple identifiant. Ce containeur ne sera jamais utilisé dans un DCP :)
- **AudioDataDLC** : containeur stockant de la donnée audio encodée et compressée sans pertes, une frame en mono, et utilisant un type d'encodage appelé encodage Rice/Golomb [2] : c'est le container retenu pour la norme DCP-IAB.

Le **DolbyAtmos** - et les autres DCP-IAB - utiliseront le **AudioDataDLC**, en 48 kHz (96 kHz [8] est supporté mais les contraintes IAB obligent à ne pas l'utiliser pour l'instant), avec un bitdepth à 24 bits.

L'**AudioDataID** permet d'identifier un asset à travers les différents **IAElements** et les différents **IAFrame**, quelques règles cependant :

- Si l'**AudioDataID** est NULL, cela indique qu'il n'y a aucun asset audio (il est interdit de l'utiliser pour un AudioData, il peut être utilisé dans un Bed Definition ou Object Definition)
- Si l'**AudioDataID** est à zéro (0), il y a bien un asset mono mais c'est un silence.

AUDIO DATA DLC : L'ÉLU POUR LE DCP-IAB !

Cette partie étant encore à l'étude.
Désolé si les informations contenues dans ce paragraphe sont volontairement vagues ou incomplètes.
(voire peut être erronées.. je n'espère pas :)

L'IAElement **Audio Data DLC** est l'endroit où sont stockés les différents éléments audios.

C'est l'IAElement qui va demander le plus de travail car son stockage n'est pas comme celui de l'**Audio Data PCM** (qui ne sera jamais utilisé dans un DCP-IAB) où il suffit d'extraire les données pour avoir des éléments sonores.

Ici, nous allons devoir reconstituer les données et les "réencoder" pour avoir une sortie PCM non-compressée et audible.

Dans un **Audio Data DLC**, vous aurez deux parties principales :

- La partie pour le **48 kHz - obligatoire**
- La partie pour le **96 kHz - facultative**

Vous aurez également deux types d'encodage à l'intérieur d'un DLC :

- L'encodage **PCM**
- L'encodage **Rice/Golomb**

Les deux types d'encodage peuvent cohabiter dans le même DLC dans différents sous-blocs (aka `NUM_DLC_SUB_BLOCKS`). Vous pouvez donc passer d'un bloc PCM à un bloc Rice/Golomb ou inversement. Le tout permettant de recréer l'asset sonore d'origine et audible.

Note: L'encodage et le décodage sont présentés dans la norme SMPTE 2098-2 - Immersive Audio Bitstream - Annexe B.

Voici la structure en pseudo-code d'un **Audio Data DLC** :

```
ElementData "AudioDataDLC" {                        # --Taille:--
    <AudioDataID>                                   #   Plex(8)
    <DLCSize>                                       #   16 bits
    <DLCSampleRate>                                 #   2 bits
    <ShiftBits>                                     #   5 bits

    <NumPredRegions48>                              #   2 bits
    foreach NumPredRegions48 {
        <RegionLength48>                            #   4 bits
        <Order48>                                   #   5 bits
        foreach Order48 {
            <KCoeff48>                              #   10 bits
        }
    }

    # Coded Residual
    foreach NUM_DLC_SUB_BLOCKS {                    #   Table 30 - BlockSize
        <CodeType>                                  #   1 bit

        # ----- Direct PCM -----
        if CodeType == 0 {
            <BitDepth>                              #   5 bits
            foreach DLC_SUB_BLOCK_SIZE48 {          #   Table 30 - BlockSize/DLC_SUB_BLOCK_SIZE (48 kHz)
                <Residual48>                        #   BitDepth (DCP-IAB : 24 bits)
                if Residual48 != 0 {
                    <Sign>                          #   1 bit
                    if Sign == 1 {
                        Residual48 *= -1
                    }
                }
            }
        }

        # ----- Rice/Golomb ------
        if CodeType == 1 {
            <RiceRemBits>                           #   5 bits
            foreach DLC_SUB_BLOCK_SIZE48 {          #   Table 30 - BlockSize/DLC_SUB_BLOCK_SIZE (48 kHz)
                quotient = 0
                <UnaryBit>                          #   1 bit
                while UnaryBit == 1 {
                    quotient++
                    <UnaryBit>                      #   1 bit
                }
                <Residual48>                        #   RicemBits
                Residual += quotient << RiceRemBits
                if Residual48 != 0 {
                    <Sign>                          #   1 bit
                    if Sign == 1 {
                        Residual48 *= -1
                    }
                }
            }
        } # RiceColomb
```

```
    }

    # 96 kHz Extension Layer
    if SampleRate == 0x1 {

        # Predictor informations
        <NumPredRegions96>                              #   2 bits
        foreach NumPredRegions96 {
            <RegionLength96>                            #   4 bits
            <Order96>                                   #   5 bits
            foreach Order96 {
                <KCoeff96>                              #   10 bits
            }
        }

        # Coded Residual
        foreach NUM_DLC_SUB_BLOCKS {                    #   Table 30 - BlockSize
            <CodeType>                                  #   1 bit

            # ---- Direct PCM ------
            if CodeType == 0 {
                <BitDepth>                              #   5 bits
                foreach DLC_SUB_BLOCK_SIZE96 {          #   Table 30 - BlockSize/DLC_SUB_BLOCK_SIZE (96 kHz)
                    <Residual96>                        #   BitDepth (DCP-IAB : 24 bits)
                    if Residual96 != 0 {
                        <Sign>                          #   1 bit
                        if Sign == 1 {
                            Residual96 *= -1
                        }
                    }

                }
            }

            # ---- Rice/Golomb -----
            if CodeType == 1 {
                <RiceRemBits>                           #   5 bits
                foreach DLC_SUB_BLOCK_SIZE96 {          #   Table 30 - BlockSize/DLC_SUB_BLOCK_SIZE (96 kHz)
                    quotient = 0
                    <UnaryBit>                          #   1 bit
                    while UnaryBit == 1 {
                        quotient++
                        <UnaryBit>                      #   1 bit
                    }
                    <Residual96>                        #   RiceRemBits
                    Residual96 += quotient << RiceRemBits
                    if Residual96 != 0 {
                        <Sign>                          #   1 bit
                        if Sign == 1 {
                            Residual96 *= -1
                        }
                    }
                }
            } # RiceColomb

        }
    }
    <AlignBits>                                         #   Variable
}
```

AudioDataID va être utilisé par les IAElements **BedDefinition** et **ObjectDefinition** pour manipuler l'objet sonore. **AudioDataID** sera unique par objet mais peut se trouver propagé dans plusieurs IAFrame dû au fait qu'on aura des morceaux (chunk) d'un objet qui sera sur plusieurs KLV.

DLCSize - sur 16 bits, donc un chiffre maximum de 65535, est la taille qui reste en dessous de **DLCSize**. Par exemple, si tout votre **ElementData** fait 500 octets. Que **AudioDataID** fait 1 octet (Plex(8)), que **DLCSize** fait 2 octets (16 bits), alors nous aurons un **DLCSize** à 497 bytes.

48 versus 96

Les noms communs comme NumPredRegions48 / NumPredRegions96, RegionLength48 / RegionLength96, Order48 / Order96, KCoeff48 / KCoeff96, Residual48 / Residual96 représentant la même chose, elles seront nommées **NumPredRegions**, **RegionLength**, **Order**, **KCoeff** et **Residual** dans le reste du paragraphe.

NumPredRegions (2 bits) est le nombre de régions prédicitives sur le 48 kHz ou sur le 96kHz. Une région va regrouper plusieurs blocs dont le nombre sera indiqué dans **RegionLength**. Si la valeur est à 0, alors il est désactivé.

RegionLength (4 bits) indique le nombre de bloc. A priori, selon la norme, le nombre **RegionLength** sera égale à `NUM_DLC_SUB_BLOCKS`.

Par exemple, si vous avez un `NUM_DLC_SUB_BLOCKS` à 10 :

- Si vous avez un **NumPredRegions** à 1, alors **RegionLength** sera à 10
- Si vous avez un **NumPredRegions** à 5, alors **RegionLength** sera à 2

- Si vous avez un **NumPredRegions** à 10, alors **RegionLength** sera à 1 (ça n'arrivera jamais ;-)

En règle générale, vous verrez un NumPredRegions à 1 et un RegionLength à 10 (car FrameRate à 24, donc `NUM_DLC_SUB_BLOCKS` à 10, voir tableau 30)

Order définit ... l'ordre dans le filtre pour le prédictif. :-D Ce chiffre sera en corrélation avec le KCoeff.

KCoeff est un coéfficient de prédiction. Vous aurez une valeur entre 0 et 1023.
À compléter / retravailler avec l'annexe B pour le calcul avec le coefficient)

CodeType va indiquer si nous avons affaire à un encodage **Direct PCM** (`0x0`) ou un encodage **Rice/Golomb** (`0x1`)

Les variables statiques **`NUM_DLC_SUB_BLOCS`** et **`DLC_SUB_BLOCK_SIZE`** sont définis dans le tableau 30.

Residual contient le signal résiduel : À compléter / retravailler

- Selon si c'est du PCM, ça sera codé en binaire directement avec la taille définie par BitDepth.
- Si c'est c'est du Rice/Golomb, il faudra faire un petit calcul entre RiceRembits, UnaryBit/Quotient et Residual.

Sign va juste définir le signe final pour le signal résiduel :

- Si c'est à 0, alors le residual est positif
- Si c'est à 1, alors le residual est négatif

Dans la partie **Rice/Golomb**, vous aurez **UnaryBit** et **RiceRembits** comme items principaux :

- **RiceRemBits** va définir le nombre de bits utilisé lors du processing du Residual, il indique le nombre de bits utilisés pour coder les bits restants (bypass bits) après le *unary coded quotient*. À compléter / retravailler
- **UnaryBit** spécifie le quotient unaire encodé pour le signal résiduel. Un code à 1 indique une incrémentation dans le quotient. Un code à 0 indique la terminaison l'encodage unaire du quotient. À compléter / retravailler

Note de recherche et d'étude :

Dans la documentation Dolby Atmos, dans la partie 96 kHz, sous-partie Rice-Colomb, Dolby n'effectue pas les mêmes calculs, le calcul UnaryBit/Quotient/Sign n'apparait pas, il stocke directement le Residual96

```
RiceCode                        # 5 bits
foreach SubBlockSize {
    Residual                    # Variable
}
```

EXTRACTION ET RECONSTRUCTION DES DONNÉES AUDIO

A terminer

AUDIO DATA PCM : LE RECALÉ DU DCP-IAB

Ce container est interdit dans un DCP-IAB

Le container **Audio Data PCM** est défini dans la norme **Immersive Audio Bitstream** mais il est exclu de la norme DCP-IAB. Vous pourrez donc le retrouver dans d'autre MXF dans d'autres contextes mais absolument pas dans un DCP.

Ce container est le plus simple, le contenu audio mono est stocké à même dans le container, vous aurez de l'encodage RIFF/WAV - PCM non-compressé.

Au niveau de la norme IAB, nous avons le choix dans la taille des blocs PCMData entre 16 ou 24 bits. Si dans un futur, l'Audio Data PCM pouvait être utilisé dans un DCP, la norme DCP-IAB imposerait tout de même une taille de 24 bits par PCMData.

```
ElementData "AudioDataPCM" {                    # --Taille:--
    <AudioDataID>                               #   Plex(8)
    foreach SAMPLE_COUNT {                      #   Table 18 - FrameRateCode to SAMPLE_COUNT
        <PCMData>                               #   16 ou 24 bits (IAB)
    }
    <AlignBits>                                 #   Variable
}
```

Pour extraire les données, il suffit de créer un header RIFF/WAV et de pousser tous les PCMData à la suite :

Voici l'exemple d'un pseudo-code python pour extraire les données au format RIFF/WAV - pour des IAB 48 kHz - 24 bits :

```
with wave.open("out.wav", "wb") as file:

    # creation header RIFF/WAV
    file.setnchannels(1)        # 1 channel
    file.setsampwidth(3)        # bit-depth (3 bytes : 24 bits)
    file.setframerate(48000)    # sampling-rate (48kHz)
    #
    # (...)
    #
    # IAElement "Audio Data PCM"
    # (...)
    SampleCount = 2000                      # 48 kHz / 24 FPS - Table 18 - SampleCount
    for i in range(0, SampleCount):
        data = IAElement.read("bits:24")
        file.write(data.bytes)
```

Attention, c'est un code ultra simpliste pour rapidement comprendre. Dans cet exemple, on ne prend pas en compte l'AudioDataID. Il faudrait écrire chaque `data` dans le bon asset audio identifié via son AudioDataID. Dans cet exemple, c'est comme si nous extractions un simple chunk sonore.

Pour reconstruire un asset audio complet, vous devriez vous appuyer sur l'AudioDataID pour traquer chaque asset audio, lire chaque KLV, chaque bitstream et chaque IAElement "Audio Data DLC" pour reconstruire les différents canaux (Bed) et les différents objets sonores.

L'ENCODAGE PLEX : LES TAILLES DYNAMIQUES

Vous remarquerez l'indication de certaines tailles avec un mystérieux **Plex(x)**.

Plex est un encodage particulier qui permet d'avoir une taille dynamique dans le stockage de la valeur finale.

Il existe deux types de Plex :

- Avec **Plex(4)**, on va démarrer le processus de décodage en commençant par lire 4 premiers bits.
- Avec **Plex(8)**, on va démarrer le processus de décodage en commençant par lire 8 premiers bits.

La différence entre les deux est le nombre de bits qu'on va lire en tout premier dans le processus de décodage. Le principe de Plex est de lire les premiers blocs de bits pour savoir si la valeur finale est stockée en 4, 8, 16 ou 32 bits.

Vous allez comprendre, mettons cela de côté et voyons dans le concret.

Avec **Plex(4)**, nous allons lire les 4 premiers octets. Si leur valeur est inférieure à `0xF` (`1111` en binaire), alors c'est notre valeur. Si c'est égal à `0xF` , alors il faudra lire les 8 octets suivants. Si les 8 octets suivants sont égaux à `0xFF` (`1111.1111` en binaire), alors il faudra lire les 16 octets suivants. S'ils sont égaux à `0xFFFF` (`1111.1111.1111.1111` en binaire), alors il faudra lire les 32 octets suivants. Bien entendu, pour chacune des étapes, si leurs valeurs ne sont pas `0xF` , `0xFF` ou ni `0xFFFF` , alors c'est la bonne valeur.

Effectuons un exemple concret en commençant par lire 4 premiers bits pour mieux comprendre le concept, en utilisant Plex(4) et donc en lisant les 4 premiers bits :

```
# méthode - étape - read(bits) = valeur
plex(4) - 1 - read(4)   = '3'
```

La valeur récupérée étant inférieure à `0xF` , alors notre valeur finale **Plex(4)** est `0x3` .

Voyons maintenant avec une autre valeur dans les 4 premiers bits :

```
plex(4) - 1 - read(4)   = 'f'       ──┐           il faut lire les 8 prochains bits
plex(4) - 2 - read(8)   = '5d'      <─┘
```

Dans les 4 premiers bits, nous avons la valeur `0xf` , nous devons donc lire les 8 prochains bits. Ces derniers donnent une valeur inférieure à `0xff` : notre valeur finale **Plex(4)** est donc `0x5d` .

Continuons la suite :

```
plex(4) - 1 - read(4)   = 'f'       ──┐           il faut lire les 8 prochains bits
plex(4) - 2 - read(8)   = 'ff'        └──┐        il faut lire les 16 prochains bits
plex(4) - 3 - read(16)  = '7643'       <─┘
```

Même principe que précédemment, notre valeur finale **Plex(4)** en 16 bits est `0x7643` .

Continuons de nouveau :

```
plex(4) - 1 - read(4)   = 'f'       ──┐           il faut lire les 8 prochains bits
plex(4) - 2 - read(8)   = 'ff'        └──┐        il faut lire les 16 prochains bits
plex(4) - 3 - read(16)  = 'ffff'         └──┐     il faut lire les 32 prochains bits
plex(4) - 4 - read(32)  = 'ef32ac01'      <─┘
```

Notre valeur finale Plex(4) en 32 bits est donc `0xEF32AC01` .

Avec **Plex(8)**, nous avons le même procédé, sauf qu'on débute par la lecture des 8 bits premiers directement :

```
plex(8) - 1 - read(8)   = '66'
```

```
plex(8) - 1 - read(8)   = 'ff'        ──┐         il faut lire les 16 prochains bits
plex(8) - 2 - read(16)  = '013d'      <─┘
```

```
plex(8) - 1 - read(8)   = 'ff'        ──┐         il faut lire les 16 prochains bits
plex(8) - 2 - read(16)  = 'ffff'        └──┐      il faut lire les 32 prochains bits
plex(8) - 3 - read(32)  = '000101b7'  <────┘
```

On s'arrête toujours à 32 bits, il n'y aura pas de 64 bits.

La valeur maximale finale d'un Plex sera de `0xFFFFFFFE` (soit 4.294.967.294 en décimal)

ANALYSE DU MXF

Vous avez un parseur d'Immersive Audio Bitstream disponible ici, à tester avec les assets IAB disponible ici.

Pour l'utiliser, il suffit d'exécuter `iab-reader` sur une frame IAB (par exemple celle-ci) ou directement sur un répertoire contenant plusieurs IAB :

```
+++ atmos-stresstest-64objects.iab/*-ImmersiveAudioDataElement.value.bin
┌──────────────────────────────────────────────────────────────────────────────────────────────┐
│┣━━━ IABitstream Frame ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┫ [1]
│┣━━━━ Preamble ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ [1]
│      PreambuleTag : 0x01
│      PreambuleLength : 1603 bytes
│      PreambuleValue : 11010effffffffffffffffff(...1571 bytes...)ffffffffffffffffffffffff
│┣━━━━ IAFrame ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ [1]
│      IAFrameTag : 0x02
│      IAFrameLength : 44501 bytes
│      IAFrameValue : 08ffadd10110529bff0200ff020(...44469 bytes...)1bfffc0300000200000000500
│┣━━━━ IAElement ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ [1]
.
.
.
(....)
```

ANNEXES

TABLE 23 : NOMBRE DE SUBBLOCKS PAR IA_FRAME

FrameRate	`NUM_PAN_SUB_BLOCKS`	SubBlock Duration (ms)
24	8	5.2 ms
25	8	5.0 ms
30	8	4.2 ms
48	4	5.2 ms
50	4	5.0 ms
60	4	4.2 ms
96	2	5.2 ms
100	2	5.0 ms
120	2	4.2 ms

TABLE 30 - BLOCKSIZE

FrameRate	NUM_DLC_SUB_BLOCKS	DLC_SUB_BLOCK_SIZE (48 kHz)	DLC_SUB_BLOCK_SIZE (96 kHz)	Rapport 48->96
24	10	200	400	x2
25	10	192	384	x2
30	8	200	400	x2
48	5	200	400	x2
50	5	192	384	x2
60	4	200	400	x2
96	5	100	200	x2
100	4	120	240	x2
120	4	100	200	x2

TABLE 18 - VALEURS DE SAMPLE COUNT

Code FrameRate	FrameRate Réel	SAMPLE_COUNT (48 kHz)	SAMPLE_COUNT (96 kHz)
0x0	24 fps	2000	4000
0x1	25 fps	1920	3840
0x2	30 fps	1600	3200
0x3	48 fps	1000	2000
0x4	50 fps	960	1920
0x5	60 fps	800	1600
0x6	96 fps	500	1000
0x7	100 fps	480	960
0x8	120 fps	400	800
0x9 - 0xF	Réservé	Reservé	Reservé

TABLE 24 : NOMS DES ZONE DEFINITION :

Number	Description
0	All screen speakers left of center
1	Screen center speakers
2	All screen speakers right of center
3	All speakers on left wall
4	All speakers on right wall
5	All speakers on left half of rear wall
6	All speakers on right half of rear wall
7	All overhead speakers left of center
8	All overhead speakers right of center

TABLE 26 - CODE POUR LE MODE OBJECT SPREAD :

Identifier	Code	Description
OBJECT_SPREAD_LOWREZ	0x0	Equal Spreading in each dimension (8 bits coding)
OBJECT_SPREAD_NONE	0x1	Point Source (no Object Spread is sent)
OBJECT_SPREAD_1D	0x2	Equal Spreading in each dimension (12 bits coding)
OBJECT_SPREAD_3D	0x3	Specified Spreading in each dimension (12 bits coding for each point X, Y, Z) - Non utilisé par Atmos

TABLE 28 : NOMS DES DIFFÉRENTES OBJECT ZONE DEFINITION

Number	Description
0	Base layer screen Loudspeakers left of center
1	Base layer center screen Loudspeakers
2	Base layer Loudspeakers right of center
3	Height layer screen Loudspeakers left of center
4	Height layer center screen Loudspeakers
5	Height layer screen Loudspeakers right of center
6	Base layer rear wall Loudspeakers left of center
7	Base layer rear wall center Loudspeakers
8	Base layer rear wall Loudspeakers right of center
9	Height layer rear wall Loudspeakers left of center
10	Height layer rear wall center Loudspeakers
11	Height layer rear wall Loudspeakers right of center
12	Base layer left wall Loudspeakers
13	Height layer left wall Loudspeakers
14	Base layer right wall Loudspeakers
15	Height layer right wall Loudspeakers
16	Ceiling Loudspeakers left of center
17	Center ceiling Loudspeakers
18	Ceiling Loudspeakers right of center

TABLE 19 : CODE POUR LES CHANNEL ID

Code	Description	DCP-IAB Constraints
`0x0`	Left	DCP-IAB
`0x1`	Left Center	DCP-IAB
`0x2`	Center	DCP-IAB
`0x3`	Right Center	DCP-IAB
`0x4`	Right	DCP-IAB
`0x5`	Left Side Surround	DCP-IAB
`0x6`	Left Surround	DCP-IAB
`0x7`	Left Rear Surround	DCP-IAB
`0x8`	Right Rear Surround	DCP-IAB
`0x9`	Right Side Surround	DCP-IAB
`0xa`	Right Surround	DCP-IAB
`0xb`	Left Top Surround	DCP-IAB
`0xc`	Right Top Surround	DCP-IAB
`0xd`	LFE	DCP-IAB
`0xe`	Left Height	
`0xf`	Right Height	
`0x10`	Center Height	
`0x11`	Left Surround Height	
`0x12`	Right Surround Height	
`0x13`	Left Side Surround Height	
`0x14`	Right Side Surround Height	
`0x15`	Left Rear Surround Height	
`0x16`	Right Rear Surround Height	
`0x17`	Top Surround	

LIMITATION DE TAILLE D'UNE FRAME IMMERSIVE AUDIO BITSTREAM :

FrameRate	Taille maximale
24	781.250 octets
25	750.000 octets
30	625.000 octets
48	390.625 octets
50	375.000 octets
60	312.500 octets
96	195.313 octets
100	187.500 octets
120	156.250 octets

Source: SMPTE 429-18-2019 - DCP - Immersive Audio Track File

NOTES

1. *« First Frame of Action (FFOA): The first frame of a post-production reel that contains image action pertinent to the program. For audio elements, it is the location of the sound that is intended to sync with the first frame of image action. The location of this frame is depicted by the location of the left edge of the frame. For example, the FFOA is 8 seconds (192 frames at 24 fps) from the left edge of the picture start frame on the leader on a post-production reel of 35 mm film. The same concept applies to D-Cinema. »* -- SMPTE 428-4-2010 - DCDM - Audio File Format and Delivery

Constraints ↩

2. Les restrictions de la norme DCP-IAB sur les **IAElements** : ↩
   - IAElement **Bed Remap** *« Bitstream shall not contain BedRemap »* -- SMPTE RDD57-2021 - Immersive Audio Bitstream & Packaging Constraints - IAB Application Profile 1
   - IAElement **Object Zone Definition** *« Bitstream shall not contain ObjectZoneDefinition »* -- SMPTE RDD57-2021 - Immersive Audio Bitstream & Packaging Constraints - IAB Application Profile 1
   - IAElement **Authoring Tool Info** *« Bitstream shall not contain the AuthoringToolInfo element »* -- SMPTE RDD57-2021 - Immersive Audio Bitstream & Packaging Constraints - IAB Application Profile 1
   - IAElement **Audio Data PCM** *« Bitstreams shall not contain the AudioDataPCM element. Instead, AudioDataDLC shall be used for all audio essence. »* -- SMPTE 429-18 - DCP - Immersive Audio Track File
   - IAElement **User Data** *« The UserData element as defined in ST 2098-2 shall not be used. »* -- SMPTE 429-18 - DCP - Immersive Audio Track File
3. Le nombre maximal d'Audio Bed est de 10 pour un DCP. Il peut monter jusqu'à 64 mais seulement sur un master Dolby Atmos. ↩
4. **La méthode de calcul du 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 20 = 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. » -- IAB specifications

   « Amplitude Gain: Many of the metadata elements contained in the bitstream specify a gain or scale factor to be applied to audio data. Throughout the bitstream, amplitude information is coded as a logarithmic value using a 10 bit mantissa, A10, to express values in the range [0, 1]. Thus the logarithmic code word, interpreted as an unsigned integer, can be mapped to a linear value as follows »

   ```
   gain = 0,              A10 = (2^10) - 1
   gain = 2^(-A10/64),    0 <= A10 <= (2^10) – 2
   ```

   « This provides gain factors logarithmically spaced from 0 dB to -96 dB in steps of approximately 0.094 dB, plus a gain factor of zero (-infinity dB). » -- Dolby Atmos Specification
5. *« SubElementCount of BedDefinition shall be set to “0” »* -- SMPTE RDD-57-2021 - Immersive Audio Bitstream and Packaging Constraints - IAB Application Profile 1 ↩
6. Les intervalles des mises-à-jour de la spatialisation : ↩
   - *« The NumPanSubBlocks specifies the division of the frame into sub frames of approximately 5 ms, as specified by Table 7. »* -- Dolby Atmos Specifications
   - *« The ObjectDefinition element provides all the information to pan an audio object. Each ObjectDefinition element updates the position of a single audio object at approximately 20-ms time intervals. »* -- Dolby Atmos Specifications
7. L'encodage de la distance : ↩

   ```
   DistanceXY = Dn/2(n-1) – (2(n-1)-1)/2(n-1),        2n-1-1 <= Dn <= (2n) – 1
   DistanceZ  = Dn/(2n – 1),                          0 <= Dn <= (2n) – 1
   ```
8. Le support et les contraintes du 48 kHz / 96 kHz : ces contraintes sont probablement dûes de par le matériel qui doit supporter du 48 kHz et pourrait supporter du 96 kHz. ↩
   - *« The compression system supports both 48 kHz and 96 kHz sample rates. »* -- SMPTE 2098-2-2018 - Immersive Audio Bitstream Specification
   - *« AudioDataDLC Element : The AudioDataDLC element supports sample rates of 48 kHz or 96 kHz with 24-bit resolution. »* -- SMPTE RDD-29:2019 - Dolby Atmos Bitstream.
   - *« ST 2098-2 Bitstream Constraints : Sample Rate : The value of all instances of Sample Rate, as defined by SMPTE ST 2098-2, shall be 48 kHz. »* -- SMPTE.RDD.57-2021 - Immersive Audio Bitstream and Packaging Constraints - IAB Application Profile 1