MXF — a technical overview

P. Ferreira
MOG Solutions

This is the second of a series of two articles on the Material eXchange Format. While the first one [1] gave an historical perspective on the evolution of MXF, this one goes further into the specifics, describing the technical details of this file format.

The ultimate goal of MXF is to interchange Essence (picture, sound and data), Descriptive Metadata and composition information: that is, very simple EDLs. This of course has to be done efficiently and in a platform-independent way. Furthermore, MXF has extensibility as one of its main principles: it must be possible to add more capabilities without changing the core format or breaking backward compatibility.

In order to achieve these goals, MXF defines a set of high-level concepts and some "scaffolding" that binds everything together:

- **Essence Containers** – a well-defined way of carrying essence;
- **Index Tables** – an essence-agnostic way of mapping time offsets to byte offsets;
- **Header Metadata** – a technical description of the structure and contents of the file, plus optional Descriptive Metadata;
- **Partition Metadata** – the metadata used to describe the physical structure of the file;
- **Random Index Pack** – an index to assist in the location of Partitions in the file;
- **Run-In** – an optional, non-standard block of data at the top of the file, only used in specialised Operational Patterns.

Like most file formats, MXF uses a header and a footer *(Fig. 1)*. While the header allows a decoder to easily identify a file as MXF, the footer unambiguously marks the end of the data.

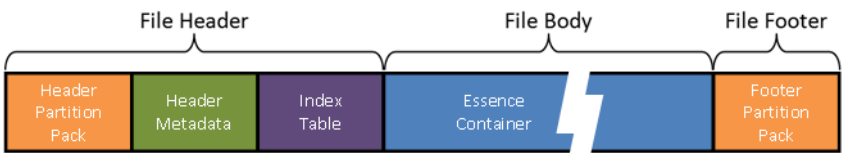

Figure 1
Layout of a simple MXF file

The big lump in the middle, the file body, is where essence is carried, typically accompanied by an index table that speeds up random access. Yet, the most interesting part is probably the block called Header Metadata: a rich and extensible model for expressing the contents of the file, to describe it unambiguously in technical terms and, optionally, to describe it in semantic terms.

KLV coding

The above is a bird's eye view of an MXF file. But at ground level, an MXF file is a sequence of bytes, each having its content and "meaning", which must unambiguously be understood by a compliant decoder. Therefore, a clear syntax must be defined for a successful interchange to happen. Then again, if MXF defined exactly what kinds of items were allowed, extensibility would be seriously impaired. In order to accommodate for future extensions, MXF adopts a strategy that permits adding new items without breaking backward compatibility: ***KLV coding*** [2].

KLV coding uses a triplet to encode each element:

- **Key** – the identifier of the element, an SMPTE Universal Label (UL) [3];
- **Length** – the length of the data (coded in BER [4], a variable-length encoding used in order to reduce the amount of space required for this item);
- **Value** – the actual value of the element.

Figure 2
MXF as a sequence of KLV packets

An MXF file is strictly composed of a sequence of KLV packets *(Fig. 2)*. Basically, KLVs can contain Data Items (e.g. a video frame) or Data Groups (e.g. a Metadata Set).

When parsing an MXF file, a decoder looks at the key and, if the key is known, reads the value and processes it; if the decoder does *not* recognize the key, it uses the length field to skip the data. This allows unknown elements (e.g. new Metadata Sets) to be discarded and not cause the decoder to go off sync.

Data items

The most obvious content for a KLV is a single item, a good example being a video frame.

Fig. 3 (upper) shows an example of a Data Item in a KLV. The key identifies the data as a picture item (this key is defined in [5]); the length is 200265 encoded in BER; finally, the value is the picture data itself.

Data groups

Not all items are wrapped alone in its own KLV. Doing that for small items such as the metadata that defines a frame width or the offset of the previous partition would incur too much overhead. Plus, it makes sense to bundle together related items in a single KLV.

For example, a "Person" metadata set can be wrapped in a single packet. Therefore, MXF uses two of the grouping methods defined in the KLV standard: ***Local Sets***, used for example to wrap Metadata Sets; and ***Defined-length Packs*** which are used, among

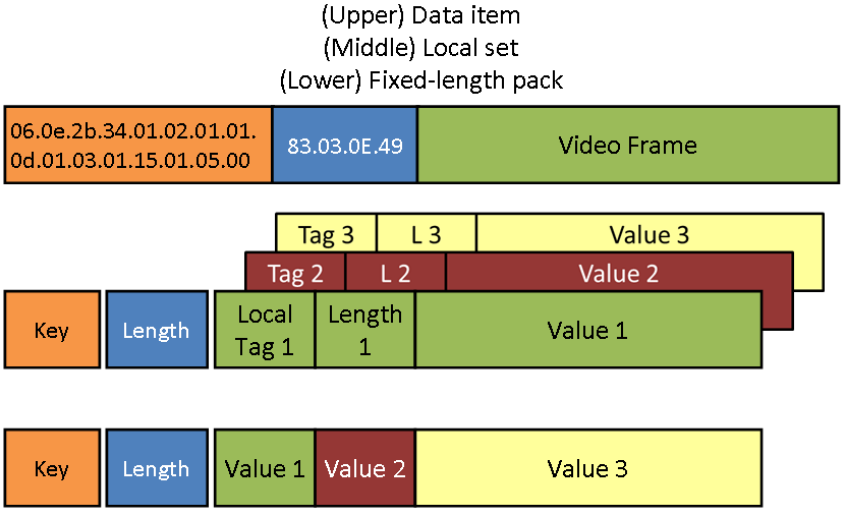

Figure 3
Examples of KLV packets

others, for Partition Packs or Index Table Segments.

In a Local Set *(Fig. 3, middle)*, the KLV Key is a 16 byte UL but each element is identified by a 2-byte *Local Tag* that can be mapped to a full UL via an MXF mechanism called *Primer Pack*. Elements follow any order inside the Set and can be omitted if not used.

In a Defined-length Pack *(Fig. 3, lower)*, a standard (e.g. MXF itself) defines the meaning of each element and their individual lengths. Elements follow a predefined order inside the pack.

KAG

One final remark is related with file-access efficiency. Most storage devices are optimized to be accessed in fixed-size "chunks", or sectors. MXF caters for this by means of a mechanism called the KLV Alignment Grid. This allows aligning "important" parts of the file with the sectors, thus increasing access speed. This is done by inserting special KLVs called *fill items* between the relevant KLV packets.

Essence containers

MXF files may refer to external essence, at the limit carrying only metadata. However, most applications use MXF to carry the actual essence.

MXF carries essence inside a structure appropriately named ***Essence Container***. The core MXF standard [5] defines the requirements for Essence Containers, which are materialised in the MXF Generic Container [6]. Then, for each essence format that may be carried in MXF, an additional mapping defines the specifics, e.g. for MPEG [7], for AES3/BWave [8] or AVC-Intra [9].

The MXF Generic Container is a streamable data container, meaning it was designed to allow the audiovisual material to be continuously decoded, which is achieved by interleaving the data streams, typically over a 1-frame duration.

The MXF Generic Container comprises a contiguous sequence of Content Packages [10], each of which has up to five basic components, known as ***items***: system, picture, sound, data and compound. Each item can contain more than one element: e.g. an audio item can be composed of four audio elements, one for each channel.

The purpose of Picture and Sound Items is trivial; Data Items are used to carry teletext, closed cap-

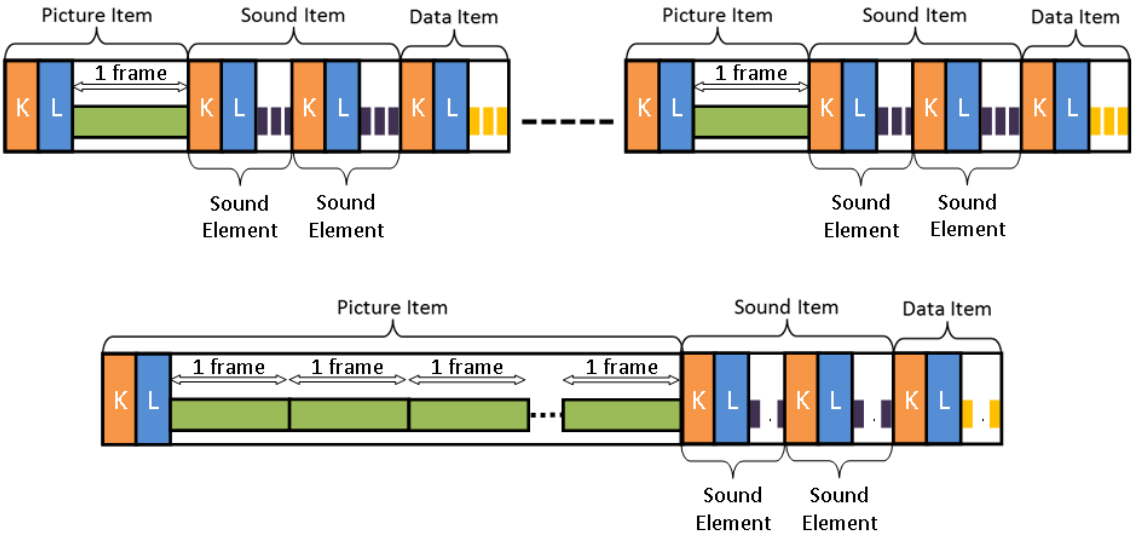

Figure 4
Frame wrapping (upper) and clip wrapping (lower)

tions or similar data; Compound Items include inextricably-interleaved essence, such as DV. The System Item provides ancillary information for each content package, as its timecode, for example.

The Generic Container can carry the essence in two basic ways: frame-wrapping and clip-wrapping. The latter is composed of one or more Content Packages, each being an interleave of all items over the duration of a frame, creating a KLV packet for each individual element *(Fig. 4, upper)*.

The former is composed of a single Content Package, that is, all samples are lumped together into a single KLV for each element *(Fig. 4, lower)*.

In practice this means that, for an MXF file to be streamable, either it has only one essence element, or it must be frame-wrapped. For example, a clip-wrapped file with video and four audio channels would have to buffer all the video and most of the audio before being able to play all tracks in parallel.

Ancillary data

VBI lines and ANC packets have many applications and, while in some cases this information can be encapsulated as standardized MXF essence or metadata items (e.g. embedded audio, timecode), in other cases this approach is not practical. For instance, some applications encode non-standard data in VBI lines, rendering it impossible to preserve that data unless it is forwarded untouched.

MXF provides four different locations for metadata:

- in the Header Metadata;
- in a separate data stream, which can be interleaved or multiplexed with essence items;
- linked to the System Item;
- embedded in the essence itself, e.g. VBI encoded with the active video.

One could think that, ideally, all metadata should be in the Header Metadata. However:

- in streaming applications, most information will not be known beforehand;
- the number of items may grow so large that it exceeds what can be fitted in a Metadata Set (64k entries);
- decoders would have to buffer all metadata in order to present it synchronously with the essence.

In order to provide for a flexible way of carrying these data, [11] defines a transport for VBI and ANC data, wrapped as data elements in an Essence Container. This allows the encoder to insert those data at the pace which makes sense. A compliant decoder can then easily interpret the data or reinsert it back into an SDI stream.

Abbreviations

ANC	ANCillary data	**OP**	(MXF) Operational Pattern
BER	Basic Encoding Rules	**RIP**	Random Index Pack
CBE	Constant Bytes per Element	**SDI**	Serial Digital Interface
EC	(MXF) Essence Container	**SMPTE**	Society of Motion Picture and Television Engineers (USA) http://www.smpte.org/
EDL	Edit Decision List		
FP	(MXF) File Package		
GoP	Group of Pictures	**UL**	(SMPTE) Universal Label
KLV	(SMPTE) Key Length Value	**UMID**	(SMPTE) Unique Material Identifier
MP	(MXF) Material Package	**VBE**	Variable Bytes per Element
MXF	Material eXchange Format	**VBI**	Vertical Blanking Interval

Index tables

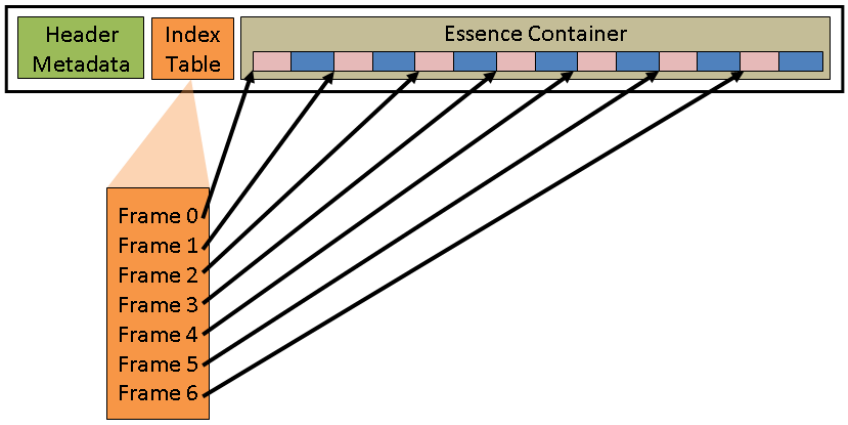

Figure 5
Index tables

While some applications just need to play a file continuously, many require accessing a file randomly, as shuttling or scrubbing, extracting part of a file's content, conforming an EDL, etc.

Achieving this properly, without additional support, requires an intimate knowledge of the essence format and, even then, it may not be efficient. Worse, the essence in an MXF file can be pretty much anything, so how does an application know where in the file to look for a frame, in a way that is agnostic to the essence format? MXF includes a flexible and highly effective tool for that purpose: the ***Index Table,*** which is, roughly, a table that maps frame indexes into byte offsets in the Essence Container *(Fig. 5)*.

How do Index Tables deal with the diversity of essence formats? There are two basic kinds of Essence: ***Constant Bytes per Element*** (CBE), which include DV, IMX, PCM, DNxHD, AVC-I; and ***Variable Bytes per Element*** (VBE), whose major representative is Long GoP MPEG.

CBE essence is quite simple to index: all frames are equally sized, so a single entry can give enough information that applies to the whole file. Plus, the index table can be constructed even before the essence is encoded.

On the other hand, indexing of VBE essence requires one entry per frame, so the Index Table grows continuously with the number of frames. Worse: since the Index Table is not known beforehand, the encoder cannot write the table before the essence, and the decoder does not know how to access the essence items until it sees the Index Table. This gives rise to a corollary: in a streamed file with VBE essence, the Index Table entries must come after the essence they index.

In order to mitigate this fact, Index Tables can be split into one or more Index Table Segments, which are interspersed with the Essence Container, allowing the encoder to flush index table segments periodically.

Partitions

This very fact of having to "flush" the index table periodically is one of the major reasons that led to MXF having partitions: whenever an encoder decides so, it creates a new partition, writes the index

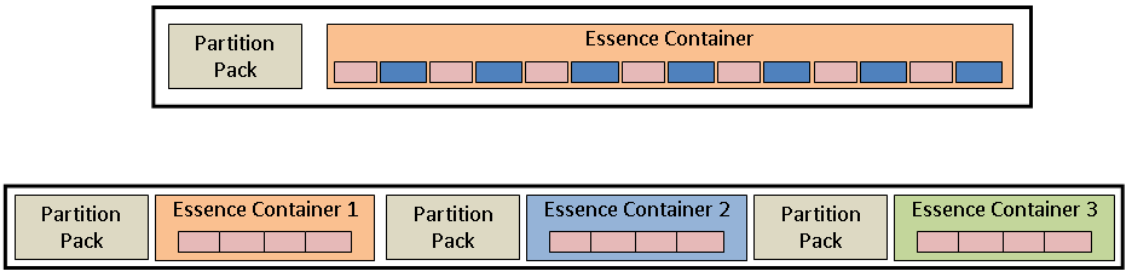

Figure 6
Interleaving vs. Multiplexing

segment, and continues on with the Essence Container.

Another reason to have partitions is the ability to multiplex several Essence Containers. In fact, MXF provides two ways of transporting several "tracks" of essence: interleaving, done at the Essence Container level *(Fig. 6, upper)*; and multiplexing, done at the partition level *(Fig. 6, lower)*.

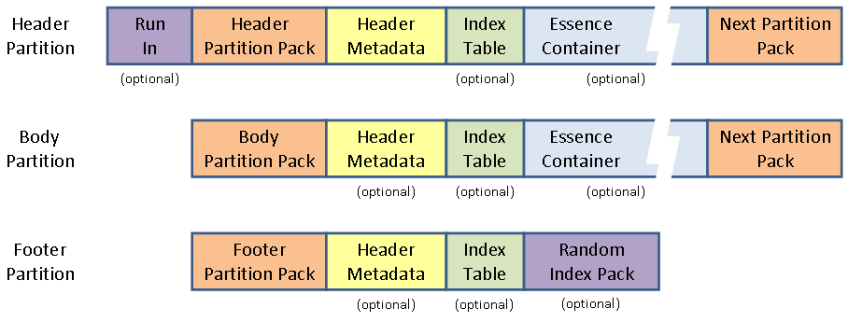

Figure 7
Partition rules

So, what are the rules for partitions? How can an MXF file be structured? As shown in *Fig. 7*, the file must have a Header and a Footer Partition; plus there can be zero or more Body Partitions.

Every partition starts with a Defined-length Pack called a Partition Pack. The key of the Partition Pack identifies it either as a Header, Footer or Body Partition. In addition, the Partition Pack contains generic information such as the MXF version, the Operational Pattern of the file, the type(s) of the Essence Container(s) in the file, whether an Essence Container and/or Index Table is present in the partition, etc.

The Partition Pack is followed by the Header Metadata, which is required in the Header but optional in Body and Footer Partitions. And why would one write the Header Metadata more than once? Well, for two good reasons: one is for allowing decoders to "catch up" in the middle of a file (e.g. in a streaming application); the other one is to allow metadata updates. This raises a question: if we can repeat the metadata, how does a decoder know when it is final? The Partition Pack itself indicates the status of the Header Metadata, such as whether it contains a partial or final version of the metadata.

Finally, *Fig. 7* depicts a Run-in, which is an optional non-KLV-coded block for specific applications; and a Random Index Pack (RIP), which is the very last item in an MXF file and is basically a partition index, letting decoders quickly jump to partitions in the middle of the file.

Structural metadata

Now that we have covered the details of the MXF file layout, it is time to look into one of MXF's most distinctive features: **Structural Metadata**.

MXF Header Metadata is composed of Structural and Descriptive Metadata. While the latter obviously serves the purpose of describing content, the former relates to the structure and capabilities of an MXF file. This includes technically describing the various essence components; conveying EDL information on how the different ECs compose the desired output timeline; identifying the packages using UMIDs; storing historical derivation information, e.g. from what tape this material was ingested, etc.

This makes Structural Metadata fundamental in MXF, as otherwise it would be no more than just a dumb set of Essence Containers.

Model

MXF Structural Metadata leverages the object-modelling techniques developed in the IT industry and builds upon a set of object classes and a number of well-defined relationships among them. Each MXF file constructs instances of these classes (objects or, in MXF terminology, *Metadata Sets*), fills in their properties (or elements) with relevant information (for example, a "Track" Metadata Set will have an "Edit Rate" element) and links the different Metadata Sets to each other (e.g.

the “Content Storage” Metadata Set will link to a certain number of “Package” Metadata Sets).

A thorough description of this object model is clearly outside the scope of this article but it is certainly worth it to highlight some of the most interesting parts. The root of the object model is the “Preface” set *(Fig. 8)*. It includes generic information about the file, such as the MXF version and the kinds of Essence Containers carried in the file, and points to a number of other objects. For the sake of simplicity, let us forget about all other objects and focus on the “Content Storage”, especially on the “Package” objects it links to.

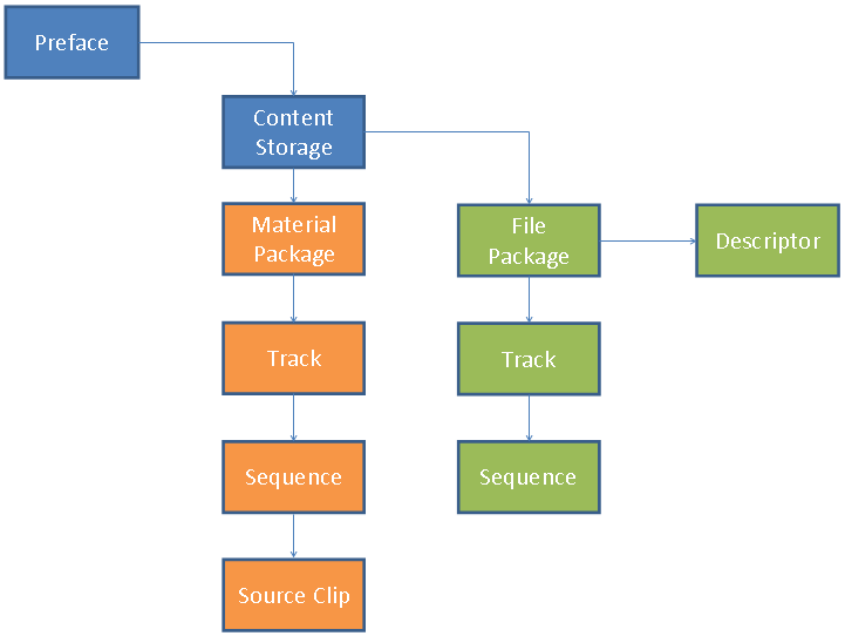

Figure 8
Part of the MXF object model

Each Essence Container (EC) in the file is described by a “File Package” (FP) Metadata Set in the Header Metadata. The FP includes a Track for each essence element. For example, an EC with video and four audio channels will typically have a Picture Track, four Sound Tracks and a Timecode Track with the timecode of the original material.

The FP also points to one or more “File Descriptor” sets. These play a hinge role in MXF as they provide a decoder with all the information required to understand the format (e.g. raster, chroma sampling, aspect ratio, audio sampling rate), without requiring it to actually decode the essence.

These FP objects describe the “input” of the MXF file. The output, what the viewer shall see, is described by the Material Package (MP). In the object model, the MP points to one or more segments of the FPs and lays down these segments, in its own set of tracks and sequences, as “Source Clip” objects. The way these segments are arranged ultimately determines what the viewer sees.

This is only a small part of the MXF object model, as it contains much more information, such as identification, hooks for descriptive metadata, historical derivation information, etc. The interested reader is invited to browse the relevant sections and annexes in the MXF standard.

Serialization

An object model is an abstract, multi-dimensional concept. In order for a MXF file to be able to convey the information to a decoder, this model has to be converted into a sequence of bytes. In MXF this is done by serializing each of the Metadata Sets as a KLV Local Set. The KLV key identifies the kind of Metadata Set and each element is wrapped in its own “mini-KLV”, identified by a Local Tag.

This takes care of “flattening” the model. The relationships between objects are maintained using several linking mechanisms, which include the UMIDs of the Packages, track identifiers and unique object identifiers.

Operational patterns

As we have seen, MXF can describe fairly complex EDLs, carry several Essence Containers, be laid out in several different ways. Forcing all decoders to handle this would be an exaggerated burden,

so MXF includes a mechanism to control the complexity of a file: the ***Operational Patterns***.

Operational Patterns are not part of the core specification but, right from the outset, MXF described the Generalized Operational Patterns (known as OP1a to OP3c). These OPs use the relationship between the Material Package(s) and the File Package(s) in order to constrain complexity, e.g., most MXF files say: "play the only File Package from start to end, with all tracks active". The complexity of the Generalized OPs varies along two axes: item complexity and package complexity *(Fig. 9)*.

There are applications for which the Generalized OPs are not enough and for that reason other OPs have been defined, either vendor-specific or as a standard, such as OPAtom [12].

OPAtom is intended for applications requiring a simple and predictable layout, with minimum scope for variation. Its most distinctive feature is that it only allows a single track of essence in a file, making it especially suitable for post-production environments, where it became highly popular.

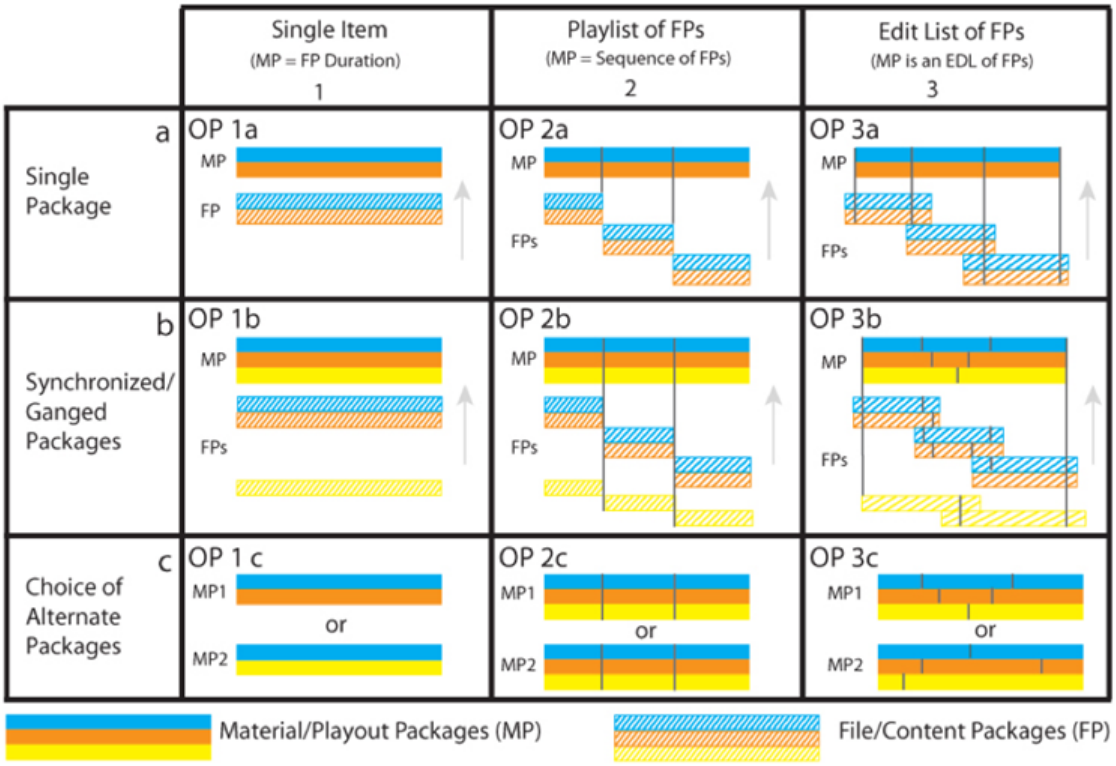

Figure 9
Generalized operational patterns

Timecode

MXF allows encoders to store timecode in several places, namely:

- in the Header Metadata, in material and source package timecode tracks;
- in system items, interleaved with the essence; or
- in the payload of picture, compound, sound and data elements, e.g. MPEG, DV, VBI or ANC packets.

While flexibility is often good, in this case it inevitably led to some confusion. Therefore, some interested parties defined guidelines on how to deal with timecode consistently. One of the most interesting efforts was made by the EBU, which published Recommendation R122 [13], with the goal of harmonising timecode implementations.

Conclusions

This article introduces the major building blocks of MXF. As explained earlier, the suite of MXF specifications consists of a large number of documents, so the interested reader is invited to browse these documents, focussing on the areas that are most relevant to the problems at hand.

MXF is widely used and has strong support from the industry; as such, there is a large number of both commercial and non-commercial discussion forums, which can help readers to find their way.

References

[1] Pedro Ferreira: **MXF – a progress report**
EBU Technical Review – 2010 Q3.

[2] SMPTE 336M-2004: **Data Encoding Protocol Using Key-Length-Value**

[3] ANSI/SMPTE 298M-1997: **Universal Labels for Unique Identification of Digital Data**

[4] ISO/IEC 8825-1:1998: **Information Technology – ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)**

[5] SMPTE 377-1M-2009: **Material Exchange Format (MXF) – File Format Specification (Standard)**

[6] SMPTE 379-1-2009: **Material Exchange Format (MXF) – MXF Generic Container**

[7] SMPTE 381M-2005: **Material Exchange Format (MXF) – Mapping MPEG Streams into the MXF Generic Container**

[8] SMPTE 382M-2007: **Material Exchange Format – Mapping AES3 and Broadcast Wave Audio into the MXF Generic Container**

[9] SMPTE RP 2027-2007: **AVC Intra-Frame Coding Specification for SSM Card Applications**

[10] **The EBU-SMPTE Joint Task Force for Harmonized Standards for the Exchange of Programme Material as Bit Streams – Final Report: Analyses and Results, July 1998**

[11] SMPTE 436: **MXF Mappings for VBI Lines and Ancillary Data Packets**

[12] SMPTE 390M: **Material Exchange Format (MXF) – Specialized Operational Pattern "Atom"**

[13] EBU Recommendation R 122: **Material Exchange Format Timecode Implementation**

Pedro Ferreira was born in 1973, in Guimarães, Portugal. He has an MSc. in Telecommunications and Computer Science from the University of Porto. After graduating, he worked at INESC Porto as a researcher on the use of Distributed Systems technology in Digital Television, collaborating as well in projects such as ACTS ATLANTIC. He also led the Distributed Systems and Essence Processing area in the BBC ORBIT project, was responsible for INESC Porto's participation in some EU IST projects and was engaged in standardization activities in SMPTE and Pro-MPEG.

In 2002, Mr Ferreira left INESC to co-found MOG Solutions, where he became responsible for R&D, leading the development of award-winning products such as MXF::SDK, bespoke projects such as the collaboration with NBC Olympics, as well as the participation in several EU projects, such as Worldscreen and EDCine.

Pedro Ferreira is now Director of Product Marketing and is striving to make MOG's products easier to use and more effective in streamlining file-based workflows. He is also responsible for MOG's training department, collaborating with the EBU and several other customers.

This version: 1September 2010

Published by the European Broadcasting Union, Geneva, Switzerland

ISSN: 1609-1469

Editeur Responsable: Lieven Vermaele

Editor: Mike Meyer

E-mail: tech@ebu.ch

The responsibility for views expressed in this article rests solely with the author