MXF : SOUND

Références	**SMPTE 377-1-2019** - MXF - File Format Specification **SMPTE 429-3-2007** - DCP - Sound And Pictures Track File **SMPTE 429-2-2013** - DCP - DCP Operational Constraints **SMPTE 428-12-2013** - DCDM - Common Audio Channels and Soundfield Groups **SMPTE 377-4** - MXF Multichannel Audio Labeling Framework **SMPTE 382M-2007** - MXF - Mapping AES3 and Broadcast Wave Audio into the MXF Generic Container **SMPTE.RDD.0052-2020-D** - Cinema Packaging-SMPTE DCP Bv2.1 Application Profile **Annexes** : - Résumé rapide du WAVE - Résumé rapide du RIFF - Résumé rapide du PCM - Condensé technique du WAV/BWF - Résumé rapide du format BWF - Norme Broadcast Wave Format (BWF)
Modèle KLV	Wave Audio Essence Descriptor : Data Groups - Local Sets Essence non-chiffré (Sound Essence) : Data Item (en clair) Essence chiffré (Encrypted Essence) : Variable-Length Pack (chiffré)

Universal Label	`06.0e.2b.34.02.53.01.01.0d.01.01.01.01.01.48.00 - Wave Audio Essence Descriptor`
	`06.0e.2b.34.02.04.01.01.0d.01.03.01.02.7e.01.00 - Encrypted Essence`
	`06.0e.2b.34.01.02.01.01.0d.01.03.01.16.01.01.01 - Sound Essence - Wave Frame-Wrapped Element - Data Item`

Ce chapitre n'évoque que l'élement de base d'un DCP : Sound Essence (MXF) ou MainSound (CPL)
Et non les autres possibilités sonores comme l'Immersive Audio ou Dolby Atmos

PRÉFACE

C'est l'élément de base dans un DCP classique.

Les **données audios stockées** (en claires ou chiffrées) sont au format **PCM WAVE/RIFF** [1] avec des samples à 24 bits (portions très courtes de données) par canaux (Left, Right, LFE, etc...) pouvant aller jusqu'à 16 canaux :

Encodage	**PCM WAVE/RIFF non-compressé et canaux entrelacés**
Sample Bitdepth	24 bits / sample
Sample Rate	48 kHz ou 96 kHz [2]
Canaux maximum	16 channels

Un MXF Audio sera constitué des différents éléments de base (minimum) :

- Un KLV **Wave Audio Essence Descriptor** : une description des données audio, les métadonnées utiles pour la suite.
- Un ou plusieurs **Sound Essence** : Les données audios brutes au format PCM WAVE non-compressé, canaux entrelacés et sans entêtes WAV/BWF.

Et d'autres KLV pour l'audio ?

D'autres KLV peuvent compléter un MXF Audio si et seulement si on utilise du **Multichannel Audio** (MCA) :

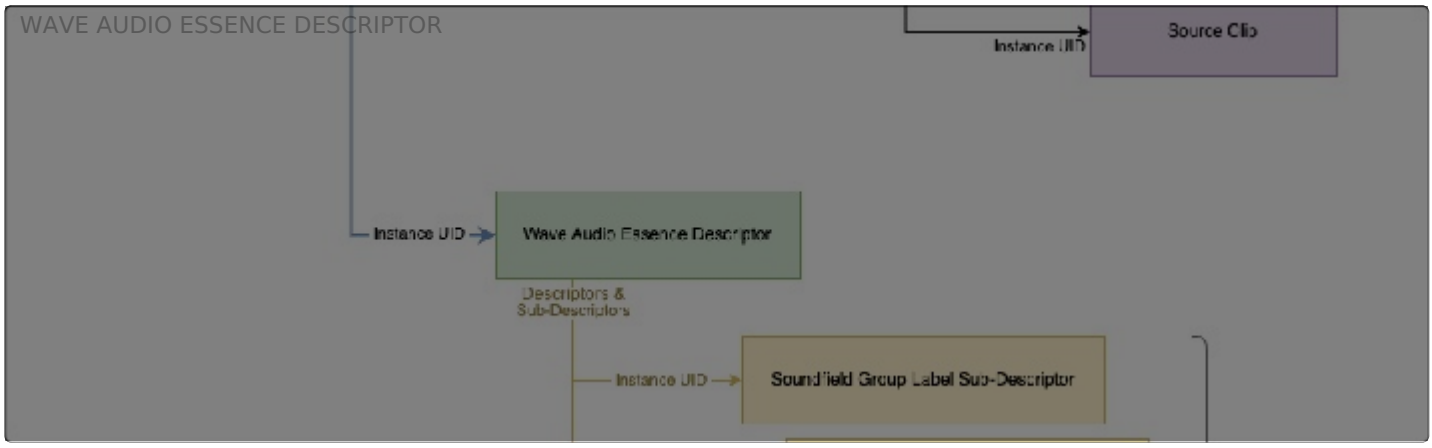
- KLV **Soundfield Group Label Sub-Descriptor**
- KLV **Channel Label Sub-Descriptor**

Le **Multichannel Audio** (MCA) est une option permettant de configurer précisement les différents canaux audios

et permet d'avoir un dynamisme dans leurs attributions.

Ce sujet précis étant trop vaste pour ce chapitre, vous retrouverez plus d'informations dans un chapitre à part Configuration Audio & Multichannel Audio (MCA).

LES KLV DE MÉTADONNÉES (PARTITION HEADER)



De base, nous aurons au moins un KLV spécifique : le KLV **Wave Audio Essence Descriptor**.

Ce dernier va donner toutes les caractéristiques techniques des essences stockées dans la partie Body (Sound Essence) et des spécifications annexes avec des sub-descriptors (notamment le **Multi Channel Audio** Framework, aka MCA)

Son Universal Label (SMPTE) est `060e2b34.02530101.0d010101.01014800` Il est de type **Local Sets** (Bébé KLV) avec des **Local Tag**.

Chaque item est une information permettant de décrire l'essence, un exemple avec **Wave Audio Essence Descriptor** assez simple :

```
╔═══════════════════════════════╦═══════════════════════════════════════════════
╟  3C0A - Instance UID          ╟  789a21c7.7ff04185.b85f00f5.7bb5d7f7
╟  3006 - Linked Track ID       ╟  1
╟  3001 - Sample Rate           ╟  24/1
╟  3002 - Container Duration    ╟  24
╟  3004 - Essence Container     ╟  060e2b34.04010101.0d010301.02060100
╟                               ╟  (Broadcast Wave audio - frame-based mapping)
╟  3D03 - Audio sampling rate   ╟  48000/1
╟  3D02 - Locked/Unlocked       ╟  False
╟  3D07 - ChannelCount          ╟  6
╟  3D01 - Quantization bits     ╟  24
╟  3D0A - Block Align           ╟  18
╟  3D09 - Average Bytes Per Second ╟  864000
╟  3D32 - Channel Assignment    ╟  060e2b34.0401010b.04020210.03010100
╟                               ╟  (SMPTE-429-2 Channel Configuration 1)
╚═══════════════════════════════╩═══════════════════════════════════════════════
```

Les données utiles sont surtout **ChannelCount**, **QuantizationBits** - qui donne la taille de chaque portion (sample) par channel - et **Audio Sampling Rate**. Ces données vont être utiles pour construire un entête pour notre export audio.

À noter quelques règles de base :

- **ChannelCount** sera toujours un nombre pair [3] (2, 4, 6, 8, 10, 12, 14 ou 16),
- **QuantizationBits** sera toujours à 24 bits
- **Audio Sampling Rate** sera soit 48000/1, soit 96000/1.

Le **Block Align** est utile pour les appareils nécessitant un alignement des bits, et pour le calculer :

```
BlockAlign    =  ChannelCount * floor( ( QuantizationBits + 7 ) / 8 )
ChannelCount  =    BlockAlign / floor( ( QuantizationBits + 7 ) / 8 )
```

Ces calculs [4] ne marchent que pour les PCM non-compressés.

Channel Assignment est important, car il va spécifier dans quelle configuration audio nous serons. Il en existe

officiellement 6 : 5 statiques et 1 dynamique (**Multichannel Audio (MCA)**).

Universal Label	Description	Type
`060e2b340401010b0402021003010100`	Channel Configuration 1	Statique
`060e2b340401010b0402021003010200`	Channel Configuration 2	Statique
`060e2b340401010b0402021003010300`	Channel Configuration 3	Statique
`060e2b340401010b0402021003010400`	Channel Configuration 4	Statique / Dynamique
`060e2b340401010b0402021003010500`	Channel Configuration 5	Statique
`060e2b340401010b0402021003020000`	MXF Multichannel Audio Framework (MCA)	Dynamique

La configuration audio est assez large et complexe pour avoir son propre chapitre (voir ci-dessous).

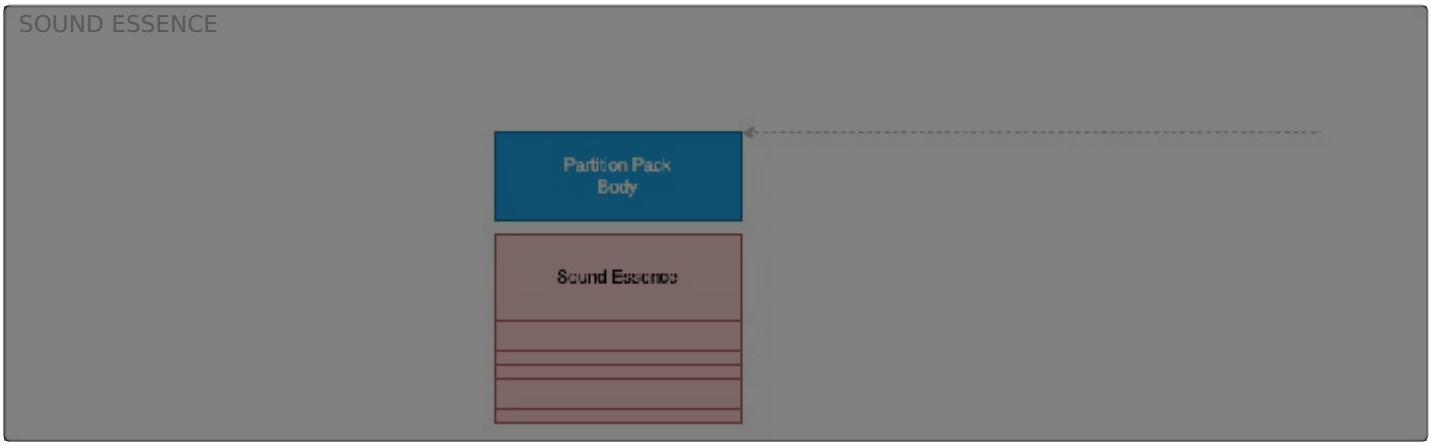
MÉTADONNÉES SUPPLÉMENTAIRES : LA CONFIGURATION SONORE & MULTICHANNEL AUDIO (MCA)

Selon le type de **Channel Assignement**, il existera des métadonnées audios supplémentaires.

Si c'est le cas, vous verrez appaître un item supplémentaire **Descriptor & Sub-Descriptors** à l'intérieur du KLV **Wave Audio Essence Descriptor** stockant plusieurs identifiants qui pointeront vers une succession de KLV supplémentaires possibles nommés **Soundfield Group Label Sub-Descriptor** et **Channel Label Sub-Descriptor**.

Le spectre étant assez large, il possède son propre chapitre Configuration Sonore et Multichannel Audio (MCA).

A L'INTÉRIEUR DU KLV SOUND ESSENCE

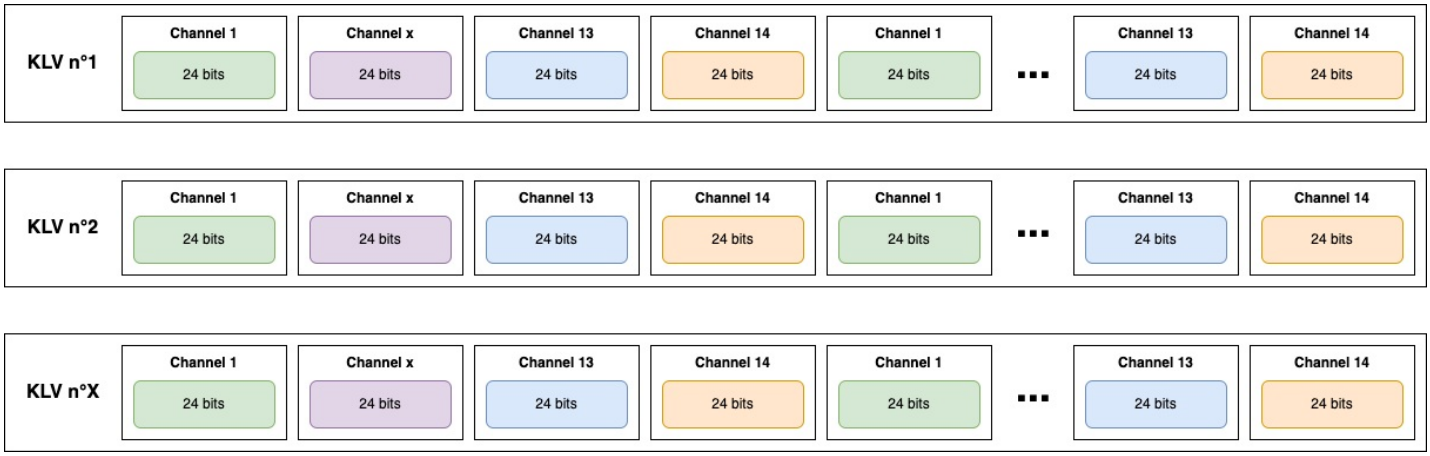


A l'intérieur d'un KLV **Sound Essence**, nous n'avons pas toute piste son, nous avons qu'une portion audio qui va correspondre à une image affichée [5].

Dans cette portion, nous aurons des fragments audio correspondant à chaque canal audio : Ce fragment est appelé **Sample**.

La taille d'un Sample est définie par l'item **Quantization Bits** (appelé aussi **Sample Width**, ou **largeur de l'échantillon** en bon françois) d'une taille fixe de 24 bits (spécifications DCI) : la taille de nos blocs de données par canaux seront toujours de 24 bits chacun.

Pour faire une seconde audible, il faut donc beaucoup KLV avec beaucoup de samples (chaque case de couleur est un sample d'un canal précis, dans notre exemple, nous avons 14 channels) :



Le petit train et ses wagonnets

Voyez cela comme des petits trains avec plusieurs wagonnets qui va transporter différentes matières.
Dans chaque wagonnet, vous aurez une petite portion d'une matière :

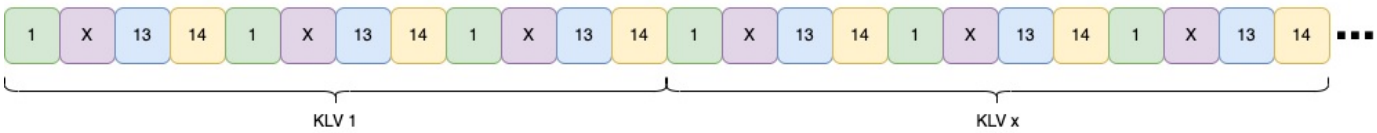
- Dans le 1er wagonnet, du bois
- Dans le 2eme wagonnet, du sable
- Dans le 3eme wagonnet, des pierres

Arriver en gare, vous aurez plusieurs petits trains avec leurs wagonnets. Et pour refaire des gros tas par matière, vous devrez sortir le bois des wagonnets "bois", le sable des wagonnets "sables" et des pierres des wagonnets "pierres".

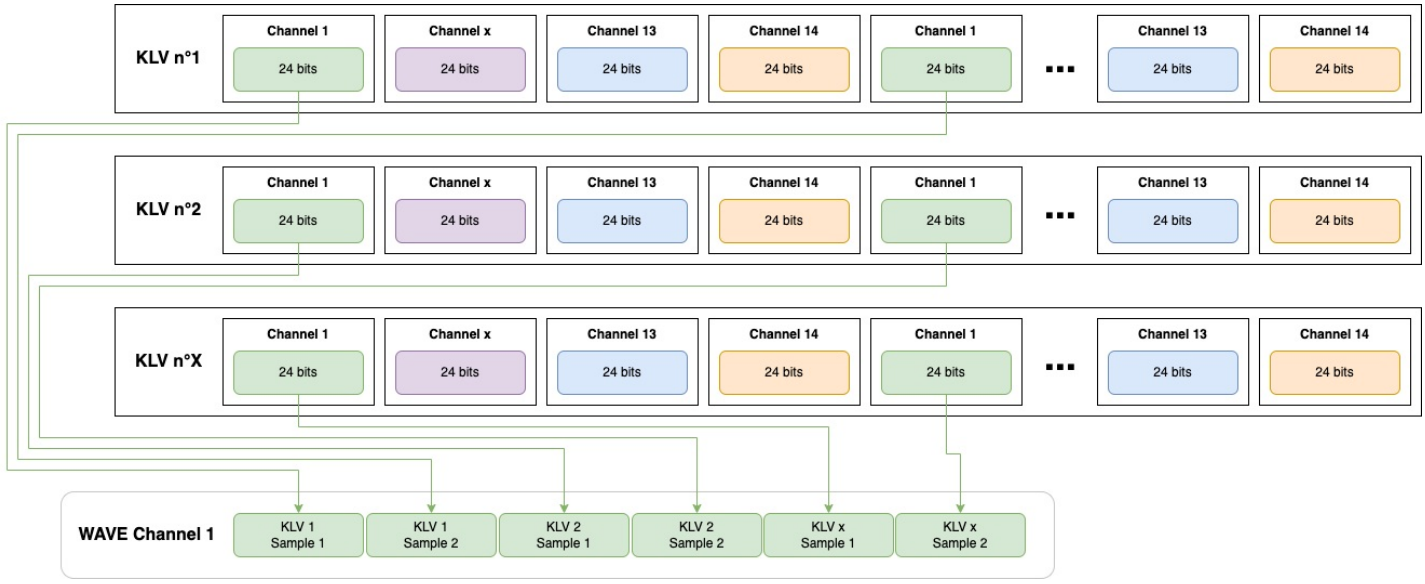
Vous aurez plusieurs samples par KLV et chaque KLV est une courte séquence sonore. Pour avoir l'entièreté d'une piste sonore, il faudra donc concaténer l'ensemble des données depuis les KLV **Sound Essence** (ou les données déchiffrées venant des **Encrypted Sound Essence**)

Autrement dit, les différents canaux sont entrelacés, nous allons passer d'une portion du canal n°1, à une portion du canal n°2, puis une portion du canal n°3 dans un même KLV; pour revenir au canal n°1, etc... puis passer aux KLV suivants.

Si vous concaténez toutes les données de ces KLV, vous aurez les données brutes d'un fichier wav multicanaux sans l'entête wave.



A noter que sur un KLV, vous aurez toujours toutes les portions de sample de l'ensemble des canaux, le KLV ne va pas s'arrêter au beau milieu d'un des canaux. Par exemple, si nous avions 14 canaux, le KLV ne s'arrêtera pas au canal n°10 puis continuerait sur le KLV suivant avec le canal n°11. Tous les KLV débuteront par le sample de 24 bits du canal n°1 et se termineront par le sample de 32 bits du canal n°14.



Vous comprenez alors que pour reécrer un channel complet, vous devrez lire chaque morceau "entrelacé" dans les autres morceaux de channels.

CRÉATION D'UN HEADER WAVE (PYTHON)

Etant donné que la mise au format MXF/KLV d'une piste sonore supprime les headers RIFF/WAV d'origine (contrairement au JPEG2000 où l'header est conservé), il faut donc récréer un header compatible RIFF/WAV :

> Attention, cet exemple ne va créer qu'un header WAVE pour 6 channels en 48 kHz.
> Il faudra adapter si vous avez plus ou moins de channels en plus ou si vous êtes en 96 kHz.
> Notez que le code source suivant "Extraire au format WAV" s'adapte au MXF.

Méthode 1 : création d'un fichier d'entête wav :

```
import wave
with wave.open("header.wav", "wb") as file:
    file.setnchannels(6)      # 6 channels
    file.setsampwidth(3)      # bit-depth (3 bytes : 24 bits)
    file.setframerate(48000)  # sampling-rate (48kHz)
```

Méthode 2 : création d'un entête en mémoire (sans passer par un fichier intermédiaire) :

```
import io
import wave

# create buffer IO
buffer = io.BytesIO()

# create wave file header
with wave.open(buffer, "wb") as file:
    file.setnchannels(6)       # 6 channels
    file.setsampwidth(3)       # bit-depth : 24 bits (3 bytes)
    file.setframerate(48000)   # Sampling rate : 48kHz

buffer.seek(0)                 #
wave_headers = buffer.read()   # RIFF/WAV headers
```

Dans la variable `wave_headers`, vous aurez l'entête BWF/WAVE utile, vous n'aurez plus qu'à faire une simple concaténation de wave_headers et vos données BWF.

Vous trouverez dans chaque langage une librairie vous permettant de manipuler des fichiers RIFF/WAV, et ainsi, vous permettre de générer un header compatible avec ce format.

EXTRAIRE LA PISTE SON AU FORMAT WAV (PYTHON)

Ce programme prend un fichier MXF, va l'analyser, lire chaque KLV **SoundEssence** et créer un fichier WAV lisible avec un bon header BWF :

```
import sys
import io
import wave

# Conversion en int
def to_int(length : bytes = b'') -> int:
    return int.from_bytes(length, byteorder='big')

if len(sys.argv) < 2:
    print("Usage: %s <mxf>" % sys.argv[0])
    sys.exit(1)

mxf_file = sys.argv[1]

with open(mxf_file, "rb") as file:

    # On va lire chaque KLV
    while True:

        # Key : Universal Label
```

```
        key = file.read(16)

        # Fin de fichier
        if not key:
            break

        # La taille du KLV (format BER)
        # BER format : on ne va lire que les 3 derniers bytes
        length = to_int(file.read(4)[1:])

        # On lit la Value
        value = file.read(length)

        # On ne va lire que les KLV :
        #   Header : Wave Audio Essence Descriptor
        #   Body   : Sound Essence
        if key.hex() != "060e2b34025301010d01010101014800" and \
           key.hex() != "060e2b34010201010d01030116010101":
           continue

        # On affiche un résumé des KLV filtrés
        print("{key} - {length:>6d} bytes - {value}...".format(
            key    = key.hex(),
            length = length,
            value  = value[0:16].hex()
        ))


        # -------------------------------------------------
        #        Read header (Wave Audio Essence Descriptor)
        #  On va lire ce header car il va nous donner
        #  les informations nécessaires pour créer
        #  le header RIFF/WAV
        # -------------------------------------------------

        if key.hex() == "060e2b34025301010d01010101014800" :
            print("read headers")

            buffer = io.BytesIO(value)

            # On va lire chaque item (bébé KLV)
            while True:

                localtag = buffer.read(2)
                if not localtag:
                    break
                item_length = to_int(buffer.read(2))
                item_value = buffer.read(item_length)

                # Sampling rate (kHz)
                if localtag.hex() == "3d03":
                    sampling_rate = to_int(item_value[0:4])
                    print("header = Sampling Rate =", sampling_rate)

                # Channel Count (1 to 6 channels)
                if localtag.hex() == "3d07":
                    channel_count = to_int(item_value)
                    print("header = Channel Count =", channel_count)

                # Quantization bits (always 24 bits)
                if localtag.hex() == "3d01":
                    quantization_bits = to_int(item_value)
                    print("header = Quantization bits =", quantization_bits)

            # Création du header RIFF/WAV
            with wave.open("output.wav", "wb") as header:
                header.setnchannels(channel_count)              # channels
                header.setframerate(sampling_rate)              # Sampling rate
                header.setsampwidth(int(quantization_bits/8))   # bit-depth (in bytes)


        # -------------------------------------------------
        #                Read each Essence
        # -------------------------------------------------

        if key.hex() == "060e2b34010201010d01030116010101":
            print("read data audio")
            with open("output.wav", 'ab') as data:
                data.write(value)
```

Dans la partie Header, nous lisons les items et récupérons les 3 éléments dont nous avons besoins pour construire le header de notre fichier RIFF/WAV :

- **Sampling Rate** (pour avoir les kHz)
- **Channel Count** (pour avoir le nombre de canaux sonores)
- **Bit-depth** (Quantization bits) qui est la taille de la portion de données audio de chaque canaux.

Avec ces trois paramètres, nous pouvons les donner à la librairie Wave qui se chargera de créer un header RIFF/WAV avec les bons paramètres.

Notre header RIFF/WAV prêt, nous pouvons lire chaque essence - qui ne sont que les parties du fichier RIFF/WAV d'origine, découpées en morceaux et stockées dans chaque KLV - et nous les ajoutons en brute sans modification - l'une à la suite de l'autre, dans notre fichier de sortie (aucun besoin de librairie spécifique).

Et voila ! vous avez un fichier sonore d'origine.

Vous retrouverez le code source de ce programme ici : mxf-klv-sound-essence.py

ECRIRE NOS PROPRES KLV

Coming soon...

TECHNIQUES

GÉNÉRER UN FICHIER AUDIO DE 10S VIDE RAPIDEMENT :

```
sox -n -r "48000" -c 2 "output.wav" trim 0.0 10.00
```

NOTES

1. A quoi sert le format Broadcast Wave Format (BWF) ? ↩

   Dans la documentation, on parlera du format BWF pour spécifier le format audio utilisé dans notre microcosme.

   Mais pour notre MXF, il ne servira à rien...

   Pour résumer très rapidement, le BWF est un WAVE avec un peu plus de métadonnées et orienté pour l'audiovisuel. Mais sur un MXF, dans les KLV **Sound Essence**, nous n'aurons ni les headers WAVE, ni les extensions de métadonnées BWF, nous n'aurons que les données brutes audio.

   Les métadonnées seront stockées dans la partie Header, dans le KLV **Wave Audio Essence Descriptor**

   Quand nous allons travailler sur les données audios d'un MXF, nous n'aurons que des données au format RIFF/WAV.

   La norme SMPTE évoque le format BWF pour une simple et bonne raison : avoir des métadonnées intégrées dans un seul fichier pour l'ensemble de la postproduction. Et lors de la création d'un DCP, de juste n'avoir qu'à lire les métadonnées du fichier WAVE/BWF pour créer à la fois le KLV **Wave Audio Essence Descriptor** et les **KLV Audio Essence**.

   Si nous n'avons pas ces headers, nous pouvons travailler avec des fichiers WAV tout ce qui a plus classique. A notre charge de créer le bon KLV Header **Wave Audio Essence Descriptor** avec les bonnes valeurs, tout marchera à la normale.

2. Les références 48 kHz et 96 kHz : ↩

   - *« If a media block supports 96 kHz audio, it shall be able to perform sample rate conversion to 48 kHz or 96 kHz at the output when necessary. »* -- DCI specifications
   - *« Audio Track File needs to contain at least 48,000 (at 48kHz sampling rate) or 96,000 (at 96 kHz sampling rate) audio samples. »* -- DCI CTP

   Pour informations, les **Audio Samples Rate** par **Edit Unit** :

Audio Sample Rate -> Edit Rate v	48 kHz	96 kHz
24	2000	4000
25	1920	3840
30	1600	3200
48	1000	2000
50	960	1920
60	800	1600
96	500	1000
100	480	960
120	400	800

3. *« The number of channels within the Sound Track File shall be an even number. »* -- SMPTE 429-2 DCP Operational Constraints ↩

4. Calcul des **BlockAlign** et **ChannelCount** - en Python : ↩

```
# Exemple de valeurs :
# - ChannelCount     = 6
# - QuantizationBits = 24
# - BlockAlign       = 18
#

>>> from math import floor

# Calcul BlockAlign
# BlockAlign = ChannelCount * floor( ( QuantizationBits + 7 ) / 8 )
>>> 6 * floor( ( 24 + 7 ) / 8 )
18

# Calcul ChannelCount
# ChannelCount = BlockAlign / floor( ( QuantizationBits + 7 ) / 8 )
>>> 18 / floor( ( 24 + 7 ) / 8 )
6.0
```

5. Quelle est la durée audio d'un KLV Sound Essence ? ↩

   Un seul KLV **Sound Essence** va intégrer suffisamment de données audio pour la durée d'une frame liée à son **Sample Rate**.

   Autrement dit, pour un **Sample Rate** à 24/1, nous aurons donc 1/24eme de KLV. Inversement, il faut environ 24 KLV audio pour faire 1 seconde [6]. donc environ 41.6 ms par KLV audio.

   Pour vérifier, si vous lisez **Wave Audio Essence Descriptor → Average Bytes Per Second**, vous le divisez par le **Wave Audio Essence Descriptor → Sample Rate**, vous aurez la taille d'un KLV audio.

   Exemple avec un 24/1 non chiffré :

```
<Wave Audio Essence Descriptor>
-------------------------------------------
Sample Rate                ‖  24/1
Average Bytes Per Second   ‖  864000
-------------------------------------------
            864000 / 24   =  36000

<Sound Essence>
-------------------------------------------
Length                     ‖  36000 bytes
-------------------------------------------
```

   Exemple avec un 48/1 chiffré :

```
<Wave Audio Essence Descriptor>
-----------------------------------------
Sample Rate                ‖  48/1
Average Bytes Per Second   ‖  2304000
-----------------------------------------
           2304000 / 48   =  48000

<Encrypted Essence Container>
-----------------------------------------
Source Length              ‖  48000
-----------------------------------------
```

6. 25 KLV pour être précis car nous avons 41.6666666...6666 ms :) ↩