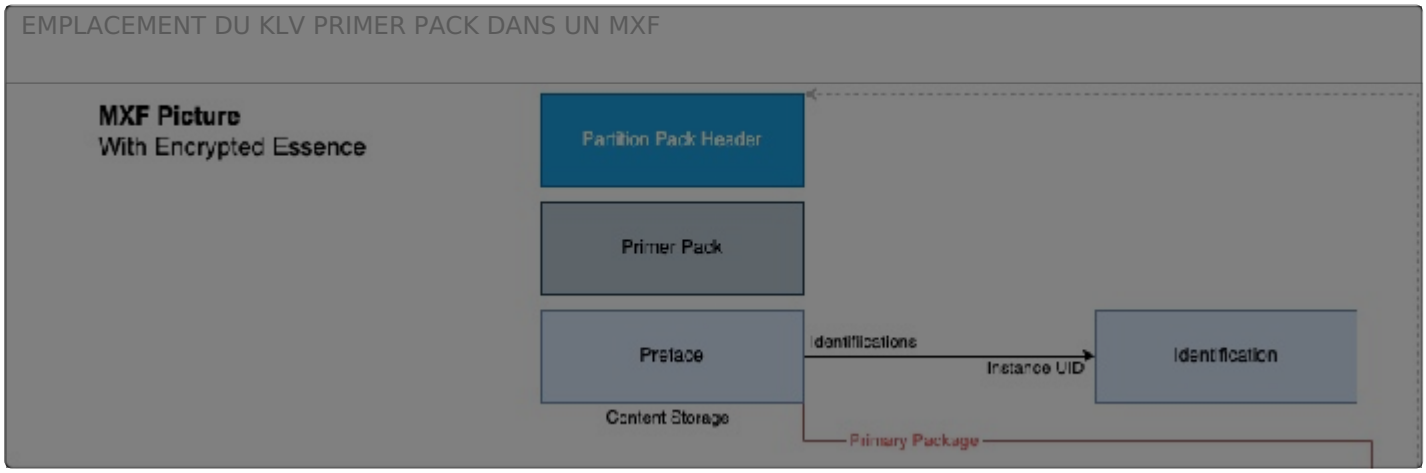
PRIMER PACK

Références	SMPTE 377-1-2011 - MXF - File Format Specification Chapitre 9.2 - Primer Pack [P54]
Modèle KLV	Fixed-Length Pack
Universal Label	`06.0e.2b.34.02.05.01.01.0d.01.02.01.01.05.01.00 (SMPTE)`

PRÉFACE



Primer Pack est un des **KLV Headers** d'un MXF de DCP qui se situe en général après le KLV **Partition Pack Header**

Primer Pack stocke une succession de **Local Tag** et d'**Universal Label**.

> Je vous conseille fortement la lecture du chapitre sur **Local Tag** sinon vous ne comprendrez pas l'importance du **Primer Pack**

Pour être plus précis, **Primer Pack** fournit une correspondance entre différents **Local Tag** (2 octets) - utilisés dans les KLV **Local Set** - et un identifiant unique en 16 octets - qui peut être un **UUID** ou un **Universal Label** - surnommé **AUID** [1].

AUID kézako ?

Officiellement, l'identifiant long est appelé **AUID** (SMPTE adore les petits surnoms pour bien nous perdre).

Un AUID est juste un surnom donné pour dire que l'identifiant long sera soit un **UUID** soit un **Universal Label**, rien de plus.

Pour vous donner un exemple imagé, c'est comme si nous utilisions le terme "Véhicule" (AUID) au lieu de spécifier si c'est une Voiture Bleue (UUID) ou une Voiture Rouge (Universal Label). Dans les deux cas, cela reste une voiture, c'est juste la couleur qui change :)

Ne vous prenez pas trop la tête là-dessus, considérez qu'on n'utilisera et ne parlera que du format **Universal Label** : dans les MXF DCP, nous n'avons que des **Universal Label** au lieu d'**UUID**.

Voyez le KLV **Primer Pack** comme une table de correspondances entre identifiant court **Local Tag** et identifiant long **Universal Label**. Les **Local Tag** seront utilisés dans les autres KLV: Au lieu d'avoir un long identifiant, on en a un plus petit.

Un exemple de correspondance :

```
1102 (Local Tag)  =>  060e2b34.01010102.06010103.02000000 (Universal Label)
```

Le **Local Tag** `1102` sera un alias de l'**Universal Label** `060e2b34.01010102.06010103.02000000` qui est l'identifiant pour le label **SourceTrackID**. Donc, si dans les KLV, nous trouvons un identifiant `1102`, nous savons alors que la donnée juste après sera un **SourceTrackID**.

Dans **Primer Pack**, nous aurons donc des centaines d'alias de ce genre pouvant faire le lien entre **Local Tag** et **Universal Label**.

DESCRIPTION

Chaque bloc comprenant le petit identifiant (Local Tag) et son correspondant en long identifiant (Universal Label) est appelé **Item**, il y en aura plusieurs mis bout à bout dans le KLV. Concrètement, si on lit chaque item, nous aurons un rendu de données de la sorte :

```
0201    060e2b34.01010102.04070100.00000000
0202    060e2b34.01010102.07020201.01030000
0602    060e2b34.01010102.05300404.01000000
1001    060e2b34.01010102.06010104.06090000
1101    060e2b34.01010102.06010103.01000000
1102    060e2b34.01010102.06010103.02000000
1201    060e2b34.01010102.07020103.01040000
1501    060e2b34.01010102.07020103.01050000
1502    060e2b34.01010102.04040101.02060000
1503    060e2b34.01010101.04040101.05000000
1901    060e2b34.01010102.06010104.05010000
FFF8    060e2b34.0101010a.04010603.01000000
FFF9    060e2b34.01010109.06010104.06100000
FFFA    060e2b34.01010109.02090301.02000000
FFFB    060e2b34.01010109.02090302.01000000
FFFC    060e2b34.01010109.02090301.01000000
FFFD    060e2b34.01010109.06010102.02000000
FFFE    060e2b34.01010109.01011511.00000000
FFFF    060e2b34.01010109.06010104.020d0000
```

La première fois qu'on étudie ce KLV, on se demande à quoi sert ce truc hormis de lister tous les Local Tag utilisés dans le MXF. Au premier abord, on ne voit pas trop l'utilité.

Et puis au fur et à mesure, on commence à comprendre qu'il peut être utile pour deux éléments majeurs :

- Le principal est celle des **Local Tag Dynamiques** et de pouvoir faire un mapping entre un **Local Tag Dynamique** et un **Universal Label** qui va représenter quelque chose derrière,
- Le second est - si un encodeur souhaite rajouter ou modifier des KLV - de pouvoir connaître les **Local Tag Dynamiques** déjà utilisés rapidement.

Pour faire un rappel de ce qui a été déjà écrit auparavant :

- Un **Local Tag** est un identifiant utilisé comme clef dans un item dans un KLV de type **Local Set**.
- Un **Local Tag** permet d'identifier le nom et le type de données qui sera dans un **item.value**.
- Les **Local Tag Statiques** (`0x0100` - `0x7FFF`) sont réservées et définies dans les documentations SMPTE. Ils ne bougeront pas d'un MXF à l'autre, ils désigneront toujours le même item. Par exemple, un attribut **Instance UID** aura toujours la valeur `3C0A` comme **Local Tag**.
- Les **Local Tag Dynamiques** (`0x8000` - `0xFFFF`) n'ont aucune attribution particulière et à priori. Ils auront leur attribution seulement par le lien avec un **Universal Label** particulier. Par exemple, un `FFFE` peut être un Cryptographic Key ID dans un MXF et tout autre chose dans un autre. Il ne faut donc jamais se baser sur un **Local Tag Dynamique** pour déterminer le type d'un **item.value**.

Avec notre précédent listing, nous pouvons déjà identifier les statiques et les dynamiques :

```
+-------------------------------------------------+
|  0201     060e2b34.01010102.04070100.00000000   |      \
|  0202     060e2b34.01010102.07020201.01030000   |       |
|  0602     060e2b34.01010102.05300404.01000000   |       |
|  1001     060e2b34.01010102.06010104.06090000   |       |
|  1101     060e2b34.01010102.06010103.01000000   |       |--- Local Tag Statiques
|  1102     060e2b34.01010102.06010103.02000000   |       |
|  1201     060e2b34.01010102.07020103.01040000   |       |
|  1501     060e2b34.01010102.07020103.01050000   |       |
|  1502     060e2b34.01010102.04040101.02060000   |       |
|  1503     060e2b34.01010101.04040101.05000000   |       |
|  1901     060e2b34.01010102.06010104.05010000   |      /
+-------------------------------------------------+
|  FFF8     060e2b34.0101010a.04010603.01000000   |      \
|  FFF9     060e2b34.01010109.06010104.06100000   |       |
|  FFFA     060e2b34.01010109.02090301.02000000   |       |
|  FFFB     060e2b34.01010109.02090302.01000000   |       |--- Local Tag Dynamiques
|  FFFC     060e2b34.01010109.02090301.01000000   |       |
|  FFFD     060e2b34.01010109.06010102.02000000   |       |
|  FFFE     060e2b34.01010109.01011511.00000000   |       |
|  FFFF     060e2b34.01010109.06010104.020d0000   |      /
+-------------------------------------------------+
```

Ainsi, lorsque nous allons lire un KLV avec des **Local Tag Dynamiques**, on doit toujours se référer au **Primer Pack** pour connaitre l'**Universal Label** adossé à notre **Local Tag Dynamique** : c'est là son principal (seul?) intérêt.

L'**Universal Label** permet véritablement identifier le nom et donc le type de données d'un item.

Par exemple, si on se retrouve avec un **Local Tag** `FFAA` et que son **Universal Label** est 060e2b34.01010109.02090301.02000000, alors ce dernier identifie un **Cryptographic Key ID** : Un identifiant spécfique pour la partie cryptogaphique.

Pour chaque **Local Tag Dynamique**, nous recherchons, dans les différentes documentations SMPTE, les informations sur son **Universal Label** attribué. Nous pourrons alors identifier à quoi correspondent réellement les différents **Local Tags Dynamiques** de ce MXF.

Les **Local Tag Statiques** pouvant être identifiés par leur identifiant court : dans les documentations SMPTE, vous aurez le **Local Tag** accompagné de leur **Universal Label** directement.

N'oubliez pas que les **Local Tag Statiques** sont considérés maintenant comme **deprecated**. Basez-vous donc si possible, uniquement sur les **Universal Label** pour identifier les différents **Local Tag**, qu'ils soient dynamiques comme statiques.

Ainsi, si nous reprenons notre listing LT+UL, nous pouvons dégrossir les informations. Pour les premiers **Local Tag** (donc statiques), vu que nous connaissons déjà leur représentation, nous pouvons déjà les définir :

```
+-------------------------------------------------------------------------------------------------------+
|  0201     <= (Local Tag) Data Definition          -   060e2b34.01010102.04070100.00000000   <= (UL) Data Definitio
|  0202     <= (Local Tag) Duration                 -   060e2b34.01010102.07020201.01030000   <= (UL) Duration
|  0602     <= (Local Tag) Event Comment            -   060e2b34.01010102.05300404.01000000   <= (UL) Event Comment
|  1001     <= (Local Tag) Structural Components    -   060e2b34.01010102.06010104.06090000   <= (UL) Structural Com
|  1101     <= (Local Tag) SourcePackageID          -   060e2b34.01010102.06010103.01000000   <= (UL) SourcePackageI
|  1102     <= (Local Tag) SourceTrackID            -   060e2b34.01010102.06010103.02000000   <= (UL) SourceTrackID
|  1201     <= (Local Tag) Start Position           -   060e2b34.01010102.07020103.01040000   <= (UL) Start Position
|  1501     <= (Local Tag) Start Timecode           -   060e2b34.01010102.07020103.01050000   <= (UL) Start Timecode
|  1502     <= (Local Tag) Rounded Timecode Base    -   060e2b34.01010102.04040101.02060000   <= (UL) Rounded Timeco
|  1503     <= (Local Tag) Drop Frame               -   060e2b34.01010101.04040101.05000000   <= (UL) Drop Frame
|  1901     <= (Local Tag) Packages                 -   060e2b34.01010102.06010104.05010000   <= (UL) Packages
+-------------------------------------------------------------------------------------------------------+
```

Pour les seconds **Local Tag** (donc dynamiques), nous n'avons aucun moyen d'identifier ce qu'ils sont, aucune documentation SMPTE vous donnera ce que représente un `FFFA` par exemple, nous devons donc nous baser sur leur **Universal Label** :

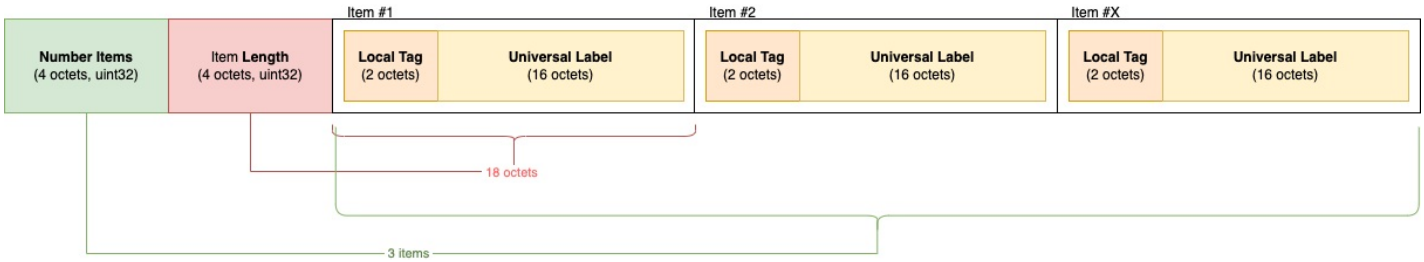
```
+-------------------------------------------------------------------------------------------------+
|  FFF8     <= ?                                -   060e2b34.0101010a.04010603.01000000   <= (UL) Resolution Size
|  FFF9     <= ?                                -   060e2b34.01010109.06010104.06100000   <= (UL) Generic Descrip
|  FFFA     <= ?                                -   060e2b34.01010109.02090301.02000000   <= (UL) Cryptographic H
|  FFFB     <= ?                                -   060e2b34.01010109.02090302.01000000   <= (UL) Message Integri
|  FFFC     <= ?                                -   060e2b34.01010109.02090301.01000000   <= (UL) Cipher Algorith
|  FFFD     <= ?                                -   060e2b34.01010109.06010102.02000000   <= (UL) Source Essence
|  FFFE     <= ?                                -   060e2b34.01010109.01011511.00000000   <= (UL) Context ID
|  FFFF     <= ?                                -   060e2b34.01010109.06010104.020d0000   <= (UL) Context SR
+-------------------------------------------------------------------------------------------------+
```

Nous venons de faire notre mapping entre **Local Tag** et leurs propriétés. Il nous sera utile quand nous irons lire les données dans les KLV utilisant les **Local Tags**. Sans cela, nous ne pourrons pas correctement interpréter les valeurs stockées dans chaque item.

> **Au final, c'est quoi l'intérêt d'avoir ce mapping au lieu d'utiliser les UL directement dans les KLV ?**
>
> Aucune idée. Autre question ? :)
>
> Je n'ai pas trouvé (ou je suis passé à côté) la véritable raison de ce choix. Pourquoi ne pas utiliser directement les **Universal Label** au lieu d'un **Local Tag** ? Je suppose que c'est utile pour les systèmes embarqués ou les systèmes ayant peu de ressources, il sera plus facile de stocker et manipuler plusieurs clefs à 2 octets que des clefs à 16 octets.
>
> Une autre possibilité, historique celle-là, les premiers **Local Set** utilisaient des **Local Tag** mais ils se sont aperçus que se baser sur 2 octets allait rapidement être bloquant et qu'il fallait donc passer à leur représentation 16 octets directement. Cependant, il fallait garder une compatibilité grâce à une table de mapping entre **Local Tag** et **Universal Label**.
>
> Ce sont deux possibilités comme d'autres sans véritables sources.

LES DONNÉES BRUTES DU KLV

Voici l'intérieur de la **Value** d'un KLV **Primer Pack** (avec seulement 3 items) :



Concrètement, voici les premiers octets d'un KLV **Primer Pack** :

```
060e2b34020501010d010201010501008300062600000057000000120201060e2b3401010102
0407010000000000202060e2b340101010207020201010300000602060e2b34010101020530
0404010000001001060e2b340101010206010104060900001101060e2b340101010206010103
0100000011020 60e2b340101010206010103020000001201060e2b340101010207020103010 4
00001501060e2b340101010207020103010500001502060e2b340101010204040101020600 0
1503060e2b340101010104040101050000001901060e2b340101010206010104050100001902
060e2b340101010206010104050200002701060e(...)
```

Comme cela, on peut déjà distinguer en tout début, l'**Universal Label** du KLV, suivi de son **Length** et enfin de **Value**. Passons maintenant à la découpe des données :

```
060e2b34020501010d01020101050100          ◄- (KLV) UL Primer Pack
83000626                                  ◄- (KLV) Taille Données
0000005700000012020 1060e2b34010101020407010000000000202060e2b34010101020702
020101030000060206 0e2b34010101020530040401000000100 1060e2b340101010206010104
060900001101060e2b340101010206010103010000001102060e2b340101010206010103020 0
00001201060e2b340101010207020103010400001501060e2b340101010207020103010500000
1502060e2b340101010204040101020600001503060e2b340101010104040101050000001901
060e2b340101010206010104050100001902060e2b340101010206010104050200002701(……)
```

Dans la partie **Value**, on distingue :

- 4 octets pour le **nombre d'item**. Ici, nous avons `0x57`, donc **87 items** en tout.
- 4 octets pour **la taille de chaque item**, ici nous avons `0x12`, donc une taille de **18 octets** par item - qui comprendront donc notre **Local Tag** (2 octets) et notre **Universal Label** (16 octets)
- Et juste après ses deux valeurs, nous avons notre succession d'items.

Chaque item est un **Local Tag** (2 octets) accompagné de son **Universal Label** (16 octets).

ETUDE RAPIDE DE L'UNIVERSAL LABEL DE PRIMER PACK

```
UL = 06.0E.2B.34.02.05.01.01.0D.01.02.01.01.05.01.00
                          ^^---------------------------- Item Designator   : Organizationally registered
                             ^^-------------------------- Organization      : AAF
                                ^^----------------------- Application       : MXF File Structure
                                   ^^--------------------- Structure Version : Version 1
                                      ^^------------------ Structure Kind    : MXF File Structure Sets & Packs
                                         ^^--------------- Set/Pack Kind     : Primer Pack
                                            ^^------------ Partition Status  : Version of Primer Pack
```

NOTES

1. *« A UUID can be stored in a data field of type UL by swapping the top and bottom 8 bytes of the UUID (the most significant bit of the first byte of such a swapped UUID is always 1); In SMPTE ST 377-1, the combined UL/UUID value is called an AUID where the UUID is byte- swapped and called an IDAU where the UL is byte-swapped. »* -- SMPTE.EG 377-3-2013 - MXF Engineering Guideline ↩