KLV : VALUE : LES DONNÉES UTILES

Convention de nommage

Pour une meilleure compréhension, dans toute la documentation, nous nommerons respectivement la clef, la taille et la valeur d'un item par **item.key**, **item.length** et **item.value**; Et pour la clef, la taille et la valeur d'un KLV resteront **Key**, **Length** et **Value**.

Les données ont une représentation, elles ne sont pas simplement posées comme cela dans la partie **Value**. Elles doivent suivre certaines règles.

Chaque règle est comme **un modèle** (dans le sens "template") : il existe plusieurs modèles, certains sont très simples et d'autres plus complexes.

Plus imagé, voyez cela comme les petits casiers qu'on place dans les tiroirs pour ranger notre bordel :

Et bien, la structure de **Value** sera identique selon le petit casier-modèle utilisé :)

LES FAMILLES DE MODÈLES

Tout d'abord, il faut savoir que les modèles sont classés dans deux grandes familles :

- **Des données brutes**, sans aucune structure. (surnommé **Data Items**) : il a 1 modèle.
- **Des données structurées** et respectant des règles de représentation. (surnommé **Data Groups**), il a 3 modèles.

Pour le premier, il n'en existe qu'un seul modèle, lui-même, le **Data Item**. Pour le second, il existe 3 modèles :

- Local Sets
- Variable-Length Pack
- Defined-Length Pack (parfois appelé Fixed-Length Pack)

Nous avons donc **quatre modèles** pour le "rangement" des données :

- Le 1er modèle (**Data Item**) est le simple, ce sont les données brutes qui sont placées directement dans la partie **Value** sans aucun traitement.
- Pour les autres modèles de la famille **Data Groups**, les données sont rangées dans des petites cases et il faudra suivre le modèle pour comprendre comment ils ont été placés.

Astuce

Comment savoir rapidement si les données sont de types **Data Item** ou **Data Group** sans avoir lu les centaines de pages de documentations ?

C'est assez simple : si vous vous souvenez, dans la partie Key, nous évoquions rapidement le **Category Designator** qui peut être soit à **Dictionaries (0x01)** ou **Groups (Sets/Packs) (0x02)**.

Et bien :

- **Dictionaries** désigne des **données brutes**,

- **Groups (Sets/Packs)** désigne des **données structurées**

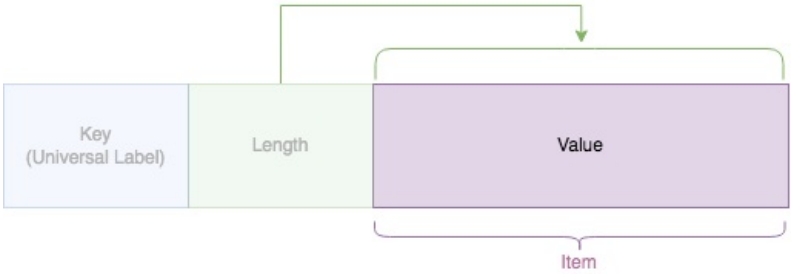
Voici un résumé :

```
Category Designator 0x01 = Dictionaries        = Data Items  = Données brutes
Category Designator 0x02 = Groups (Sets/Packs) = Data Groups = Données structurées
```

Si ce n'est pas très clair pour l'instant, ne vous inquiétez pas, cela va venir avec le temps.

Revenons plus en détails sur chaque type de données en commençant par le plus simple : **Data Items**

LES DONNÉES BRUTES : DATA ITEMS OU DICTIONARY



Peu de chose à dire, il suffit de lire les données brutes directement. C'est un peu l'équivalent avec ce casier :

C'est le type le plus simple: il suffit de lire le **Length** pour déterminer la taille totale de **Value** et de récupérer les données brutes.

Ces données peuvent être une frame JPEG2000 non-chiffrée (vous verrez les entêtes du J2C dès les premiers octets de **Value**), les pistes sonores au format WAV non-chiffrées ou bien d'autres encores.

Dès que vous voyez ce type de KLV, il n'y aura aucun traitement particulier à effectuer hormis extraire la donnée et - suivant le format - le traiter par son codec habituel. Par exemple, pour du JPEG2000, vous avez la librairie libre OpenJPEG ou la librairie propriétaire Kakadu pour ne citer que les plus connus.

Key aura donc son **Category Designator** à `0x01` (Data Items - Dictionary)

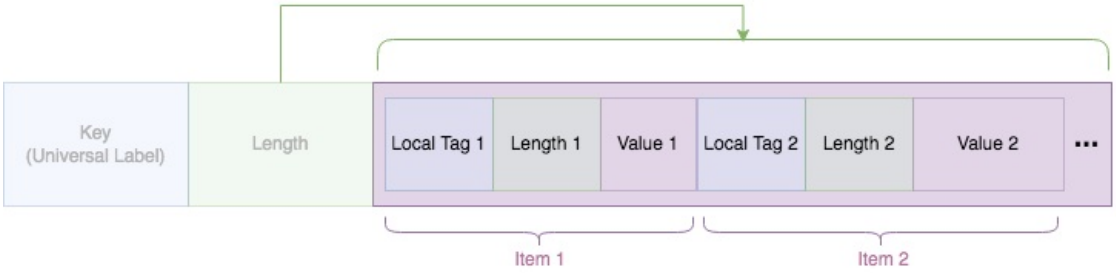
LES DONNÉES STRUCTURÉES : DATA GROUPS OU SETS / PACKS

Les données structurées sont des données rangées en respectant certaines règles. Certaines règles permettent aux données d'être très libres - dans leurs emplacements, leurs tailles - et d'autres plus restrictives qui imposent des emplacements stricts ou la présence obligatoire de certaines données.

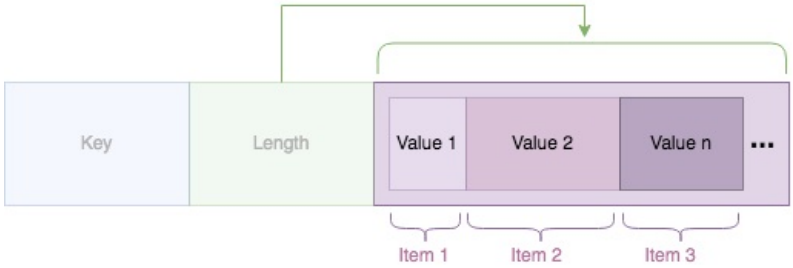
Nos équivalents casiers seraient ceux-là :

Voici les **trois** modèles normalisés pour les données structurées :

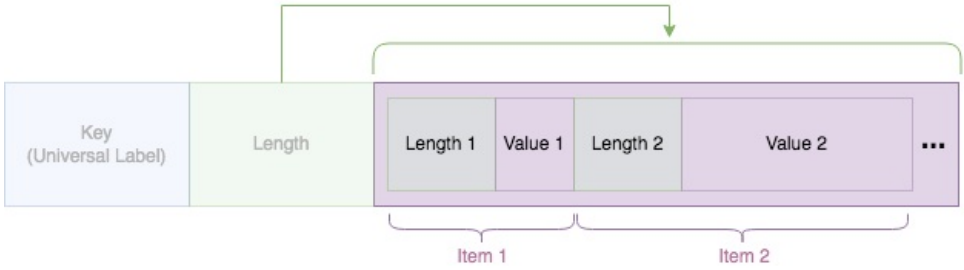
- **Local Sets** : Ils ont un champ pour une petite clef (**item.key**) appelé **Local Tag**, un champ pour la taille des données (**item.length**) et un champ pour les données en eux-même (**item.value**) : oui, ce sont des KLV à l'intérieur de KLV, je les appelle des **bébés KLV** :



- **Defined-Length Pack** : Ce sont des données fixes avec une taille précise. Elles n'ont ni item.key ni item.length - seulement les **item.value** placées l'une à la suite de l'autre :



- **Variable-Length Pack** : Les données variables - sans item.key - avec seulement **item.length** et **item.value** :

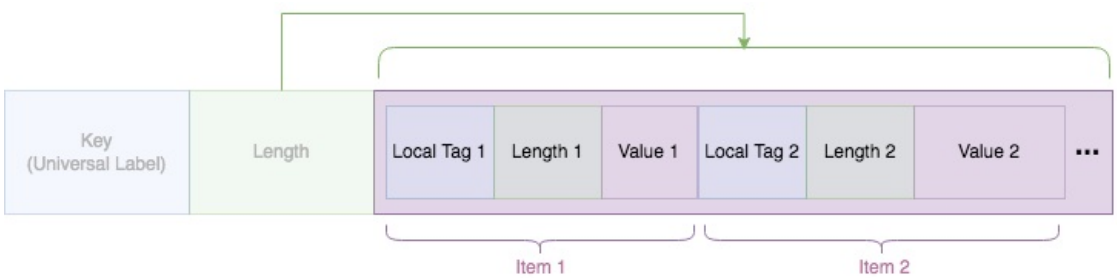


Voila les trois modèles pour les données structurées.

Key aura donc son **Category Designator** à `0x02` (Data Groups - Sets/Packs)

Voyons en détail chaque modèle des données structurées.

LES DONNÉES STRUCTURÉES : LOCAL SETS : LES BÉBÉS KLV



Category Designator	`0x02` **(Data Groups - Sets/Packs)**
Registry Designator	`0x53` (Local Sets) [1]
Exemple de KLV	Preface, Identification, Content Storage, Material Package, Timeline Track, Sequence, Index Table Segment

Ce modèle est appelé **Local Sets**.

Le **Local Sets** stocke une multitude de petit KLV - que je surnomme des bébés KLV - de leurs vrais noms **Items**.

Chaque item respecte les règles des KLV que nous avons déjà vu mais avec quelques modifications :

- Leurs clefs (**item.key**) sont nommées **Local Tags** et sont encodés sur 2 octets (16 bits)
- Les tailles (**item.length**) sont encodées sur 2 octets (16 bits)
- Les données (**item.value**) ... sont les données brutes.

Quelques règles sur ces items :

- Il peut y avoir **un ou plusieurs items**
- Il n'y a **aucune obligation de présence** de tel ou tels items : certains peuvent être présents et d'autres absents, tant qu'il y a une clef pour valider la présence ou non de tels types de données, tout va bien.
- Ils peuvent être dans **n'importe quel ordre** : chaque item peut être placé dans n'importe quel ordre dans **Value**, ne considérez donc pas qu'un item en particulier sera toujours au début de **Value**, car vous l'avez toujours vu au début, c'est probablement parce que l'encodeur MXF a été écrit pour placer cet item à cet endroit, mais un autre encodeur peut le placer à un autre endroit.

Si vous voulez en savoir plus sur le principe des **Local Tags**

La hiérarchie des Local Sets

Les différents **Local Sets** sont régies par une hiérarchie, tous dépendent du format de base **Interchange Object** qui va définir les items de base, leurs règles (obligatoires ou non, leurs tailles, leurs types). Par exemple, cette base définie qu'il y aura un **Instance UID** obligatoirement, c'est pourquoi vous verrez toujours un **Instance UID** dans tous les KLV de type **Local Sets**.

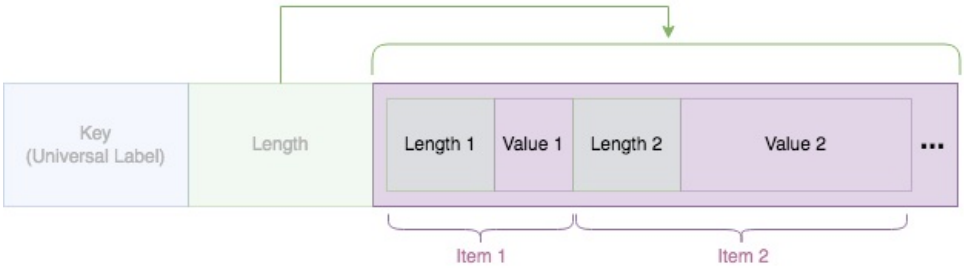
Puis, chaque type de KLV (ex. Timeline Track) aura son propre format qui va ajouter des attributs, des nouveaux items, etc...

D'autres types de KLV pourront utiliser un autre format comme base et rajouter ses propres attributs, comme une pile d'assiette.

Exemple avec le KLV **Descriptive Metadata Segment** qui a son propre format (DMSegment) mais qui va dépendre en début du format **Comment Marker** qui va lui-même dépendre du format **Event** qui va dépendre du format **Segment** qui va dépendre du format **Structural Component** et enfin qui dépend bien évidemment du format **Interchange Object**. À la fin, vous aurez une structure de données très complexe, car chaque format va rajouter ses propres items.

Tous les formats sont définis dans l'annexe du SMPTE 377-1 - MXF - File Format Specification. Ce seront des gros tableaux avec plein d'informations dedans, vous pourrez pas les louper ;-)

LES DONNÉES STRUCTURÉES : VARIABLE-LENGTH PACK : LES DONNÉES SANS KEY MAIS AVEC LENGTH ET VALUE



Category Designator	`0x02` **(Data Groups - Sets/Packs)**
Registry Designator	`0x04` (Variable-Length Pack) [2]
Exemple de KLV	Encrypted Essence

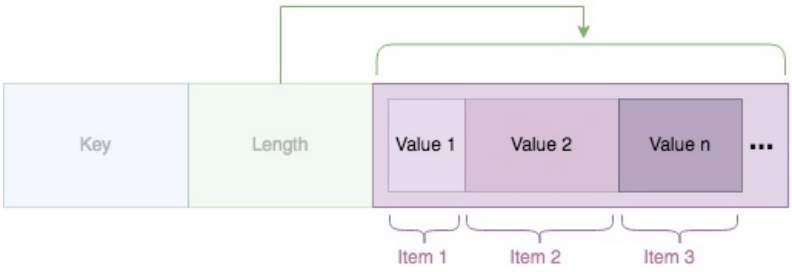
Ce type reprend toutes les caractéristiques de **Local Sets** mais sans les clefs. Tous les items seront définis dans un certain ordre et suivant des critères définis dans la norme.

Dans Variable-Length Pack, vous aurez donc un **Length** puis un **Value**, ainsi de suite.

Normalement, vous ne pourrez définir précisement où se trouve tel ou tels items sans avoir lu le premier **item.length** qui permettra de déterminer la première **item.value**, qui déterminera ainsi le second **item.length** et ainsi de suite.

Notez que c'est la théorie parfaite. Il arrive que pour certains types de KLV, les normes définissent directement la taille des **item.length** et des **item.value** (ce qui rend inutile les **item.length**...).

LES DONNÉES STRUCTURÉES : DEFINED-LENGTH PACK : LES DONNÉES FIXES SANS KEY NI LENGTH.



Category Designator	`0x02` **(Data Groups - Sets/Packs)**
Registry Designator	`0x05` (Defined-Length Pack)
Exemple	Partition Pack (Body, Footer, Header), Primer Pack, Random Index Pack

Ce type de KLV est très simple : les items ont chacun un emplacement et une taille fixe. Il est donc assez simple d'aller chercher une donnée en particulier à un emplacement particulier.

A contrario, il est impossible de déroger à une valeur. Si elle est inutile, elle devra être complétée avec du vide pour respecter la structure. On se retrouve donc parfois avec une suite de zéros inutiles.

Ce type de KLV est parfois appelé **Fixed-Length Pack** dans certaines documentations SMPTE.

LES DONNÉES STRUCTURÉES : FUCKED PACK : LES DONNÉES SEMI-FIXES / SEMI-VARIABLES

Surprise !

Ce n'est pas un terme officiel, c'est le surnom que j'ai donné à ce type de données structurées :-)

Ce sont des structures de données particulières : elles respectent soit une partie des **Local Sets**, soit des **Variable-Length Pack** soit des **Fixed-Length Pack** ... mais pas tout à fait : Certaines débutent avec des **Local Tags** mais ommettent la partie Length et passent directement à la **Data** avec une taille fixe; et d'autres ont juste les **Lengths** sans les **Local Tags** et les données ont une taille variable et d'autres encore ont des **Lengths** qu'ils n'utilisent pas ou bien respectent une partie du **Variable-Length Pack** mais d'un coup ajoute un bloc de **Fixed-Length Pack** (ou inversement).

Je mettrais par exemple, le KLV **Encrypted Essence Container** dans cette section, même si au niveau de la norme, cela n'est pas exact : il est considéré comme un **Variable-Length Pack**. J'expliquerai pourquoi dans les différentes parties des KLV respectifs (un exemple avec Encrypted Essence Container).

CONCLUSION

Voici un tableau récapitulatif des différents modèles pour la **Value** d'un KLV :

Data Items	-	Données brutes	item.value
Data Groups	Local Sets	KLV	item.key + item.length + item.value
	Variable-Length Pack	Données à tailles variables	item.length + item.value
	Fixed-Length Pack	Données à tailles fixées	item.value

Voyons maintenant les différents types de KLV

NOTES

1. Dans un MXF, nous avons plusieurs **Local Set** avec des valeurs de **Registry Designator** comme `0x03`, `0x13` et `0x53`, mais nous n'utiliserons que `0x53`. Voir KLV: Key: L'identifiant de type pour plus d'informations ↩
2. Variable entre 1 octet et 4 octets mais pour notre cas, c'est un BER. Cela est déterminé par la valeur du **Registry Designator**. ↩