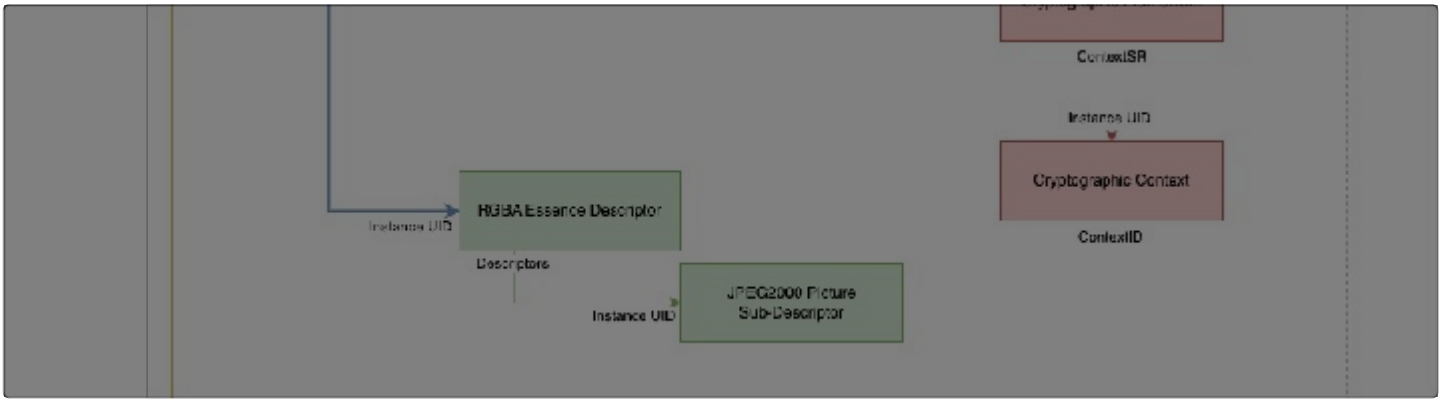
MXF : PICTURE : RGBA ESSENCE DESCRIPTOR

Références	SMPTE 422-2014 - MXF - Mapping JPEG2000 Codestreams into the MXF Generic Container SMPTE 429-4-2006 - DCP - MXF JPEG2000 Application SMPTE 429-3-2007 - DCP - Sound And Pictures Track File [1] SMPTE 377-1-2011 - MXF - File Format Specification - RGBA Essence Descriptor (p148) SMPTE 384M-2005 - MXF - Mapping of Uncompressed Pictures into the Generic Container
Modèle KLV	Data Item

Universal Label	`060e2b34.02530101.0d010101.01012900 - RGBA Essence Descriptor`
	`060e2b34.02530101.0d010101.01015a00 - JPEG2000 Picture Sub-Descriptor`
	
	`060e2b34.01020101.0d010301.15010801 - Picture Essence, non-chiffré`
	`060e2b34.02040101.0d010301.027e0100 - Encrypted Essence (SMPTE)`
	`060e2b34.02040107.0d010301.027e0100 - Encrypted Essence (Interop)`

PRÉFACE



Le KLV **RGBA Picture Essence Descriptor** est un sous-élément du type **Generic Picture Essence Descriptor** :

- Les deux éléments sont définis dans **SMPTE 377-1-2011 - MXF - File Format Specification**.
- Son **Universal Label** est `060e2b34.02530101.0d010101.01012900`

LES MÉTADONNÉES

Voici un rendu d'exemple :

```
╔═══════════════════════════════╦═══════════════════════════════════════════════════════════════════════════════
║   3C0A - Instance ID          ║  2cb46505.be6b41c3.abbe8848.d06e21c9
║   FFFF - SubDescriptors       ║  1 item(s) : 6054268f.8fdf47ba.98db7167.5d3f704b
║   3006 - Linked Track ID      ║  2
║   3001 - Sample Rate          ║  24/1
║   3002 - Container Duration   ║  1
║   3004 - Essence Container    ║  060e2b34.04010107.0d010301.020c0100 (JPEG2000 Picture Element - Frame Wrapped)
║   320C - Frame Layout         ║  FULL_FRAME (0)
║   3203 - Stored Width         ║  4096
║   3202 - Stored Height        ║  2160
║   320E - Aspect Ratio         ║  4096/2160 (1.90)
║   3201 - Picture Essence Coding  ║  JPEG 2000 4K Digital Cinema Profile (060e2b3404010109040102020301010 4)
║   3406 - Component Max Ref    ║  4095
║   3407 - Component Min Ref    ║  0
║   320D - Video Line Map       ║  2 item(s): 00000000, 00000000
║   3401 - PixelLayout          ║  8 item(s):
║                               ║  - X' - DCDM X color component (SMPTE 428-1 X') - 12 bytes
║                               ║  - Y' - DCDM Y color component (SMPTE 428-1 Y') - 12 bytes
║                               ║  - Z' - DCDM Z color component (SMPTE 428-1 Z') - 12 bytes
║                               ║  - Termination - 0 bytes
║                               ║  - Termination - 0 bytes
║                               ║  - Termination - 0 bytes
║                               ║  - Termination - 0 bytes
║                               ╚  -  Termination - 0 bytes
╚═══════════════════════════════════════════════════════════════════════════════
```

SubDescriptors fait référence au KLV **JPEG2000 Picture Sub-Descriptor** (ci-dessous)

PixelLayout va décrire comment sont stockés les pixels et leurs formats. Dans notre cas, nous aurons toujours un X'Y'Z' en 12 bits. Si l'encodeur n'a pas pu déterminer la valeur de **PixelLayout**, il va mettre des 0 partout.

Essence Container indique le type de container pour l'essence avec son Universal Label :

Type de MXF	Universal Label	Type de container
MXF Picture	`060e2b34.04010107.0d010301.020c0100`	**JPEG2000 Picture Element - Frame Wrapped**.
MXF Sound	`060e2b34.04010101.0d010301.02060100`	**Broadcast Wave audio - Frame-Based mapping**
MXF Subtitle	`060e2b34.0401010a.0d010301.02130101`	**TimedText Essence Container**
MXF Immersive Audio (IAB)	`060e2b34.04010105.0e090605.00000000`	**Immersive Audio Data Essence Container**

Il existe d'autres type de container, pour notre cas (JPEG2000), notre UL sera celui de **JPEG2000 Picture Element**

Frame Layout va déterminer si c'est du progressif ou de l'entrelacé (en gros), il y a plusieurs codes :

Code	Description
0	Full Frame
1	Separate Fields
2	Singe Field
3	Mixed Fields
4	Segmented Frame

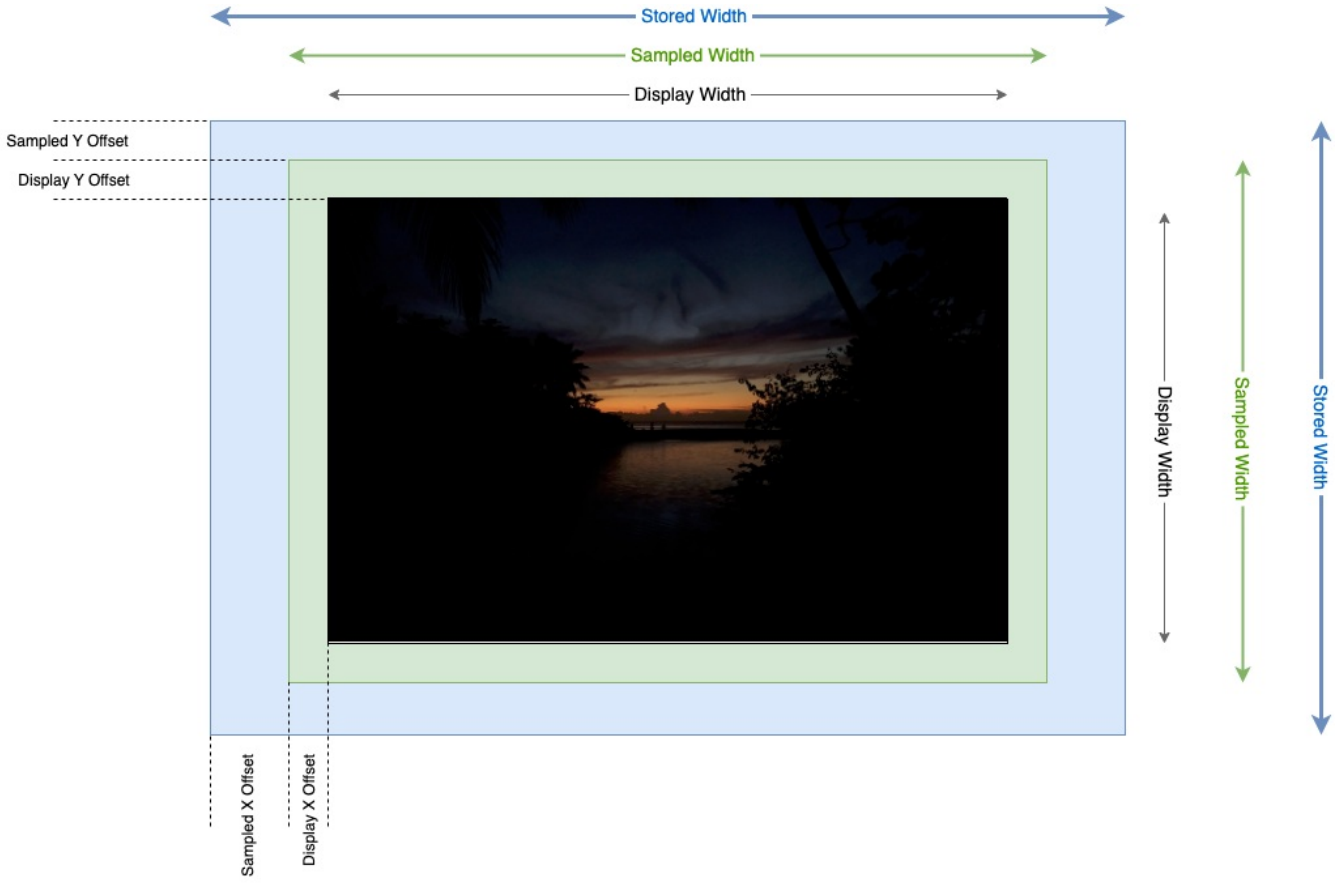
On sera toujours sur du Full Frame.

Stored vs **Sampled** vs **Display** sont 3 possibilités de gestion et d'affichage de l'image.

- **Stored** va indiquer la taille de l'image stockée, avec toutes les informations possibles, même qui ne devraient pas être projetés.
- **Sampled** va inclure l'image et les données auxiliaires (même métadonnées)
- **Display** est l'image qui va être affichée sur l'écran et juste l'image en supprimant tout le reste qui pourrait parasiter la projection.

Dans un MXF DCP, toutes les items **Sampled** et **Display** seront absents : seules seront définies **Stored** qui correspondront donc aux valeurs utilisées pour l'affichage à l'écran.

Voici un récapitulatif entre **Stored**, **Sampled**, **Display** et les différents **Offset** :

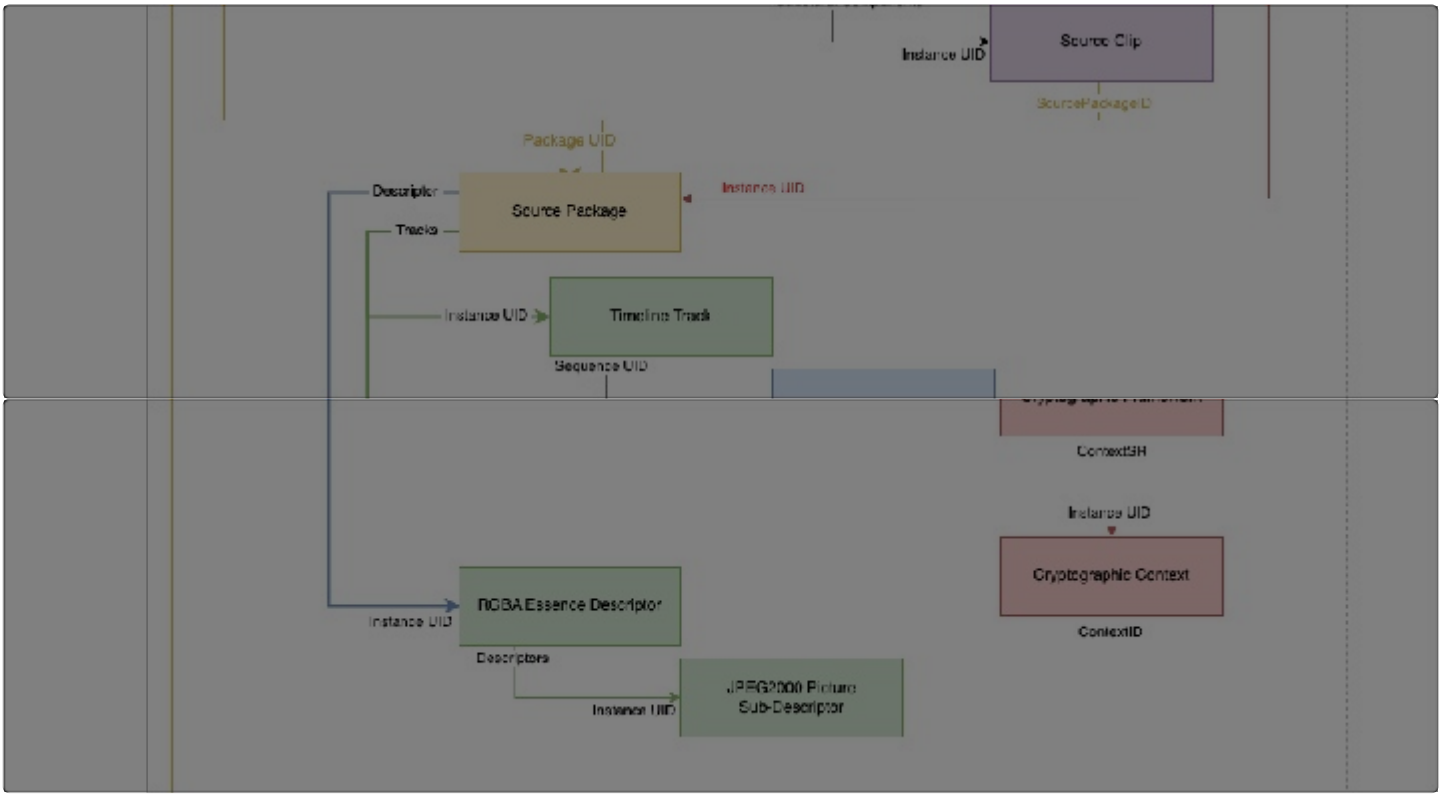


Component Max Ref est la valeur maximale qu'on pourra avoir dans l'encodage RGB. Ici, nous aurons toujours 4095 car 12 bits et que

2^{12} = 4096). Mais alors pourquoi 4095 et pas 4096: parce qu'on compte à partir de 0 donc il y aura 4096 valeurs allant de 0 à 4095 :)

Component Min Ref est la valeur minimale pour l'encodage RGB. Pour nous, nous partirons toujours de 0.

Linked Track ID est le **Track ID** défini dans l'une des **Track** du Package (ici Source Package) qui est défini dans le **Descriptor** de ce dernier :



Rapidement pour les éléments comme **DisplayF2Offset** et **StoredF2Offset** : vous ne les trouverez jamais dans un MXF DCP car lorsque le FrameLayout est à FULL_FRAME, ces éléments n'ont aucun intêret et ne doivent donc pas être présent.

Picture Essence Coding est un identifiant Universal Label permettant d'identifier le type d'encodage et/ou de compression de l'essence :

Type	Universal Label
JPEG 2000 2K Digital Cinema Profile	`060e2b3404010109040102020301010 3`
JPEG 2000 4K Digital Cinema Profile	`060e2b3404010109040102020301010 4`
JPEG 2000 Undefined Digital Cinema Profile	`060e2b340401010a040102020301017f`

Selon le type de contenu (2K ou 4K), vous aurez l'un ou l'autre. Le dernier est à utiliser sur du JPEG2000 pas encore normé (du 8K ? :) Vous en trouverez d'autres dans ce fichier.

Video Line Map indique la première ligne active de chaque champ de l'image. Sa valeur par défaut sera 0. Cet item est surtout utile quand le **FrameLayout** est différent de `FULL_FRAME`, c'est-à-dire si vous avez des images entrelacées. Vu que nous sommes en progressif, vous aurez (normalement) toujours 0. (il m'est arrivé de découvrir dans certains MXF produits et diffusés une valeur à (2, 4) avec pourtant un FrameLayout en `FULL_FRAME`)

STRUCTURES DES DONNÉES

Voici la structure complète d'un **RGBA Essence Descriptor**, énormément d'items sont peu ou pas utilisés dans un MXF DCP :

Local Tag	Nom de l'attribut	Type	Taille	Fixe/Variable SMPTE	Obligatoire
3C0A	Instance ID	UUID	16 octets	Fixe	**Obligatoire**
0102	Generation UID	UUID	16 octets	Fixe	Optionnel
0101	Object Class	AUID	16 octets	Fixe	Optionnel
dyn	ApplicationPlug-In Batch	Array[UL]	8+16n	Variable	Optionnel
2F01	Locators	Array[UL]	8+16n	Variable	Optionnel
dyn	SubDescriptors	Array[UID]	8+16n	Variable	Optionnel

3006	Linked Track ID	Uint32 (→TrackID)	4 octets	-	Optionnel
3001	Sample Rate	Rational	8 octets	-	**Obligatoire**
3002	Container Duration	Length	8 octets	-	Optionnel
3004	Essence Container	UL	16 octets	-	**Obligatoire**
3005	Codec	UL	16 octets	-	Optionnel
3215	Signal Standard	Enum	1 octet	-	Optionnel
320C	Frame Layout	UInt8	1 octet	-	**Best Effort** [1]
3203	Stored Width	UInt32	4 octets	-	**Best Effort**
3202	Stored Height	UInt32	4 octets	-	**Best Effort**
3216	StoredF2Offset	Int32	4 octets	-	Optionnel
3205	Sampled Width	UInt32	4 octets	-	Optionnel
3204	Sampled Height	UInt32	4 octets	-	Optionnel
3206	SampledXOffset	Int32	4 octets	-	Optionnel
3207	SampledYOffset	Int32	4 octets	-	Optionnel
3208	DisplayHeight	UInt32	4 octets	-	Optionnel
3209	DisplayWidth	UInt32	4 octets	-	Optionnel
320A	DisplayXOffset	Int32	4 octets	-	Optionnel
320B	DisplayYOffset	Int32	4 octets	-	Optionnel
3217	DisplayF2Offset	Int32	4 octets	-	Optionnel
320E	Aspect Ratio	Rational	8 octets	-	**Best Effort**
3218	Active Format Descriptor	UInt8	1 octet	-	Optionnel
320D	Video Line Map	Array[Int32]	8 + 8 octets	-	**Best Effort**
320F	Alpha Transparency	UInt8	1 octet	-	Optionnel
3210	Transfer Characteristic	UL	16 octets	-	Optionnel
3211	Image Alignment Offset	UInt32	4 octets	-	Optionnel
3213	Image Start Offset	UInt32	4 octets	-	Optionnel
3214	Image End Offset	UInt32	4 octets	-	Optionnel
3212	FieldDominance	UInt8	1 octet	-	Optionnel
3201	Picture Essence Coding	UL	16 octets	-	**Decoder Required**
321A	Coding Equations	UL	16 octets	-	Optionnel
3219	Color Primaries	UL	16 octets	-	Optionnel
3406	Component Max Ref	UInt32	4 octets	Fixe	Optionnel
3407	Component Min Ref	UInt32	4 octets	Fixe	Optionnel
3408	Alpha Max Ref	UInt32	4 octets	Fixe	Optionnel
3409	Alpha Min Ref	UInt32	4 octets	Fixe	Optionnel
3405	ScanningDirection	Orientation	1 octet	Fixe	Optionnel
3401	PixelLayout	RGBALayout	8 x (1 + 1) = 16 octets	Fixe	**Best Effort**
3403	Palette	DataValue	Variable	Variable	Optionnel
3404	PaletteLayout	Array[RGBALayout]	Variable	Variable	Optionnel
????	Gamma	UL	16 octets	Fixe	Optionnel ?

Les éléments en gris ne sont pas utilisés la plupart du temps mais ils peuvent l'être dans certains contextes. N'oubliez pas que un **RGBA Essence Descriptor** dépend d'une hiérarchie de type, il y aura donc énormément d'éléments utiles pour d'autres utilisations de MXF (pas forcément que pour un DCP), il est donc normal d'avoir des pans entiers d'items en gris.

Le document faisant référence sera **SMPTE 429-4 - DCP - MXF JPEG2000 Application** qui va donner toutes les items et leurs valeurs par défaut.

Notez que certains éléments considérés comme **Best Effort** (donc requis) peuvent ne pas exister. Ne me demandez pas pourquoi : pourtant, la norme les impose mais sur énormément de MXF DCP, des éléments - comme **Video Line Map** ou **PixelLayout** par exemple - peuvent ne pas être présents.

Pour le type **RGBALayout** de l'item **PixelLayout**, c'est un encodage spécifique, c'est 8 éléments de 2 octets chacun. Pour chaque élément, le premier octet est un code qui va donner le type (exemple R, G, B, X, Y, Z, etc...) et le second octet la DepthColor. Si c'est entre 1 à 32, ça sera juste de 1 à 32 octets :) - sinon si le code est :

- 253 : un DepthColor HALF - Floating Point 16 bits
- 254 : un DepthColor IEEE - Floating Point 32 bits
- 255 : un DepthColor IEEE - Floating Point 64 bits

Dans nos MXF DCP, nous n'aurons que des codes 0xD8, 0xD9 et 0xDA qui réprésente un X', Y', Z' (voir chapitre XYZ)

Pour le premier octet, voici la liste des codes (les 3/4, c'est un simple code ASCII, sauf qu'ils ont rajouté d'autres éléments qui ne le sont pas) :

Code Hexa	Description
`0`	Terminaison (aucun élément)
`52`	R - Red component
`47`	G - Green component
`42`	B - Blue component
`41`	A - Alpha component
`72`	r - Red component (LSB)
`67`	g - Green component (LSB)
`62`	b - Blue component (LSB)
`61`	a - Alpha component (LSB)
`46`	F - Fill component
`50`	P - Palette code
`55`	U - Color Difference Sample (U, Cb, I, ..)
`56`	V - Color Difference Sample (V, Cr, Q, ..)
`57`	W - Composite Video
`58`	X - Non co-sited luma component
`59`	Y - Luma component
`5a`	Z - Depth component (SMPTE ST 268)
`75`	u - Color Difference Sample (U, Cb, I, ..) (LSB)
`76`	v - Color Difference Sample (V, Cr, Q, ..) (LSB)
`77`	w - Composite Video (LSBs)
`78`	x - Non co-sited luma component (LSB)
`79`	y - Luma component (LSBs)
`7a`	z - Depth component (LSBs) (SMPTE 268)
`d8`	X' - DCDM X color component (SMPTE 428-1 X’)
`d9`	Y' - DCDM Y color component (SMPTE 428-1 Y’)
`da`	Z' - DCDM Z color component (SMPTE 428-1 Z’)

Note: Dans la norme **SMPTE 429-4 - MXF JPEG2000 Application**, il est indiqué qu'un item **Gamma** existe, cependant il n'existe aucune référence dans la norme SMPTE 377-1 (MXF - File Format Specification). Il se peut que l'item désigné par **Gamma** se cache peut-être derrière l'item **Color Primaries**, ou **Coding Equations** ou bien **Transfer Characteristic**. Cependant, sur l'analyse de plusieurs MXF DCP, je n'ai constaté aucun de ces items.

HIÉRARCHIE DU FORMAT

Interchange Object → Generic Descriptor → File Descriptor → Generic Picture Essence Descriptor → RGBA Picture Essence Descriptor

CODE : LIRE CE KLV

Code source pour lire les headers :

```
import sys
import io

# 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:

    while True:

        # Key : Universal Label
        key = file.read(16)

        # End of file
        if not key:
            break

        # Length (BER format)
        length = to_int(file.read(4)[1:])  # BER format - read last 3 bytes

        # Value
        value = file.read(length)

        # Filter by KLV : RGBA Essence Descriptor
        if key.hex() != "060e2b34025301010d01010101012900":
           continue

        # Show each KLV
        print("{key} - {length:>6d} bytes - {value}...".format(
            key    = key.hex(),
            length = length,
            value  = value[0:16].hex()
        ))

        # read each item
        data = io.BytesIO(value)

        # "item" is a baby KLV:
        #  localtag (key)  :  2 bytes
        #  item length     :  2 bytes (no BER format)
        #  item value      :  variable bytes

        while True:

            # get local tag (2 bytes)
            localtag = data.read(2)

            if not localtag:
                break

            # get item length (2 bytes, directly int, NOT Ber format)
            item_length = to_int(data.read(2))

            # get item value
            item_value = data.read(item_length)

            # Show each item
            print("{localtag} : {item_length:>2d} : {item_value}".format(
                localtag    = localtag.hex(),
                item_length = item_length,
                item_value  = item_value.hex()
            ))
```

Voici son exécution sur le MXF 2D.mxf :

```
$ ./mxf-klv-picture-rgba-essence-descriptor.py "2D.mxf"

# KLV RGBA Essence Descriptor
060e2b34025301010d01010101012900 -    189 bytes - 3c0a00102cb46505be6b41c3abbe8848...
3c0a : 16 : 2cb46505be6b41c3abbe8848d06e21c9
ffff : 24 : 000000010000000106054268f8fdf47ba98db71675d3f704b
3006 :  4 : 00000002
3001 :  8 : 0000001800000001
3002 :  8 : 0000000000000001
3004 : 16 : 060e2b34040101070d010301020c0100
320c :  1 : 00
3203 :  4 : 00001000
3202 :  4 : 00000870
320e :  8 : 0000100000000870
3201 : 16 : 060e2b3404010109040102020301010 4
3406 :  4 : 00000fff
3407 :  4 : 00000000
3401 : 16 : 00000000000000000000000000000000
```

DUPLICATE JPEG2000 Picture SubDescriptor

On voit qu'avec quelques lignes, on récupère l'ensemble des informations brutes :

- **Colonne 1** : le Local Tag - la clef spécifiant le type de valeur (comme pour la Key d'un KLV)
- **Colonne 2** : la taille en octets de sa value.
- **Colonne 3** : la valeur brute, sans aucune conversion.

Vous devrez maintenant convertir chaque valeur brute dans un format qui lui est propre. Et pour cela, il faudra vous baser sur le **Local Tag** en colonne 1.

Notez que pour les **Local Tag**, vous devrez lire le KLV Primer Pack afin de faire correspondre chaque **Local Tag** avec sa véritable désignation (Universal Label, Type, Taille de chaque sous-élément, Formatage, ..). Plus d'informations dans le chapitre Local Tag.

Même sans convertir, nous pouvons déjà identifier quelques éléments, comme des **Universal Label**.

Prenons un exemple, le `320E` a une taille de 8 octets. Si on regarde sa désignation, c'est un **Local Tag Statique** représentant un **Aspect Ratio** et de type "rational", cela veut dire de format "num/num". La valeur de 8 octets (`0000100000000870`) doit être séparé en deux: 2 x 4 octets : `00001000` et `00000870` . Les deux valeurs convertient, nous avons `4096` et `2160` . Nous aurons donc un "4096/2160" qui correspond à l'Aspect Ratio de notre film (1.90 arrondi).

Vous devrez donc, pour chaque valeur, faire correspondre son type et son formatage. En lisant chaque docs SMPTE, on découvre ces petits pokemons d'informations. Fort heureusement, une partie est compilée dans le chapitre Local Tag ou bien dans mxf-analyzer.

LA SUITE...

Le KLV supplémentaire est le JPEG2000 Picture Sub-Descriptor

NOTES

1. En dehors des attributs classiques comme **Req** (Obligatoire) et **Opt** (Optionnel), les attributs particuliers comme **Best-Effort**, **E/Req** et **D/Req** représentent un statut particulier défini par ce tableau : ↩

Abréviation	Nom complet	Encodeur MXF	Decodeur MXF	Considéré comme (SMPTE 395)
Req	Required / Obligatoire	Shall encode	Shall decode	**Required**
E/Req	Encoder Required	Shall encode	May decode	**Required**
D/Req	Decoder Required	May encode	Shall decode	Optional
B.Effort	Best Effort	Shall encode	Should decode	**Required**
Opt	Optional / Optionnel	May encode	May decode	Optional

   Les valeurs pour **Optional** et **Decoder Required** ne doivent être encodées que si leurs valeurs exactes sont connues par l'application qui va créer ou modifier le MXF. Si ce n'est pas le cas, les items et leurs valeurs ne doivent pas être encodés dans le MXF.

   Le valeurs pour **Best Effort** doivent toujours être encodées. Le **Best Effort** indique que l'encodeur peut ne pas avoir l'information. Si l'encodeur ne sait pas au moment de la création, il doit utiliser une valeur par défaut dite "Distinguished Value" qui est indiquée dans la norme.