SMPTE ENGINEERING GUIDELINE

Material Exchange Format (MXF) — Engineering Guideline (Informative)

Warning: This Engineering Guideline is purely informative and meant to provide tutorial information to the industry about a Standard or Recommended Practice. Do not rely on this for interoperability, but rather the Standards and Recommended Practices referenced herein.

Table of Contents

Approved
January 29, 2013

Foreword

SMPTE (the Society of Motion Picture and Television Engineers) is an internationally-recognized standards developing organization. Headquartered and incorporated in the United States of America, SMPTE has members in over 80 countries on six continents. SMPTE's Engineering Documents, including Standards, Recommended Practices, and Engineering Guidelines, are prepared by SMPTE's Technology Committees. Participation in these Committees is open to all with a bona fide interest in their work. SMPTE cooperates closely with other standards-developing organizations, including ISO, IEC and ITU.

SMPTE Engineering Documents are drafted in accordance with the rules given in Part XIII of its Administrative Practices.

SMPTE EG 377-3 was prepared by Technology Committee 31FS.

Introduction

The Material Exchange Format (MXF) was developed by the industry to provide a means of exchanging video, audio, data essence and metadata content in a unified data format. The MXF file structure provides a means of exchanging metadata using a formal data structure that permits interoperability together with an essence container that can accommodate many different essence formats both compressed and uncompressed and including video streams, individual pictures, sound and essence data (such as sub-titles, VBI data etc.).

The starting point of the work was the report from the Task Force for Harmonized Standards for the Exchange of Program Material as Bitstreams as identified in the Bibliography. The work of this report was used as the basis for an exchange file format by the Pro-MPEG Forum which acted as a catalyst to encourage industry-wide participation to create a unified file format that could be used for distribution, exchange and cut-editing. The results of this initial development were then brought to SMPTE to create an internationally recognized sets of standards, recommended practices and engineering guidelines that form the MXF document suite.

The other key aspect of the development of MXF was the adoption of the SMPTE Key-Length-Value approach to coding the whole of the MXF file structure using both individual and group coding for a consistent approach within the whole file format. The evolution of XML coding can be seen as attractive, but in many cases, such coding is not suitable for the high-speed and wide-bandwidth requirements of high definition digital television and digital cinema. Instead, there are methods that can be used to transform the metadata parts of MXF files to XML and vice-versa (though only practical for the metadata part of MXF files).

1 Scope

This guideline gives an introduction to and the background for the Material eXchange Format (MXF). This guideline describes the technology involved in the format, the names of the various elements within the format, and the way in which the format can be used within the real world applications.

Some parts of the descriptions within this guideline are generic to file formats, while other parts are specific to the material exchange format. There are descriptions of the object-oriented technology used within the MXF format, as well as a discussion of the metadata that can be used within the file. There are worked examples within this guideline to guide users and help improve the interoperability of applications using different MXF files.

2 MXF Document Suite Structure

The MXF specification is split into a number of categories in order to create a document structure that allows new applications to be covered in the future. These categories are:

Category 1 – MXF File Format Specification – Normative (SMPTE ST 377-1)

Category 2 – MXF File Format Extensions – Normative (e.g. SMPTE ST 377-2 is the KLV Extension Syntax, KXS)

Category 3 – Engineering Guidelines – Informative (including this document)

Category 4 – Operational Patterns – Normative (e.g., OP1a is SMPTE ST 378)

Category 5 – MXF Descriptive Metadata Schemes – Normative (e.g., DMS-1 is SMPTE ST 380)

Category 6 – Essence Containers – Normative (e.g., the MXF Generic Containers: SMPTE ST 379-1 and SMPTE ST 379-2)

Category 6a – Mapping Essence and Metadata into the Essence Container – Normative (e.g., Mapping MPEG Streams into the Generic Container is SMPTE ST 381 (all parts))

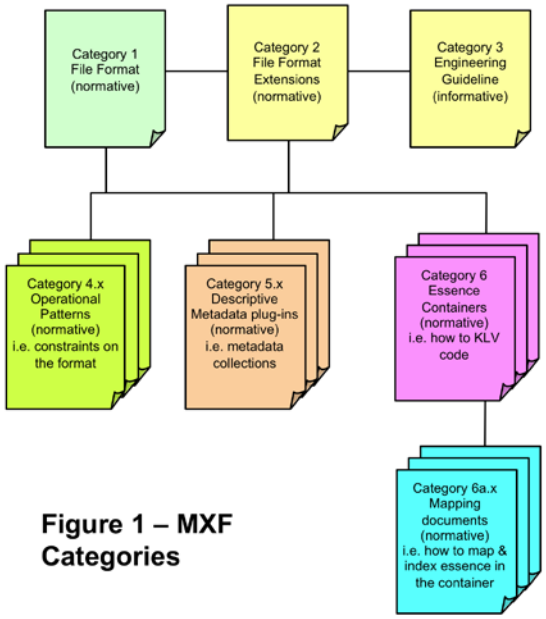

Figure 1 – MXF Categories

When creating an MXF application or system, you need to ensure that you have the latest versions of all of these documents.

Note that in some cases, the MXF specifications have been revised by splitting into multiple parts (e.g. SMPTE ST 381 parts 1~n) or providing different variants (e.g. SMPTE ST 379-1 and SMPTE ST 379-2).

The MXF categories are summarized below.

Category 1 is a normative definition of the format of an MXF file. It defines the toolbox from which different file interchange tools can be chosen to fulfill the requirements of different applications. The MXF file format defines the syntax and semantics of MXF files. Since MXF is an extensible format, new standards can be added that define extensions to the format defined by SMPTE ST 377-1. Category 1 also contains formal definitions of the many acronyms, terms and data types used in the MXF specifications, including this engineering guideline.

Category 2 provides for extensions to the file format to provide tools not present in the basic file format. These extensions provide new tools to extend the capabilities on MXF to provide extra functionality as needed for specialist applications,

Category 3 contains informative documents that help users to understand various aspects of MXF. This document is MXF engineering guideline, which is an introduction to MXF. This document introduces many of the concepts and explains what problems MXF is intended to solve. Part 3 also includes other engineering guidelines, including a descriptive metadata engineering guideline (SMPTE EG 42), which explains the concepts behind the use of descriptive metadata in MXF files. Other engineering guidelines can be added to the MXF document suite as and when there is a requirement.

Category 4 documents describe the operational patterns of the MXF format. In order to create an application to solve a particular interchange problem, some constraints and structural metadata definitions are required before SMPTE ST 377-1 can be used. An operational pattern defines those restrictions of the format that allow interoperability between applications of defined levels of complexity. Applications that use the MXF format need to adhere to one of the operational patterns in order to achieve interchange.

Category 5 documents define MXF descriptive metadata sets that can be plugged in to an MXF file. Different application environments will require different metadata sets to be carried by MXF. These collections of metadata sets are described in the part 5 document(s).

Category 6 documents define the essence container of the MXF format for containing picture and sound essence. There can be limitations to the essence container that are needed for a particular operational pattern. The reader is advised to cross-reference at least categories 1 and 4 both before and during implementation. The MXF generic container is a standardized essence container providing an encapsulation mechanism that allows many formats to be mapped into MXF.

Category 6a comprises a number of documents for mapping many of the essence and metadata formats used in the content creation industry into the defined MXF essence container.

The MXF document suite makes reference to other documents that contain information required for an MXF system. One such document is SMPTE ST 336 that provides the basic coding rules used throughout MXF. Other documents include the SMPTE metadata dictionary, SMPTE RP 210, which contains definitions of metadata parameters and the SMPTE Labels Register, SMPTE RP 224, which contains a list of defined 16-byte labels for use in MXF sets.

SMPTE ST 377-1 provides a full list of all MXF documents for user guidance.

The Bibliography of this document provides a list of additional documents and publications for background reading to aid the understanding of MXF.

2.1 About this Document

The information in this document is ordered to aid readability. Concepts are introduced gradually and repeated in more detail later in the document. This is done to make the document easier to read; however, it does make the document somewhat less useful as a guide. The table of contents [see pages 1 and 2] is provided at the start of the document to allow easy access to each heading within the document.

This document is a guideline and does not specify conformance points. Nor does this document contain conformance terms except where present in a quote from normative documents for the purpose of illustration.

3 MXF Concepts

The concepts in MXF are introduced in a way that gives an overall view of the specification and the concepts embodied within it. Once the introduction is understood, the requirements of the file format are discussed. Some specific words and phrases used in the specification are then defined and finally

the material exchange format is introduced in much more detail. Although this entire document is informative, its aim is to give sufficient information for technical and non-technical readers to have a basic understanding of MXF.

3.1 MXF Background

The MXF specification is designed to interchange multimedia information as a file. The design allows users to take advantage of non-real time transfers and to package together essence and metadata for effective interchange between servers and between businesses. MXF is not a panacea, but is an aid to automation and machine-to-machine communication. It allows essence and metadata transfer without the metadata elements having to be manually re-entered.

The MXF specification is intended to allow the storage and interchange of video, audio, data essence and metadata as a file. MXF files can be efficiently stored on a variety of physical media as well as transported over communications links.

Finally, the MXF specification is intended to be extensible. Considerable effort was put into ensuring that SMPTE ST 377-1 is compression codec independent, resolution independent and can be constrained where needed to suit a large number of application environments. The document structure has been created to allow new applications to extend MXF in a backwards-compatible way.

3.2 MXF Design Criteria

The MXF file format was designed for use in high throughput hardware or software devices. This translates into the need for well-defined design parameters for buffer size, latency, and the need for algorithmic simplicity. MXF is also intended to cover a very large application space, and not all the requirements apply to all the applications.

3.2.1 Representing multi-byte numbers

There are two distinct ways in which multi-byte numbers are stored in computer systems - big-endian and little-endian. Big-endian systems place higher value bytes in the lower value addresses (i.e. MSB first), whereas little-endian systems do the reverse. Because of the two storage formats, MXF defines the storage format as follows:

- MXF header metadata uses the big-endian data format only. The primary reason behind this decision was to address the handling of dark metadata where the endian-ism cannot be known (because the metadata is dark).

- Essence containers use the byte order defined by the individual specifications of each essence container where needed.

3.2.2 Requirements for application diversity

Some applications can require metadata to be processed separately from the essence data. Other applications can require metadata to be stored with the essence data. This requires efficient insertion and extraction of the metadata from the essence data container(s) of the file.

Indexing of the file content is desirable to provide for rapid access to the file contents and this desire is served by the provision of index tables. Some applications prefer such index tables to be accessed separately from the essence; others can require the two to be accessed together. In some cases, the index tables are most naturally stored at the start of the file; however, while recording, the most natural location is at the end of the file. This diversity requires efficient insertion, extraction and relocation of index tables within the file.

4 MXF File Structure

The MXF format follows a common theme of many file formats with the following basic structure:

- A file **header** that provides information about the file as a whole, including labels for the early determination of decoder compliance. The header metadata in the file header is a complex data structure that provides a full description of the contents of the essence container.

- A file **body** that comprises picture, sound and data essence stored in essence containers. Essence containers from different tracks can be interleaved or separated. Section 9.6 on operational patterns goes into more detail on how the essence content is structured.

- A file **footer** that terminates the file. The file footer can include some information not available at the time of writing the file header (such as the duration of the file). In certain specialized operational patterns, the file footer can be omitted.

A simple MXF file is illustrated in Figure 2.

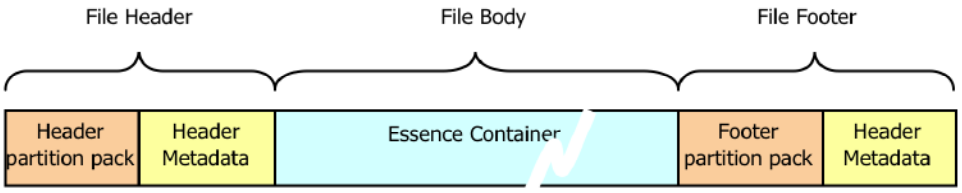

Figure 2 – Simple MXF file

4.1 What are the File Header, the File Body and the File Footer?

The basic file header, file body and file footer are identified in Figure 2. The reason for the split is simple. The file header is designed to be small enough that it can easily be isolated and sent to a microprocessor for parsing. The bulk of the file will usually be the file body — this is the picture, sound and data essence that commonly exceeds 90% of the file data. The file footer provides a means to put the header metadata at the end of the file.

Why is a file footer needed? In certain applications, such as recording an audio-video stream to an MXF file, there will be header metadata values that are not known until the recording is finished. The file footer provides a mechanism for providing such header metadata values. The file footer also provides a clear indication that the file has terminated.

4.2 MXF Partitions

MXF allows files to be divided into file partitions for the file header, the file body and the file footer. This sub-section describes the use of file partitions in MXF files including the reasons for their use.

4.2.1 File partitions

MXF files can include optional partitions that can be used to split up the file components to provide a variety of requirements as follows:

1. Robustness of metadata information by repetition of the header metadata;
2. Multiplexing of different essence containers;
3. Easier location of essence container data when using high-speed tape devices;
4. Optimizing the distribution of the data in a file for storage or transmission.

Notes:

1. Repetition of the header metadata within a body partition is dependent upon the application on a per-application basis. Such applications can be found, for example, in the transfer of an MXF file as a stream over a unidirectional link and in data tape shuttling. One purpose of such header metadata repetition is to support the recovery of critical metadata in applications where the file can be interrupted or where a decoder starts to receive data in mid-transfer.

2. Multiplexing and storage optimization can be a complex subject and is highly dependent on the storage or transmission device used. Hard discs, DVDs, satellite links and tape devices all have different requirements. The MXF structure allows a great deal of flexibility in the positioning of the partitioning information and the use of fillers to allow optimization for different devices. Typically, if storage or transmission optimization is important in an application then the MXF encoder will know which parameters are important. MXF provides the tools, but encoders can make those optimizations that add value to a specific application.

4.2.2 Partition Kinds

A partition is a division of data within an MXF file. There are three different kinds of partition in an MXF file:

Header Partition – this is the first partition of the file and occurs only once in the file header of an MXF file.

Footer Partition – this is the last partition in the file and occurs only once in the file footer of an MXF file.

Body Partitions – one or more partitions located in the file body. Multiple body partitions can be used to divide an essence container into a sequence of parts, provide for multiple essence containers or provide for multiple essence containers each of which are divided into a sequence of parts.

A header or body partition can be open or closed, but the footer partition can only be closed. The normative definition of the terms ‘open’ and ‘closed’ is provided in SMPTE ST 377-1 and introductory information is given below:

Open – This marks the information in a partition with a “caution” notice. Any metadata information in the partition was correct at the time of writing, but the application writing the file had not completed the writing process. This means that some of the information can be absent, or can turn out to be plain wrong when the file is ultimately closed. For example, a capture device can have identified a picture and a sound track when it initially started writing the file. During the writing process, a second sound track commenced — this track was not described in the open header metadata.

Closed – This marks any metadata information in the partition as finalized. The application or device creating the file correctly terminated the file and all the properties of the Metadata sets were filled in to the best of the application’s ability. In the example above, a repetition of the Header Metadata would be placed in the footer that correctly described the existence and duration of the second Sound Track. All closed partitions in a file must have the same metadata property values without fail. This allows an MXF decoder to determine that the metadata is correct as soon as it finds a closed partition. SMPTE ST 377-1 states that the file footer, if present, will always be a closed partition.

An MXF File can only be called a “closed” file if there is at least one closed partition with metadata. It is important to note that robustness is enhanced when all the partitions in a file are closed. If a file is accidentally truncated during a transfer and the only closed partition in the file was the footer, then the file is no longer a “closed” file. If robustness is desired (and it usually is), application and device developers are urged to close all the partitions of their files. All valid MXF files ought to be closed however certain situations, such as an interrupted file transfer, can leave an “open” file that is still partly usable. The ability of a device to be able to handle “open” MXF files is an application issue.

In an ideal world, the two states of “open” and “closed” would be sufficient to describe all the files in existence. The desire for simple hardware and software, however, means that some capture devices and applications will not be able to parse the wide variety of essence types they might expect to place in an MXF file. To cope with this condition, the states “complete” and “incomplete” have been defined in SMPTE ST 377-1 to mark the status of the essence descriptor(s) in the header metadata of an MXF file.

These states are summarized below:

Complete – Each of the properties in the header metadata with a status of “required” or “best effort” are present in the file and are correct. The status of each of the properties is defined in SMPTE ST 377-1.

Incomplete – One or more properties within in the header metadata with a status of “best effort” has a distinguished value. The distinguished value is used to mark the property as “unknown at the time of writing”. An MXF file can still be a closed file because all the other properties of the file are known. Some of the header metadata can be incomplete due to the absence of an essence parser at the time of file creation. This allows an application to report many of the metadata properties of the file, but certain essence decoders might need to parse portions of the file before it is playable.

Maximum robustness is achieved when applications and devices create Closed and Complete MXF files.

Each partition starts with a partition pack that defines what sort of partition it is, followed by the following optional items:

Header metadata

Index table segment(s)

Essence container data

From these and other restrictions, a user can limit an MXF partition to contain only a single “thing”; i.e., a single essence type with its associated index table segments. If different essence containers need to be multiplexed together within the file, then a new partition can be started when the essence container changes.

4.2.2.1 Other Partition Kinds

MXF can be extended to include other kinds of file partitions. For example, SMPTE ST 410 defines a Generic Stream partition to allow large data streams to be carried in MXF files. An example use-case is where there is a need to carry non-KLV coded data such as an XML file. SMPTE RP 2057 defines a means of carrying text-based metadata in a Generic Stream partition.

4.3 KLV Coding in MXF

MXF files use key-length-value (KLV) coding throughout for flexibility and extensibility. KLV coding is defined in SMPTE ST 336. This mechanism is used to encapsulate the individual elements of an MXF file in such a way that devices can ignore information when the key of a KLV triplet is unknown. The length parameter tells the user how much data is to be used or ignored.

Note: A full review was published in the July 2000 edition of the SMPTE Journal (Vol. 109, No 7, Engineering Report).

Every item of metadata and essence in an MXF file has a **key** (as a 16-byte value) and a **length** that defines how long the value of the item is, followed by the **value** of the item. The key is a SMPTE Universal label and follows the rules defined in SMPTE ST 298.

SMPTE ST 336 includes not just the encapsulation of individual data items, but also the encapsulation of collections of individually coded KLV data items into logical data sets and packs. MXF sets are KLV coded

objects that contain a group of individually coded metadata items in a non-ordered sequence whereas MXF packs are KLV coded objects that contain a group of individual metadata items in a specified ordered sequence.

A decoder that does not recognize a key is able to skip over the unknown (dark) value and inspect the next key. This allows extra functionality to be added to the MXF specification at a later date, knowing that older decoders will be able skip over unknown values.

Individual data words within the key are ISO object identifiers (OID) using primitive BER (basic encoding rules: ISO/IEC 8825-1 ASN.1). Descriptions of primitive and constructed encoding of ISO object identifiers can be found in SMPTE ST 298, Annexes D and E.

In summary, BER OID encoding creates an identifier from a series of one or more sub-identifiers, none of which can be zero. The first zero sub-identifier is interpreted as the termination of the identifier. Thus if it is desired to make a sub-identifier from a series of bit flags, at least one bit needs to be a fixed '1' to prevent misinterpretation of the sub-id as a terminator if all the other bits are '0'. Any of bits 6 through 0 can be used as the fixed '1' bit, often using either b0 or b6.

This means that the most significant bit of each 8-bit value is a flag to say that the word is greater than a 7-bit value and therefore needs to be followed by more bits. For example, if the 12-bit value **b** (b_{11} .. b_0) is to be mapped into a KLV key, then Figure 3 illustrates a possible mapping into successive bytes X and Y of a 16-byte key:

Byte No. bit	X 7	X 6	X 5	X 4	X 3	X 2	X 1	X 0	Y 7	Y 6	Y 5	Y 4	Y 3	Y 2	Y 1	Y 0
Value	1	0	b_{11}	b_{10}	b_9	b_8	b_7	b_6	0	1	b_5	b_4	b_3	b_2	b_1	b_0

Figure 3 – Example of BER OID encoding

Figure 3 shows a binary 1 in bit 7 of byte X to indicate that this is a multi-byte value and a binary 0 in bit 7 of byte Y to indicate that this is the last byte of the multi-byte value. A byte value of 0 is often used to terminate a label so, for example, setting bit 6 of byte 2 to a value of 1 can be used to prevent accidental termination from occurring in the case where bits 5~0 are all zero.

Note that the actual mapping of bits into a label key needs to be normatively defined in an appropriate document. Note also that, at the time of writing this guideline, this multi-byte OID technique is not in use in any of the MXF specifications, although it is permitted in the SMPTE UL specification as defined by SMPTE ST 298.

MXF parser writers need to be aware that this technique could be used in the future so that, instead of, for example, using 3 single-byte OIDs, a single 3-byte OID could be used. Note that the number of words in other identifiers could be less than 16, or there could be a requirement for 16 bytes of which the final words are set to 0. As an example, where the length of SMPTE keys is fixed at 16 bytes, any unused bytes at the end of a SMPTE key value need to be set to zero.

The **Length** field is BER (Basic Encoding Rules: ISO/IEC 8825-1 ASN.1) coded and is neither binary nor a fixed number of bytes. BER coding allows the length field to have a variable number of bytes. So how do you know the length of the length field?

The length field is always coded MSB (most significant byte) first. If bit 7 of the first byte is a '0' then the 7 least significant bits contains the length value (0 .. 127). If bit 7 of the first byte is a '1' then the 7 least significant bits tell you the number of bytes in the length field; e.g., the value '83_h' means that the next 3 bytes contain the length field. The format document gives recommendations for the upper limit of the length field. Decoders need to be able to handle both short form (1 byte) and long form (>1 byte) BER coding.

The examples below show a length value of 64 coded in the three different ways:

40_h	short form coded
83.00.00.40	long form coding using 4 bytes overall
87.00.00.00.00.00.00.40	long form coding using 8 bytes overall

In specialized operational patterns (see Section 9.6.1), the file is allowed to start with a non-KLV run-in. This is to permit synchronization bytes or other bytes to be added at the front of the file for certain (limited) applications. In all other circumstances, there will be no run-in and the entire file must consist only of KLV elements with NO gaps.

4.4 MXF Metadata

MXF files using the data model allow it to represent a file interchange data stream. This means it can represent an output timeline that has video, audio and data essence content. It has a logical metadata structure, a defined physical representation (KLV) and is interoperable with other MXF systems. It has been designed so that specific editing systems can open an MXF file without modification to either the MXF file or the editing system.

In practice, this means that there is a lot of overlap between MXF and editing functionality. MXF is targeted at file interchange throughout the broadcast and content creation chain, whereas editing tools are optimized for the multiple generations used in content creation and post production. As a rough rule of thumb, essence content interchange, cut-edit functionality or simpler is an MXF application whereas editing is more appropriate for more complex functions.

Data modeling arranges the file format in terms of objects within an object model. These objects are chosen and defined to reflect the actual processes and content items that go into the MXF file. An IT engineer would take great care before modifying an object model to remove things that looked like they were not needed - the implications for future enhancement and interoperability might be serious and not always obvious. Likewise for the data model of the MXF header metadata.

4.4.1 Header Metadata Data Model

The structure of the MXF header metadata is defined by an object-oriented data model.

Section 7 of this document gives an outline of the header metadata concepts and how the data model describes the file essence. More details about this data model approach are given in SMPTE EG 42 with a specific focus on descriptive metadata.

4.4.2 Metadata Placement

The MXF model creates “hooks” on which the metadata can be placed. These hooks live in the file header, the body partitions, some essence containers and the file footer.

The header metadata area provides for the structural metadata that describes the essence container parameters, but it can also contain descriptive metadata that allows the content to be described. For example production, clip and scene information is described in the MXF descriptive metadata scheme 1 document (part 5 of the MXF specification).

The essence container can also contain timeline-oriented metadata such as time-code and, potentially, pan-scan metadata. Such timeline-oriented metadata can be present either within the essence data or within the essence container.

There are certain metadata parameters that might live in multiple places. The most obvious of these is time code. This can exist in the header metadata, but might also live embedded within the essence container data; e.g., in the GOP header of an MPEG essence container. This repetition is often important and the handling of any conflict between the different instances of the data is application dependent.

Although only occupying a small fraction of the size of a typical MXF file, the header metadata is often, for those inexperienced in formal data models, the most difficult part to understand. The following sections introduce the metadata categories and the topic of object-oriented coding in a general and easy to understand manner. For a more rigorous explanation, there are many reference books that cover the principles and use in far more detail than given here.

4.4.3 Header Metadata Categories

Broadly speaking, the metadata items can be split into two categories: structural metadata and descriptive metadata. The structural metadata is intended to bind the different elements of the file together and is needed to define the basic file structure. The descriptive metadata is intended to supply extra information about the file, such as a program name or scene description.

There are a large number of metadata elements defined in SMPTE ST 377-1. To understand the restrictions on the use of metadata elements, it is necessary to understand the terminology described in Section 7.

4.4.3.1 Structural Metadata

MXF structural metadata is used to describe different essence types and their relationships along a timeline. The MXF structural metadata defines the way in which the output timeline of the file relates to the one or more stored top-level file packages. It defines the synchronization of different tracks along a timeline. It also defines the picture size, picture rate, aspect ratio, audio sampling and other essence description parameters.

The structural metadata is defined in SMPTE ST 377-1. Most of the parameters are defined in the MXF file format document, but additional descriptors and labels can be defined in individual essence mapping documents. More information on the structural metadata concepts appear later in this document.

4.4.3.2 Descriptive metadata

MXF descriptive metadata comprises information in addition to the structure of the MXF File. This can be intended for human use (as in the majority of the SMPTE ST 380, the MXF DMS-1 specification) or it can be information for machine use, such as a track of information containing depth information for 3D processing.

SMPTE ST 377-1 provides a plug-in mechanism that allows different Metadata sets to be defined and used in an MXF environment. SMPTE ST 377-1 also provides mechanisms for uniquely identifying the metadata scheme(s) present in the file, mechanisms for preventing numerical conflict with existing metadata and a mechanism for determining the version of the descriptive metadata specification used.

No single metadata definition and structure will be appropriate for everyone, so a mechanism was added to allow the integration of other metadata schemes without redeveloping applications and equipment. The MXF plug-in mechanism is lightweight and allows versatility and extensibility for the users.

When descriptive metadata is added using the plug-in mechanism, many of the features of MXF are achieved automatically. The ability to create multiple tracks and synchronize them against each other, the ability to add metadata events synchronized with the video / audio or other tracks and the ability to use metadata in the output timeline that was available in the source file are all part of the standard MXF feature set. This document will outline only the basics of a descriptive metadata scheme. A fuller treatment of the subject can be found in the descriptive metadata engineering guideline, SMPTE EG 42. It is worth noting that descriptive metadata can be for both human and machine use. Much of the machine-descriptive metadata relates to special properties of the essence and has an intimate spatio-temporal relationship to the essence.

4.4.3.3 Application Specific Metadata

SMPTE ST 377-1 provides a consistent method for users to formally add extensions by enabling application specific metadata to be added to the header metadata. This method also provides for the simple removal of such application specific metadata if that is needed. This provision has some parallels with the addition of descriptive metadata in that there are MXF sets that provide the hooks to add (or remove) application specific metadata in the same way as for descriptive metadata.

An application specific plug-in can provide MXF files with additional information not present in the MXF specification that describes some new feature not present in regular MXF files. This additional metadata is provided in the form of additional properties to existing MXF sets and additional sets to the MXF structural metadata. When application specific metadata is added, there are corresponding identifiers that provide decoders with information about the application specific metadata that has been added. This differs from the simple addition of optional properties to existing sets in that a decoder can simply ignore optional properties on the basis that they add no value to the header metadata.

As per descriptive metadata, application specific metadata is always defined by another standard and it provides identification in the form of a SMPTE universal label that uniquely identifies the application specific application metadata scheme together with an application metadata specification that acts as the common gateway for all application metadata schemes.

4.4.4 Dark metadata

Dark metadata is the term given to metadata that is unknown by an application. This metadata can be additional properties added to SMPTE ST 377-1 or metadata that is part of the MXF standard, but not relevant to the application processing the MXF file. It can also be privately defined and generated metadata. It is important that there are some rules on the use of dark metadata to prevent numerical or namespace clashes when private metadata is added to a file that already contains dark metadata. Rules for handling dark metadata are defined in SMPTE ST 377-1.

4.5 MXF Essence

4.5.1 Representing Time and Timelines with Tracks

Time and timelines are features of the logical representation of an MXF file. The concept of time within the file is independent of the arrangement of bytes within the file, although constraints can be applied in order to get certain functionality from the file stream. Time is used to measure the duration of the content as well as to synchronize the content. In MXF a “track” is used to represent the passage of time. A track has units to represent time and has an associated duration property. Some tracks have segments that butt against each other to form a continuous sequence of video (timeline tracks) whereas others can have overlapping events that refer to the point at which descriptive metadata is valid (event tracks). The mechanism for adding descriptive metadata definitions to an MXF file is to add new tracks on which to “hang” the metadata items.

To synchronize two or more tracks, they need to be somehow related. This is done by placing multiple tracks within a package (a container for tracks) which synchronizes the start and duration of each track. Note, however, that tracks can have different time measurement units within the package. Time is normalized within a track by its “edit rate” property. This in turn gives us an edit unit, of 1/edit rate.

4.5.1.1 What are the units of time?

There are two main units of time used with an MXF file. These are:

Edit Units = 1/Edit Rate – used to mark time along a track

Sample Units = 1/Sample Rate – used to describe the underlying sample rate of the essence

Edit units can be chosen for the convenience of the file writer, whereas sample units define the sampling structure of the essence. A sequence of audio samples, for example can have a sample rate

of 48 kHz, whereas the track that describes the sequence can have an edit rate of 50 Hz so that synchronization with parallel tracks is numerically simplified. For video streams, the sample rate is usually defined as the field or frame rate of the content and not the sampling clock.

4.5.2 Essence containers

An essence container defines the encapsulation of a particular type of essence. Its purpose is to allow the essence content to be wrapped in KLV and to have associated with it an optional index table to allow rapid access to a given time offset within the essence. The essence container is structured to allow easy multiplexing with other essence containers and to allow identification of the decoding requirements needed to display, listen to, play, or execute the essence content.

An essence container specification defines a unique SMPTE label for identification as well as a method for encapsulating the essence in a KLV structure. Different essence containers can place restrictions on the interleaving of the essence data to be compatible with existing applications. The SMPTE label allows decoders to make a fast go/no-go check of the essence type at the very beginning of the file.

An MXF file can have more than one essence container. The precise number of essence containers and their relationships is constrained by the operational pattern with which the file complies.

The “generic container” is an MXF essence container defined as a SMPTE standard. This is intended to carry all the mainstream essence types in existence at the time of creating SMPTE ST 377-1. It is very simple in operation, yet flexible enough to carry uncompressed material as well as reordered MPEG compressed material. Associated with the generic container are a number of SMPTE container mapping standards that define how the actual essence byte stream is placed in the generic container.

4.5.2.1 Essence container specification

There are several specific questions that need to be asked when putting an essence container into an MXF file. These are notably:

1. What limitations are placed on the essence container when it is in an MXF file?
2. Are there interleaved variants of the essence container?
3. How are the contents of the essence container KLV coded?
4. How are the essence containers padded to fit the chosen KAG size?
5. What is done with metadata embedded within the essence container?
6. How are index tables used with the essence container?

The essence container and mapping specifications provide the answers to these questions. It is the intention of the essence container and mapping documents to restrict the choices of an essence container implementation sufficiently to allow interoperability between devices, yet with enough flexibility to solve real world problems.

4.6 MXF File Tools

This sub-section identifies the tools provided by MXF. In some cases, the description provided here is complete. Other cases describe only the overview and leave details to a later section.

4.6.1 Index tables

MXF files can include an index table that provides rapid conversion from sample-based indexes (e.g., time code) into byte offsets within an essence container. The index table can be segmented and can be stored before, after or multiplexed with the essence data segments. Index table segments can be placed in file partitions (see Section 4.2.1) to limit memory requirements or to provide per-stream index tables that are position independent within a file.

An index table provides for random access within an MXF file. Specifically, it allows random access by a time index. This means that if you want to access the picture, sound or data essence located at, for

example, 10 seconds into the essence container, then an index table will provide the translation between the time value and the byte offset within the essence container. MXF index tables are quite complex because the format is designed to cope with interleaved essence containers that can be constant or variable bit rate and that also can be temporally reordered on the disk compared to the presentation order (e.g., long GOP MPEG-2 files).

SMPTE 377M-2004 limited the length of index tables to 64KB with a 2-byte length field and this required some files to employ index table segments. SMPTE ST 377-1:2011 provides for the use of a 4-byte index table length field that provided for large index tables without the need to provide index table segments.

4.6.2 Random index pack

The random index pack (RIP) is optional but, where present, is always found at the end of an MXF file where it provides a list of the positions of all the partitions within the file and contains a mechanism for quickly determining its own existence.

This differs from an index table, which provides the byte offsets of the essence content within a partition. The difference can be clearly seen when two different essence containers are multiplexed together. There will be two separate index tables, each of which contains conversions between temporal offsets and byte offsets within each essence container. In contrast, the RIP gives absolute positions of all the partitions within a file, which can then be used to locate index tables without having to parse the entire file.

4.6.3 Operational patterns overview

An operational pattern (OP) is used to constrain MXF complexity and defined in a SMPTE standard. The generalized operational patterns are intended to split the complexity depending on the complexity of processing required by an MXF decoder. Specialized operational patterns are likely to be created in order to constrain MXF for a particular ‘application space’. Usually an MXF file is interchanged for a purpose. This can be the exchange of an ingested clip, a camera output, a finished program or the interchange of a partially edited program. These requirements have different implications for the structure of the file. Different operational patterns define the structural metadata that is required to satisfy a particular application. In general, the higher the number of the operational pattern, the more complex the file and the more functionality is required in the decoder. Simple operational patterns such as OP1a can be used with both linear and non-linear access devices. Some more complex operational patterns require non-linear access devices.

Each operational pattern has an assigned SMPTE label value that allows MXF decoders to quickly establish the complexity of an MXF file.

More detailed information about operational patterns is given in Section 9.6.

4.6.4 KLV alignment grid

The KLV alignment grid (KAG) can be thought of as gridlines spaced on uniform byte boundaries in each partition. To achieve good performance, all the important KLV items within the file (header metadata, content packages of the essence, etc.) must line up on the grid. This means that the first byte of the key needs to be on a grid boundary. Note that a device or application must always be able to read an MXF file regardless of whether the elements within the file fell on rigid byte boundaries within the file.

The reference point for a KAG is the first byte of the key of a partition pack, and the KAG value is valid within the partition. SMPTE ST 377-1 states:

“*The first gridline in any partition shall be the first byte of the key of the partition pack that defines that partition.*”

In order to have a global KAG value, each and every partition pack needs to have the same KAG value. Additionally, to maintain this global KAG value, the first byte of each and every partition pack

needs to lie on a KAG boundary. Finally, if there is a run-in, its length in bytes needs to be an integer multiple of the KAG value.

The KAG is a performance enhancer because it reduces the need to search every byte for the start of each file component. It is possible that some process could make a change to a file that breaks the KAG rules, but is unable to modify the KAG value in the partition header. An MXF decoder that is receiving a file can benefit from a certain KAG value because its internal storage is arranged on rigid boundaries. But it needs to continue to check each of the KLV triplets received for confirmation that they still lie on the KAG. The majority of files that use the KAG feature will respect the value in the partition header, but some do not or cannot. The MXF application receiving the file that does not respect the KAG ought not fail under this condition, but performance could be severely restricted. For example, the receiving application can choose to process the incoming stream to force it to be aligned to the KAG by inserting fill KLVs. This can slow it down and cause it to recalculate index tables.

4.6.5 MXF references

MXF uses different referencing mechanisms for different purposes. One example that could cause confusion is the difference between references to the top-level “file package” and “the essence”. The MXF content storage set uses Instance UIDs to reference all the packages in an MXF file. One of these will match the Instance UID of a file package within the file. This is a strong reference to the package. The package itself is a description of the essence, but is not the essence itself.

The content storage set also uses Instance UIDs to keep a list of essence container sets. These are used to group the various IDs that enable an MXF decoder to work out which partitions and index tables relate to which top-level file package.

This determines how a material package SourceClip references the essence. This structure does not use instance UID values as a reference, but the 32-byte UMID of the essence. This is because the material package is referencing the essence of which the top-level file package is a description and the essence is identified by a SMPTE UMID.

More detailed information about MXF references is given in Section 7.4.

5 MXF Interoperability

5.1 Working with Stream Interfaces

MXF files will often be processed in streaming environments. This will include streaming to and from videotape and data tape, and transmission over unidirectional links or links with a narrow-band return-channel.

In these environments it is impractical to rewind the stream to update parameter values so files need to be written sequentially. This implies that the minimum buffer size and latency are determined by (among other things) the maximum KLV packet size. Use of MXF streaming needs to take into account all the constraints of the operational pattern [see Section 9.6] in use, as well as extra restrictions imposed by the particular streaming data link before recommending buffer sizes or latency requirements.

Sequential writing is necessary when source or link or destination operate only in streaming mode. Random access writing is permissible before or after data transfer, for example, to optimize downstream access performance.

Operational pattern labels have special qualifier bits that indicate whether the file has been created for streaming.

5.1.1 MXF interoperation with stream interfaces

MXF files can be directly created from standardized formats such as MPEG-2 system and elementary streams, AES3 data streams and DV DIF packet streams. These formats can be mapped from one of

several real-time (or quasi real-time) interfaces such as SMPTE ST 259 (SDI), SMPTE ST 305.2 (SDTI), SMPTE ST 292 (HD-SDI), or transport interfaces with real-time protocols such as IEEE-1394, ATM, IEEE802.3 (ethernet), ANSI fiber channel and more.

When a streaming file is captured, a file header is created and the essence is KLV wrapped on the fly. The data rate increases due to the KLV wrapping and addition of headers. Real time streaming devices will need to ensure that any buffering requirements of a streaming interface are catered for with this change of data rate.

Conversion to and from the source format is always possible, but sometimes there will be loss of information. Not all streaming and storage formats are able to store the rich metadata constructs available in an MXF file. Often there will be a lossy data mapping where information in one format cannot be represented in the other. Eliminating this undesired loss is a function of the systems engineering that interconnects MXF and non-MXF systems.

5.1.2 Stream recovery in MXF files

Streaming environments also impose requirements for recovery and re-synchronization in several different circumstances such as:

1. When a packet or other data block is lost.
2. When a decoder joins a transfer that is already in progress.
3. When a transfer or partial transfer is restarted.
4. When it is necessary to access or retransmit a file that is still being received (“pre-play”).
5. When overall metadata is modified during the time of transfer.

The first of these (packet loss) usually requires a return-channel or forward error correction for effective protection. The other circumstances can be addressed by judicious design of the stream format to allow for re-synchronization points and for repetition of important metadata.

5.2 MXF Interoperability with Other Files

MXF files employ a data structure together with a set of constraints and plug-ins to create files that can be directly written and read by other systems. MXF is also able to inter-operate with other file formats by utilizing techniques such as referencing external essence and using the run-in to provide extraneous non-MXF data. Different metadata models can be plugged into the MXF file format to provide extensions and the KLV structure itself can be converted to formats such as XML for exporting MXF data to other systems.

6 A Guide to the Wording of the MXF Standards

MXF files use a data model for the header metadata. Because of this, many of the words used in the MXF standards can be unfamiliar to users familiar with more traditional tape and disc storage systems. In all MXF documents, normative terms are defined within the document and are self-consistent. The main glossary of terms and data types can be found at the start of SMPTE ST 377-1.

6.1 Conformance Text

All new and recently revised SMPTE documents now contain a conformance ‘boiler-plate’ which defines the conformance language and its use. This provides for a more rigorous definition of a given standard.

If you are using an original version of a SMPTE MXF standard, then you need to ensure you are still compliant with the latest version of that standard. The original MXF documents were developed without the conformance ‘boiler-plate’ and, compared to the most recent revision, could be less rigorous about the implementation requirements.

6.2 Functional Descriptions — Encoder Required, etc.

The following terms describe functionality that needs to be supported in order to create an interoperable MXF environment. SMPTE ST 377-1 defines these normative terms. Extra text and words given here are for guidance only.

6.2.1 Required

A **required** item is essential to both encoder and decoder. An example of a required metadata item is a preface set. The encoder needs to encode this and the decoder needs to understand it and act on it.

6.2.2 Encoder required

An **encoder required** item must be encoded, but a decoder can choose to ignore it. An encoder ought not to assume that a decoder has taken notice of such an item.

6.2.3 Decoder required

A **decoder required** item can be sent by the encoder. If sent, the decoder must act upon the item. If not sent, then the decoder can either do nothing, or set the item to a default value or take a predefined default action if specified by the relevant document.

6.2.4 Optional

An **optional** item can be sent by the encoder if it is known. If sent, the decoder can choose to ignore the item. If not sent, then the decoder can either do nothing, or set the item to a default value or take a predefined default action if specified by the relevant document.

6.2.5 Best effort

A **best effort** Item is very important to a decoder, but cannot be known by the encoder at the time of file creation. These items have distinguished values that mark them as not known such that when these distinguished values are used, the file becomes an “incomplete” file as explained in Section 4.2.2.

Note that a ‘default’ value for an item is the value that a decoder uses in the absence of the item. A ‘distinguished’ value is used by an encoder to signal that the item value is unknown by the encoder. The difference between ‘default’ and ‘distinguished’ is important.

6.2.6 Dark

A **dark** item is one that is unknown by a decoder or an encoder. This item can be proprietary and is unknowable by a decoder. It can be an extension to SMPTE ST 377-1 that has not been incorporated into a device or application. It can even be metadata in the original specification that is not relevant to a device or application. All that is certain is that the meaning of the metadata is unknown. In certain application environments, encoders can be required to carry dark metadata and decoder can be required to make dark metadata available.

6.2.7 Incompatible

An encoder must not send incompatible items. This data classification is provided to allow certain data items to be forbidden if they could prevent successful or deterministic decoding. There are no “incompatible” items defined within SMPTE ST 377-1, but the concept of “incompatible” items gives a common term for users to describe a class of essence data or metadata that needs to be avoided.

6.3 Encoding and Decoding Concepts

One of the key points of developing any new work is to consider the layering of any file format and its contents. This helps us to understand the meaning of an ‘encoder’ and a ‘decoder’ at any given layer. Unfortunately attempts to introduce words such as “encapsulate” have not been well accepted and words such as “encoder” are forced to have slightly different meanings depending on context.

The layers for encoders and decoders can be broken down as follows:

Table 1 – Content layering

Layer	File Body	File Header & Footer
Application	Source Coding (e.g. 486, 576, 720, 1080 active video lines etc.)	Data Interpretation (e.g. dictionary of data definitions)
Essence Coding	Compression Coding (e.g. MPEG, DV etc.)	Data Communication (e.g. relationships between objects)
Container	Essence Container (e.g. ES, PS & TS for MPEG, DIF for DV)	Data Container (e.g. KLV sets, Objects)
Encapsulation	MXF coding (KLV)	
Transport	Transport (IP packets, SDI, etc.)	

The overall system can be summarized as follows:

1. A MXF system *accepts* essence represented in its "source coded format".
2. The essence is optionally *compressed* through a source **encoder**.
3. The essence components are *multiplexed* by an MXF **encoder** into partitions.
4. The multiplexed partitions are *encoded* into an MXF file by an MXF **encoder**.
5. The MXF file is *decoded* by an MXF **decoder** to present the essence to a user.
6. The MXF file is *demultiplexed* by an MXF **decoder** to split the file into its different essence components.
7. The encoded essence is *decompressed* by an essence **decoder**.
8. The decompressed essence is *displayed* or *presented* in its "source coded format".

Note: Processes are *italicized*, nouns are in **bold**.

Not all processes will be supported by all equipment. Many devices will operate over all layers to provide a network or stream interface at the lowest layer, and an interface to the user at the highest layer. However, devices that simply ‘store and forward’ need only respond to the lowest two layers and devices that ‘unwrap’ the data contents to provide the raw data streams only respond to the lowest three layers.

6.4 Essential Terms

This section illustrates the use of essential terms, including: element, item, container, stream, body, partitions, multiplexing, interleaving and more.

An MXF file can reference external essence in addition to including essence within the MXF file body. The MXF file body can have several essence containers that are multiplexed together, each of which can sometimes be called a stream. Each of these essence containers can have a single piece of essence or can have different essence elements that are interleaved together. Each of these elements can be categorized into picture items, audio items, data items and system items. This results in an MXF file body that can contain a multiplex of essence containers that in turn contain interleaved essence items that in turn contain the individual interleaved essence elements.

That might be difficult to comprehend, so what follows is an example that will help to clarify this extreme example of MXF capabilities. Imagine a file that was captured in DV and has had its stereo sound extracted and separately edited. Later in the process, an orchestral score was added in using a separate essence container and described by a separate file package. The operational pattern 1b mechanism is used to synchronize the two file packages. The resulting file looks logically like Figure 4.

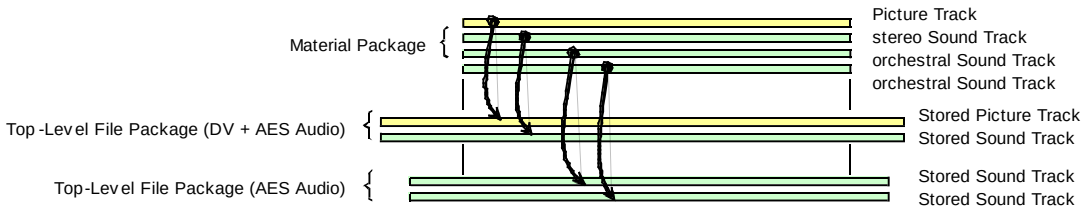

Figure 4 – Multiplexing and interleaving — Logical view

This seems a simple logical view, but the physical representation is much more complex as shown in Figure 5.

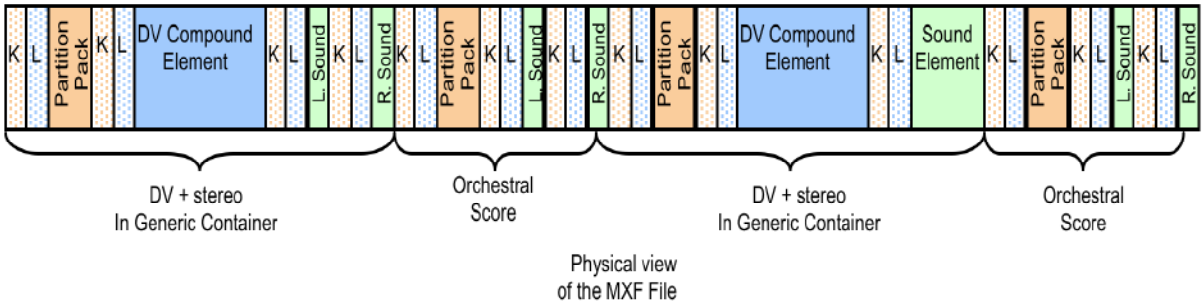

Figure 5 – Multiplexing and interleaving — Physical view

The final sentence of the second paragraph of this section can now be better understood. The file is a multiplex of different partitions; in this case two generic containers are multiplexed using the partition mechanism. One of these generic containers uses an interleaving of essence items — a DV compound item and a sound Item. In each of the multiplexed generic containers, the sound items contain an interleaving of sound elements — the left and right channels.

It is also worth noting that DV itself (per the example of the third paragraph) is an intrinsic interleave of DV-DIF blocks. In most MXF processes, this level of interleaving is left to the essence codec and is usually opaque to MXF.

There are normative descriptions of these words in the MXF format document and in the MXF generic container documents.

6.4.1 Essence Element

In many places in SMPTE ST 377-1 documents, the term essence element is used generically.

Where discussing low level wrapping of the data in a generic container mapping, the term essence element is used to mean “a KLV wrapped essence entity that has a defined key”. For any given key, any essence elements with that key relate to the same essence stream.

In other, macroscopic, discussions of interleaving and multiplexing, the term essence element is used to describe all the KLV wrapped essence entities with a given key, such as a single video data stream. When a stream has a single video data stream and an associated audio data stream, the essence container would be regarded as having two essence elements, regardless of how many KLV packets were used to hold the essence.

This contextual use of the term essence element can cause confusion, but it was considered worse to try to invent new contextual terms for any terms that have subtle changes depending on the context.

7 Metadata Classifications and Placement

The main objective of the material exchange format is to exchange program material together with attached metadata information. The metadata information is intended to accurately describe the essence either within the file or referenced by the file.

This section provides a very brief overview of some of the underlying concepts of metadata as it is used within an MXF file. A fuller description of the use of MXF descriptive metadata can be found in the SMPTE EG 42.

In general terms, the use of metadata has many dimensions as follows:

1. It is in widespread use within different content-based industries, including broadcast, film, music and web authoring.

2. It is in widespread use in different content-based applications, including capture/creation, production, post-production and archive/libraries.

3. It can be divided into several different broad categories including business transactions, publication information, content identification and labeling, compositional information and formatting, etc.

4. It can have different states such as being static for the file, static for a defined duration and dynamic (with several kinds of dynamic including transitory, metronomic, incrementing and so on).

5. It can have different levels of stability with elements having durable values that remain stable throughout the file or transient values that can frequently change over time.

Metadata in MXF files can be divided into four broad categories:

1. Structural metadata: A set of information that defines the essence structure; i.e., how the essence was edited and what source components were included in what derivation chain.

2. Descriptive metadata: A set of information that describes, parameterizes or catalogues content, such as episode number, copyright holder and so on.

3. Application metadata: A set of information that can be added to a file to extend the structural metadata in a way that allows its easy removal at a later date.

4. Dark metadata: Unknown to an application at the time of processing. This can be for many reasons including private metadata, unknown extensions to MXF and standardized metadata items that are not handled by the application.

MXF provides the ability to bind metadata, essence and data essence streams together via the structural metadata. MXF also provides a descriptive metadata (DM) mechanism that allows independent DM schemes to be created as plugs into the overall MXF file format.

The placement of metadata in a file can be in one or more of several possible locations most suited to the application of the particular metadata item. Figure 6 indicates several broad locations where metadata can be stored.

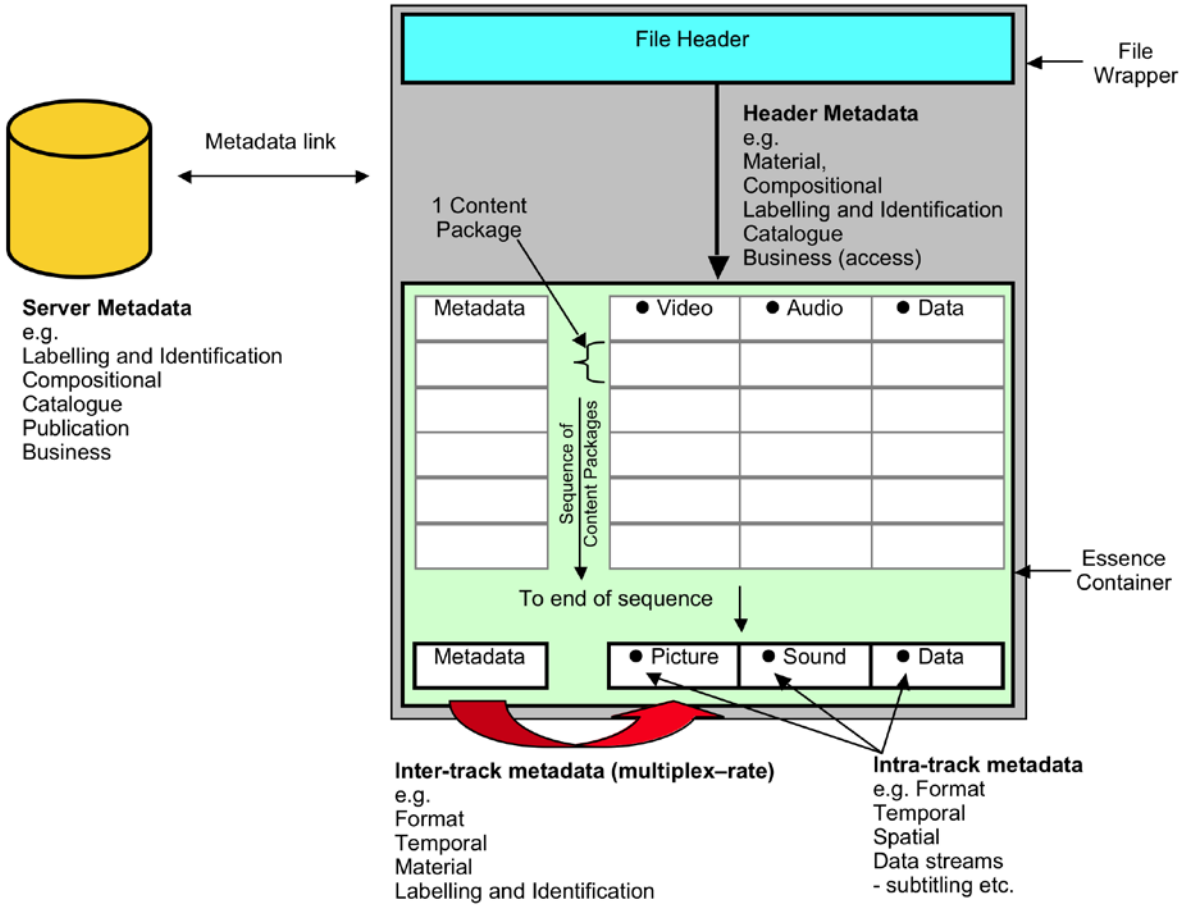

Figure 6 – Different locations for metadata storage

7.1 Embedded Metadata Location

Embedded metadata (intra-track in the figure above) is that which is tightly embedded in the essence stream such as is present in MPEG-2 Video ES and AES3 data. Metadata that is embedded is typically:

Format: for decoder operation;

Temporal: with particular reference to time-code;

Spatial: such as pan-scan vectors and aspect ratio;

Extra data: such as captioning, subtitles, etc.

7.2 Linked Metadata Location

Linked metadata (inter-track in the figure above) is that which is closely linked to the content, whether video, audio or data content, through a container on a picture-by-picture basis. This metadata is typically interleaved with the content and maintaining a tight timing relationship with it. As an example, the system item of SDTI-CP provides this metadata location. Metadata that can be stored as linked to the frame is that relating to:

Format: often as a duplicate of the embedded metadata;

Temporal: mostly as temporally variable metadata extra to any embedded metadata;

Material: including the extended UMID; and

Label: simple labeling of the content.

7.3 Attached Metadata Location

Attached metadata (header metadata) is that which can appear in a file header such as is present in MXF. It can encompass a wide variety of metadata; in particular:

Content: providing metadata about the content in the file body;

Compositional: providing simple or complex editing information for the clip or program;

Label: providing a full set of content labeling and identification;

Catalogue: for location of events, markers and for archival metadata; and

Business: for access and security information.

7.4 Server Metadata Location

Server metadata can be used to replicate almost all of the metadata described so far. However, it is particularly useful for the following metadata sets:

Label: providing a full set of content labeling and identification metadata;

Compositional: providing simple or complex editing information and historical derivation metadata;

Catalogue: for use in off-line searches;

Publication: defining when and where content is to be delivered; and

Business: for audience information, program statistics, etc.

8 MXF Header Metadata Concepts

In this section, the concept of header metadata objects will be introduced, as will the idea of collecting objects and information into packages. This section is intended to improve the reader's understanding of the concepts. It is not intended to be a rigorous definition of the terms. The actual definitions of packages, strong references and the like can be found in SMPTE ST 377-1 and other MXF documents.

Object orientated design is one approach to software design where a. a complex system is represented by describing each of its components as an independent object.

Section 9.5 explains how the header metadata packages and tracks describe to the contents of the MXF essence container.

8.1 What are Objects?

An object contains data and procedures grouped together as an entity. In MXF, objects are coded as KLV local sets as defined in SMPTE ST 336. In SMPTE ST 377-1, the word "set" is used in nearly all cases to encapsulate an object of a given class. The specification of the class properties is done using tables in the normative MXF documents, and the behavior is specified in the text of the document.

The quickest way of explaining objects is by means of an example. We will use the track object to describe a track in the essence container.

A track can be thought of as a linear progression of audio/video data with a finite duration and analogous to a track on a video recorder. It starts at the start; it ends at the end and it lasts for its duration. The start, end and duration are known as properties.

A feature of object oriented design is "inheritance". This permits description of different sorts of essence tracks which all share some common properties that they inherit from the parent super class, but have extra properties or functionality added to make them useful. For example, consider an event track. The linear progression of audio/video data along a track can now be marked with events. An

event can start at any point along the track. It can be instantaneous (i.e. no duration) or it can last for a defined time. Events can also overlap (though not on the same track).

Another sort of track is a timeline track. Similar to an event track, it starts at a certain time, ends at a certain time and therefore has a known duration. This track has a restricted functionality in that it only allows source clips to be placed on the track. All the source clips on the track must be contiguous, which means there are no overlaps and no gaps. Both timeline and event track types inherit properties and functionality from a common track class.

8.2 What is a Class?

A class is a definition of the behavior and properties of a generic entity. The definition of a class from which a MXF object is created comes from the data model. Generic classes with general functionality are defined. Classes with specific functionality then inherit the general class features. MXF applications populate these sets with values to create MXF objects in a file.

The textbook example is a given make and model of a car. All the cars from the same class have the same generic behavior and properties. When describing a particular car, all the properties (such as color, engine size) are given values. This is called an object or an instance of the class. The class definition includes all the core design parameters that are common to all instances. A given make and model of a car (i.e. the class) can be a blue or red but are still clearly the same car, except the color ‘property’ has been changed between the two objects.

Classes are defined by a set of data items, where each item is commonly called a property. When an instance is made from a class, it becomes an object and values are assigned to all the properties.

Modeling of a system can involve the creation of many similar classes. This document has described that there are different sorts of track. Each of these tracks has properties that are very similar. In modeling terms, there is an abstract super class that defines the common functionality of all the different tracks. Abstract means that the class is never used directly. Super class means that the purpose of this class is to create subclasses that add to all the properties of the super class. A generic track is an abstract super class (where the term “abstract” means that this class is never directly instantiated). A timeline track and an event track are two concrete subclasses that share all the common properties of the track class and have added their own specific properties and behaviors. Such concrete subclasses are coded as sets or packs.

8.3 What is a Package?

A package is the metadata description of an essence container for a number of tracks that represent the passage of time. The package mechanism allows related tracks to be “ganged” together in parallel. This allows metadata and essence to be synchronized to a common timeline.

Each package describes some aspect of the essence or data in a file and the different types of packages including the material package, the file package and the source package are explained here with the help of some real world analogies.

It is important to note that the tracks within a package are synchronized in time. This synchronization is determined by a specified offset value from the beginning of each track within the package.

8.3.1 What is the material package?

The material package is a metadata structure that generally describes the output timeline of the file. If you imagine the file being “played” in an MXF player, you would expect to see video, hear audio and view the data as though it were a tape in a VTR. The material package contains the “hooks” that allow this to happen. It contains timing information about the output — for example, how the time is measured. It contains information about the output tracks — how many and what format they take. It also provides hooks to say where the essence data comes from to fill these tracks (i.e., which top-level file packages).

Figure 7 illustrates how the material package can be viewed as a set of parallel tracks — one for each kind of essence in the output stream. There is metadata associated with the file that has a global scope, such as the name, the UMID, etc. Each track contains further metadata to describe the way in which the final output needs to be created from the top-level file packages.

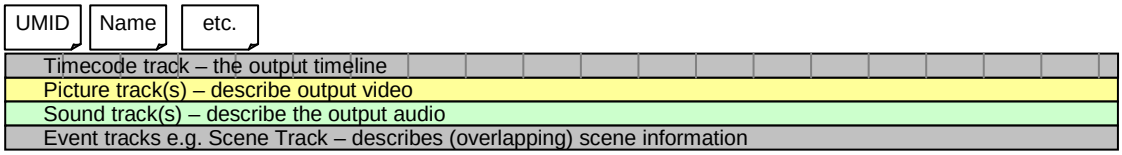

Figure 7 – Material package

Figure 8 shows the logical relationship between the pictures. It shows how the material package track can define a sequence of SourceClips. Each SourceClip in the material package indicates which portion of a top-level file package needs to be "played" next. This is the way in which MXF supports edit decision lists (EDLs).

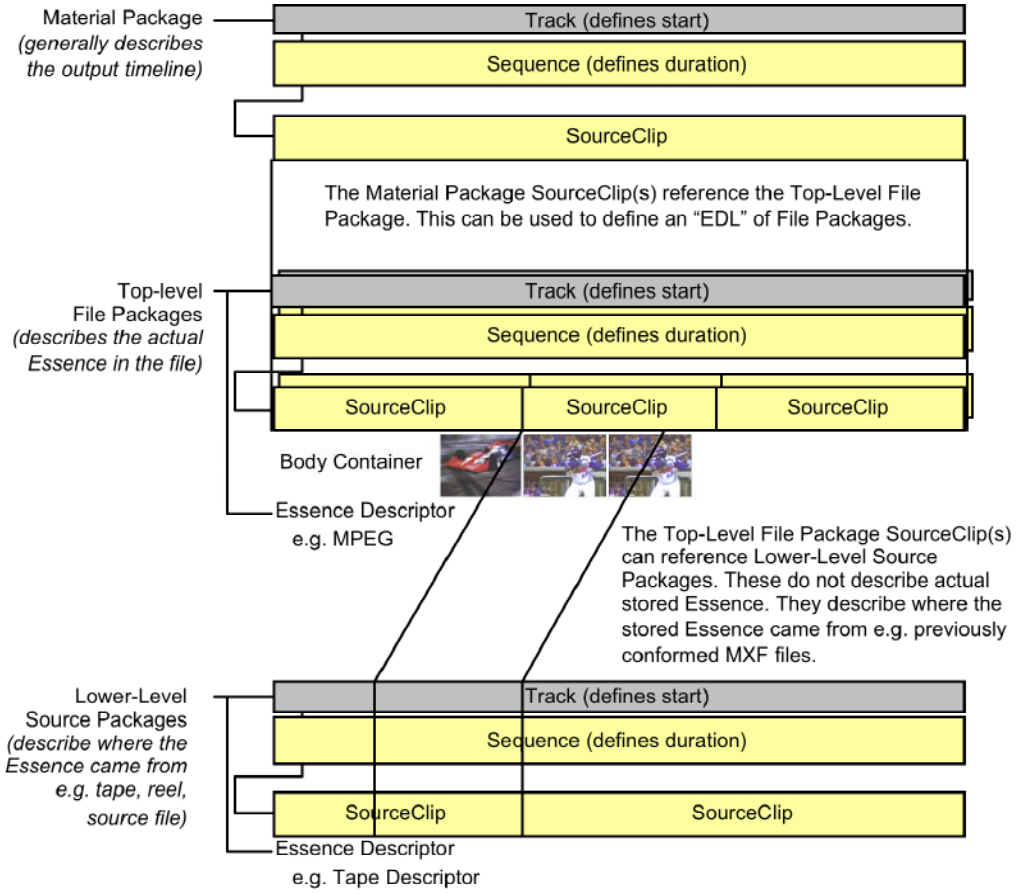

Figure 8 – Relationship between the packages

The material package in Figure 8 shows how the SourceClip references the entire top-level file package. Only the file packages in the top level of an MXF file describe the actual essence in the file body.

The MXF operational patterns constrain the relationships between the material package SourceClips and the file package(s) in an MXF file. In an OP1a file, there is no editing support and the material package references the entire top-level file package. In an OP3c file, complex timeline relationships are allowed that can require the MXF decoder to provide random access capabilities within the essence container.

8.3.2 What is a top-level file package?

A top-level file package contains a collection of metadata items and sets that represent the storage of some essence. It is described as though the essence tracks were in a file — hence the name top-level file package.

This essence can be stored in the file body, externally in a separate file (located by information in the essence descriptor) or even on external media (such as a disc or tape). The top-level file package contains the tracks that describe the type of essence, the compression scheme used (if any) and the source coding parameters such as the number of samples, pixels and aspect ratio of the essence as appropriate.

The tracks in the top-level file package can be made up from a number of SourceClips that are used as historical annotation to indicate where the content came from.

8.3.3 What are lower-level source packages?

The SourceClips in the top-level file package can refer to either file packages (for essence files) or physical packages (for tape or disc storage). In SMPTE ST 377-1, the generic class “source package” is used to refer to either file source packages or physical source packages.

In SMPTE ST 377-1, a lower-level source package (i.e. one that is not at the top level) is used to describe the derivation of the essence; i.e., where it came from. This metadata, provides historical information about the source of the file package. Lower-level source packages can contain either file descriptors that reference a file with the content or physical descriptors (such as tape descriptors) that refer to a physical location or storage medium for the content.

8.4 How are Packages Coded?

In MXF, each metadata object is coded and identified as a KLV local set and has a value that contains all the locally coded metadata items in sequence as a tag (typically 2 bytes), length (typically also 2 bytes) and the individual metadata item value.

A way of referring to objects is needed within MXF, and three options are provided. For example, the statement, “A material package has one time code track object”, is quite clear. This is known as a strong (one to one) reference between the material package and the time code track object.

Figure 9 illustrates that there were several options considered for coding data sets (i.e. objects) in MXF packages. The figure illustrates three sets in a package, which are a participant set, a person set and an organization set. The three methods considered for coding these sets are illustrated in the figure in the following order:

1. strong reference by embedding using hierarchical KLV group coding,
2. strong referencing by UID to link groups in a data stream and
3. weak referencing by UID to reference groups in a data stream.

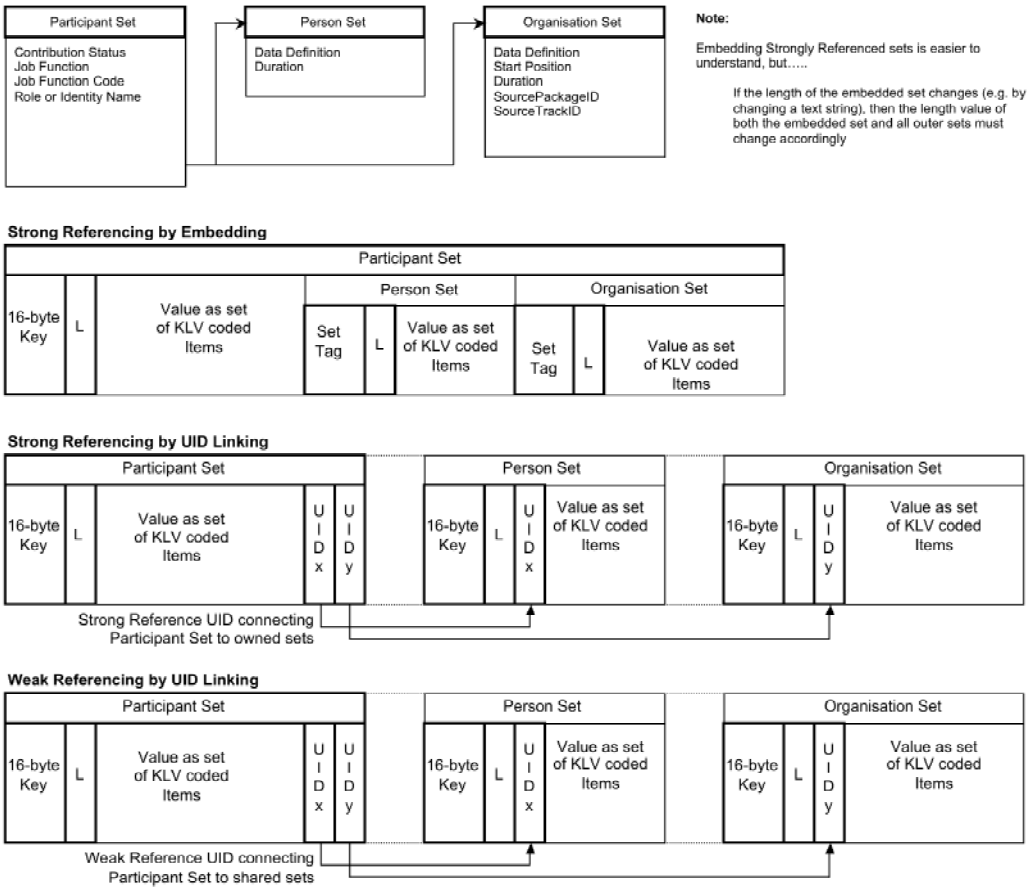

Figure 9 – Reference options using embedding and referencing

Strong referencing by embedding was not used for the MXF header metadata because any changes would require the entire header metadata structure to be re-built. For this reason, only UID references are used in the MXF header metadata.

The adopted data model uses only instance UID values as the single method of connecting sets with two variants – strong references and weak references as defined in Section 7.4.1.

MXF header metadata referencing is instantiated as a data sequence as illustrated in Figure 10.

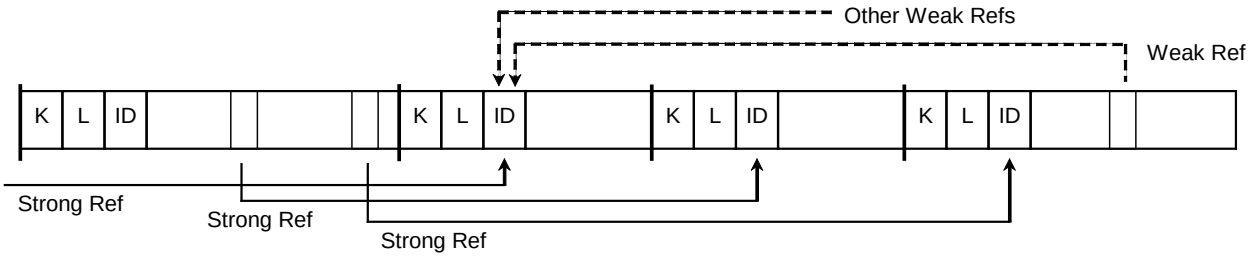

Figure 10 – Instantiating strong and weak referenced data sets in the KLV coded header metadata

Note that the metadata sets are contiguous in order to preserve the KLV coding protocol (i.e., there are no gaps between metadata sets).

8.4.1 Description of Referencing Methods

MXF header metadata uses several kinds of referencing methods, described as follows:

Strong references. Strong referencing implies ownership of the referenced object as well as a one to one relationship with it. This requires an instance UID property in the referenced data set and a property of type StrongRef in the referring data set. When an MXF application creates a tree of interlinked objects starting at the MXF preface set, all objects will have at least 1 strong reference so that they are “owned” and can fit into the overall tree. An object can additionally be weakly referenced by a large number of other objects.

Weak references. Weak references use an instance UID in the referenced data set; one or more other data sets can refer to the referenced data set by using the same weak reference UID value. The advantage of a weak reference is that the values of metadata items in a data set can be shared by several referring data sets. Everything within an MXF file that is the target of a weak reference needs also to be owned by one strong reference otherwise the target could be isolated (meaning that the target could be an orphan with no parental reference).

A **reference collection** is a list of UIDs connecting the referencing entity to zero or more other entities (either weak or strong).

A **reference array** is a set of ordered references (or vector). This implies that the order is significant for whatever reason.

Note that because all properties in MXF must be unique, all StrongRef and WeakRef properties are strongly typed. This means that the property can only have a StrongRef to a specific set type (or one of its subclasses). Thus, SMPTE ST 377-1 uses the nomenclature “StrongRef (MyClass)” to mean a strong reference only to an object of type “MyClass” or to an object derived from MyClass.

For every reference in an MXF file, an MXF decoder needs to be able to find a set that is the target of that reference. Why does the previous sentence use the phrase: “needs to”? From the definitions above, you would expect that a decoder would always be able to find the target of a strong reference. In the absence of any extensions to SMPTE ST 377-1, this would be a true statement. However, additions can be made to the format specification and additional metadata sets and schemes could be developed as the format matures. Decoders that do not recognize these extensions are likely to discover that there are dark metadata sets (i.e., any set key that is not understood by the decoder) within the file and that there are references without identifiable targets.

“Clever” decoders can help in this situation, by looking inside dark sets, especially those whose local tags appear to be stored in the primer pack. Instance UIDs could then be discovered with some high confidence and the presence of dark extensions to SMPTE ST 377-1 discovered. In some circumstances, this behavior can be quite helpful, but in general, making intelligent guesses about dark sets is outside the scope of SMPTE ST 377-1. It might also lead to unpredictable results.

To summarize the MXF referencing behavior:

1. References are made from a property in one set of type WeakRef or StrongRef to the InstanceUID property in another set.

2. All header metadata sets (other than the primer) are linked to the preface (directly or indirectly) by strong references.

3. All strong references in any instance of header metadata match one and only one set in that instance.

4. Weak references can be made to "global definitions" that are outside the file. In these cases the WeakRef will be either a UUID or a UL. Therefore, if a weak reference cannot be matched in the file it can be regarded as a global definition.

5. Typical global definitions are codec ULs, container ULs and compression ULs, which are used to enumerate different codec, container and compression mechanisms

6. As dark metadata can exist in the header metadata area, this means that references of any kind can appear to be unresolved even though they are correct. MXF decoders need to be able to cope with this.

8.5 Using ULs and UUIDs

SMPTE ST 377-1 contains a large number of sets and properties (also referred to as items). The normative definition of the properties — what they are and their type (e.g., integer, UL, string, etc.) are identified by a UL that is defined in the SMPTE metadata dictionary (SMPTE RP 210).

8.5.1 UL properties and their scope

In the MXF format document, bytes 9 onwards of the UL entry in the dictionary are replicated in the "UL designator" column of set definitions. Use of local set coding provides efficiency by substituting a 16-byte UL value with a short 2-byte tag.

Some of the property values in SMPTE ST 377-1 are themselves ULs presented as self-identifying labels. These labels are generally used to identify lists of unique things. For example, the "picture coding type" property has a label value that directly identifies the picture coding type in use. All the picture coding types that are known to MXF are listed in the SMPTE labels registry. Applications that need to determine the meaning of a label must use the SMPTE labels register (SMPTE RP 224) as the normative reference.

At the time of writing, SMPTE RP 224 is available as a free download from the SMPTE Registration Authority website at: <http://kws.smpte-ra.org/mdd/index.html>

In certain cases, an encoder can place an un-registered UL or a non-UL unique identifier in a property of type “UL”. Example cases are where additional MXF features are being developed, but have not yet been standardized, and where private extensions are added for use in a carefully controlled MXF system. Some of these cases are outside the scope of the MXF format, but decoders need to make every effort to handle these files gracefully. For example, decoders must not rely on the values being validly coded as a registered SMPTE label.

8.5.2 UUID properties and their scope

Universally unique IDs (UUIDs) are algorithmically computed unique numbers that can be used in MXF files in two different ways. Firstly, they are used for making links between different parts of the same file, such as with strong and weak reference Instance UIDs. Secondly, they are used to provide identifying or typing information, such as where a property’s local tag is translated via the primer pack into a UUID. In the first case, the UUIDs have partition scope; an occurrence of the same UUID in two different files, or even two different partitions of the same file, does not imply any relationship between them — even thought the likelihood of the same UUID being generated twice is extremely remote. In the second case the UUIDs have global scope, so that wherever the same UUID is used it has the same meaning.

8.5.3 Byte order of UUIDs

IETF RFC 4122 states that in the absence of explicit specification to the contrary, UUIDs are encoded as a sequence of 16 bytes starting with the bytes holding the time field and ending with the node ID. However, the significance of byte order depends on the scope of the UUID.

If the scope of a UUID is local to the file then the byte order is unimportant, providing each occurrence of that UUID uses the same byte order. In these cases the default order specified in RFC 4122 must be used.

Where UUIDs have global scope the byte order is significant. In these cases the byte order will be given when the UUID value is published. For example, where a manufacturer publishes the UUID that a particular device inserts into the “product UID” field in the identification set, the byte order of that UUID will be specified as well as the values of the bytes.

8.5.4 Storing ULs and UUIDs in the same property

Some data fields, such as the UID property of the LocalTagEntry batch in the primer, can contain either a UL or a UUID. In this case there is an advantage to using a modified byte order for the UUIDs. All UUIDs have a 1 in the most significant bit of the "clk_seq_hi_res" word (byte 9), whereas all ULs have a 0 in the most significant bit of the first byte. If UUIDs are stored with a byte order that places the "clk_seq_hi_res" word first, then it is always possible to tell if the value is a UL or a UUID by examining the MSB of the first byte. This modified byte order also prevents the remote possibility of a UUID being stored that matches a registered UL. For these reasons, it is recommended that when any UUID is published for inclusion in a data field that can also contain ULs, the byte order specified for that UUID be the same as the RFC 4122 default order, but with the upper and lower eight bytes swapped.

The section above gives rise to the following guidelines:

> 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.
>
> According to the ST 377-1 specification, MXF decoders are expected accept a swapped UUID wherever a UL is expected and a swapped UL wherever a UUID is expected.

8.5.5 ULs identifying the file's handling requirements

In the partition packs of the MXF file, there are a number of properties whose UL values are intended to give an indication of the codec and handling requirements needed for the file. This information is intended to be a performance enhancement to provide "fail-fast" functionality. This information is located in the first few bytes of every file so that an application can quickly determine if it is able to handle the content of a file. The information is copied from the authoritative information in the header metadata.

The operational pattern UL identifies the timeline complexity of the file. The essence container ULs identify the essence data that is contained in the file so that an application can determine if a suitable codec is available. These numbers are registered values in SMPTE RP 224 so that an application that cannot handle a particular essence container type is able to report the essence type in the file. This type of reporting behavior helps users to identify content and is encouraged. Anonymous failure such as "a codec cannot be found" without reporting what sort of codec was sought is not encouraged. Older decoders that are unaware of recently added UL values need to at least attempt to report the ULs that were not known. It is important to note that it cannot be possible for this information to be provided by all MXF encoders and that decoders must not fail if this information is empty or missing.

If an MXF file contains multiple essence containers, but they are all of the same type, then the essence container label appears in the partition pack only once. This non-duplication is to ensure that a higher operational pattern file with 100 small MPEG clips need not insert 100 ULs in the list.

Some essence container specifications (such as the MPEG long GOP generic container mapping) define essence container ULs for the different MPEG streams that can be encountered when trans-wrapping from MPEG program stream to MXF. It is possible that the list of essence containers will contain a UL for the sound data and a UL for the picture data even when the resulting file contains only a single essence container with interleaved sound and pictures. During the design of MXF it was felt that there needed to be a descriptor for each of the different types of audio so that the MXF decoder requirements could be determined rapidly.

As an example, if you have an OP1a MXF file with MPEG-2 video, two channels of AES audio and time code, the file would have:

- 2 ULs in the EssenceContainer list (1 video, 1 audio);
- OP1a declared in the partition pack and the preface set;
- 4 tracks: 1 picture, 2 sound, 1 time code;
- Material package tracks have the same duration as the top-level file package tracks.

MXF decoders need to be able to cope with the case where there are many essence containers of the same type with a single UL in the EssenceContainer list. MXF decoders also need to be able to cope with the case where there are several ULs in the EssenceContainer list, each of which relates to a different element of a single MXF generic container.

8.5.6 What is the difference between the Essence Container UL and the Essence Coding UL?

There has been some industry confusion about the use of the Essence Container UL and the Essence Coding UL and this sub-section sets out to clarify this issue.

8.5.6.1 Essence Container UL:

The Essence Container UL is defined in both the SMPTE ST 379-1 and SMPTE 379-2 essence container specifications. It is also replicated in the Essence Container property of the File Descriptor defined in SMPTE ST 377-1, Annex F.2. This replication provides a fast lookup for decoders so that they do not have to parse the MXF generic container.

The Essence Container UL value identifies how essence is wrapped in the MXF generic container and also identifies the format of the essence that has been wrapped. The list of values is typically small because the essence is wrapped in frames or clips and the essence format itself is typically limited to a small set of frame rates and picture formats. The UL has also been used in some cases to indicate bit-rates in cases where the essence has been compressed to a defined bit-rate value or set of bit-rate values.

The UL values for the Essence Container UL are listed in SMPTE RP 224 under the Item Designator node value of 0D.01.03. (where “0D” is byte 9 of the UL).

8.5.6.2 Essence Coding UL:

In contrast, the Essence Coding UL is defined ***only*** in SMPTE ST 377-1 and it is a property of the Essence Descriptors (Picture, Sound and Data – see SMPTE 377-1, Annexes F.4.1 [Picture Essence], F.5 [Sound Essence] and F.6 [Data Essence]).

The Essence Coding UL value identifies how the essence itself is coded (i.e. not how the essence is wrapped in the MXF GC). This value is used to identify how the essence data (whether compressed or uncompressed) is represented in terms of sampling rates (4:4:4, 4:2:2, etc.), coding formats (RGB, YCbCr, etc.), parametric values (4:3, 16:9 for picture essence), compression formats where applicable (MPEG, DV etc.) and audio formats for multi-channel sound essence.

Essence Coding UL values are defined in SMPTE RP 224 under Item Designator node value of 04.01 for Picture Essence coding, 04.02 for Sound Essence Coding and 04.03 for Data Essence coding (where “04” is byte 9 of the UL).

8.5.7 Data definitions

There are several MXF sets that are generic (e.g., the sequence set) and the specific behavior is identified by a data definition property that has a 16-byte "magic number" value as a label value that a decoder can use to figure out how to handle the component. These label values are registered in SMPTE RP 224.

8.6 Coding Objects as Sets

KLV coding allows related metadata items to be grouped together in sets; e.g. titling metadata might be grouped into a set for convenience. SMPTE ST 336 defines several mechanisms for grouping the data together. Basically, a set comprises an outer KLV that defines the set and a number of inner KLVs that define the data items.

The inner keys could be full length (Universal set) or could be shortened for processing and storage convenience. KLV sets using these shortened item keys are known as local sets and the technique is fully defined in SMPTE ST 336. This standard defines how all sets have Universal labels with a consistent definition in the first 8 bytes of the type of data set or data pack being used. The options provided are:

- Universal set;
- Local set;
- Variable length pack;
- Fixed length pack;
- Global sets (not used in MXF)

All MXF decoders must support local sets. Encoders need to use the sets as required by the operational pattern. If there is no guidance in the operational pattern, then the encoder needs to opt for a local set using the local tags as defined in SMPTE ST 377-1. Note that 2-byte lengths in local sets are always coded as big-endian (i.e., MSB first).

Every property in MXF has a full 16-byte Universal label so that the property can be interchanged with other systems as either a single KLV item or as a Universal set.

SMPTE ST 377-1 uses KLV local sets with 2 byte tags and 2 or 4 byte lengths and includes a special pack structure called the “primer pack” to ensure that dark metadata properties can be created and handled without the possibility of a numerical clash of local tag values. Why is this important? Imagine that two encoders, X and Y, each independently, want to extend the MXF identification set to include some vital property of their application in every MXF file that they save. Without the primer pack, there is a finite chance that they will both choose the same local tag value for their private metadata property and when they open each other’s files, they will misinterpret or potentially even corrupt each others’ metadata properties. The primer pack mechanism exists to prevent this happening.

8.7 Using Text

Many of the text fields in MXF are encoded using UNICODE. The coding technique is UTF-16 with big-endian byte order to allow good international support. More information on UNICODE can be found in the reference listed in annex B. There are occasions when ISO-646 text is used. This is often to comply with some other standard such as the ISO-639 language descriptor codes.

Text is stored in a KLV or tag-length-value structure. Zero word termination of strings is optional. A string can be the same length as the “L” of the KLV or the “length” of the tag-length-value with no zero word at the end. Alternatively, a shorter string can be placed in the space allocated by the KLV or the tag-length-value structure by inserting a zero word after the last character of the string. MXF decoders need to support both mechanisms.

8.8 Tracking Changes with Generation Numbers

A generation UID number is a weak reference to the identification set that was created when the MXF file was saved or modified by an application. Each time a MXF file is modified, a new identification set is created. If a metadata set is changed, the generation UID property is updated so its value will be the same as the generation UID of the Identification set that was created when the property was modified.

Some generation number properties are optional and decoders must not rely on their existence. However, in certain applications, they can be very useful in that they allow your application to check if another application has modified the data in the properties of a group. For this reason, some applications require the generation UID property value to be correctly updated every time a file is modified in any way.

The generation UID property value also allows your application to track whether another application has modified data in an MXF file that can invalidate data that your application has stored in extensions. The generation UID property is a weak reference to the identification object created when an MXF file is created or modified. If your application creates extended data that is dependent on data stored in MXF built-in classes or properties, you can use the generation UID property to check if another application has modified the MXF file since the time that your application set the extended data. To do this, your application stores the value of the generation UID of the identification object created when your application sets the value of the extended data.

9 MXF in Detail

9.1 General Overview

SMPTE ST 377-1 defines a file format for the transfer of program material between equipment in the professional broadcast and production environments. Stream and file transfers are both used for the interchange of program material, with file transfers increasing in proportion to stream transfers. Neither will dominate; rather they will co-exist and the MXF file is designed to work within both transfer classes.

Files are often created directly from incoming streams and are often converted into streams for emission and distribution. The MXF standard specifies an MXF file format that is readily convertible to and from common streaming formats with low overhead and without loss of data. MXF files can also be stored.

In order to appreciate the differences between stream and file transfers, the major characteristics of each can be summarized as follows:

File transfers...

1. Can be made using removable file media;
2. Use a packet-based reliable network interconnect that is usually acknowledged;
3. Are usually transferred as a single unit (or as a known set of segments) with a predetermined start and end;
4. Are not normally synchronized to an external clock (during the transfer);
5. Are often point-to-point or point-to-multipoint with limited multipoint size;
6. File formats are often structured to allow access to essence data at random or widely distributed byte positions.

Stream transfers...

1. Use a data streaming interconnect and are usually unacknowledged;
2. Are open-ended, with no predetermined start or end;
3. Are normally synchronized to a clock or are asynchronous, with a specified minimum/maximum transfer rate;
4. Are often point-to-multipoint or broadcast;
5. Streaming formats are usually structured to allow access to essence data at sequential byte positions. Streaming decoders are always sequential.

Figure 11 illustrates the interoperation between streaming transfers based on stream interfaces such as SDTI and file transfers between disc servers and tapes. One of the features of file transfers is that servers can support playout before file closure (i.e., read from a partially written file while it is still in the process of writing), so blurring some of the distinctions outlined above.

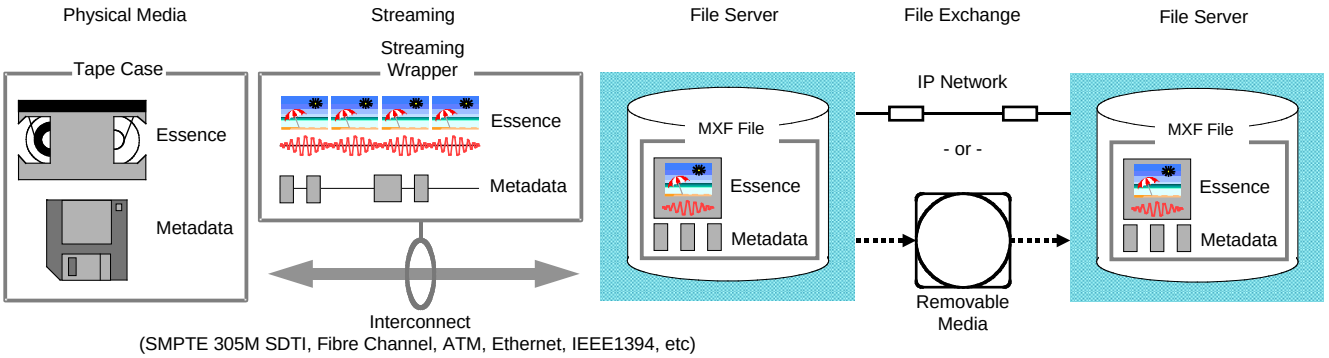

Figure 11 – MXF files and streaming formats

9.2 Content Package Model

The content package model used in SMPTE ST 377-1 is based on that defined by the EBU/SMPTE Task Force Report and illustrated in Figure 12.

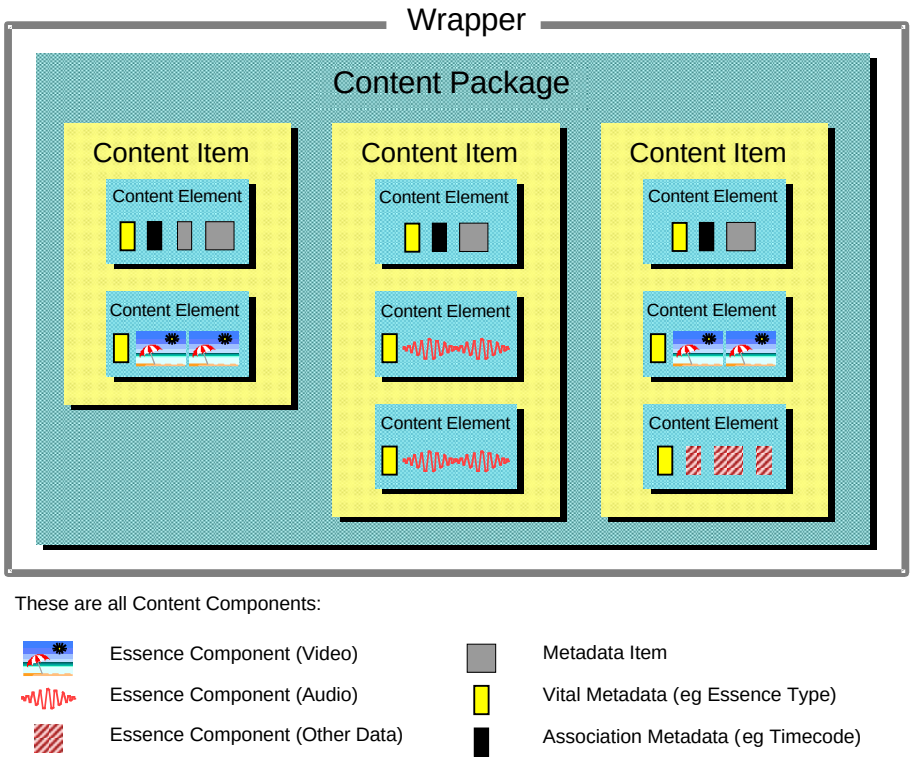

Figure 12 – Content package model

The content model uses the terminology of SMPTE ST 336 (KLV coding) and SMPTE ST 298 (Universal labels), which define:

- Universal labels (ULs) used as keys;
- Key-length-value (KLV) formatting of individual metadata and essence items;
- Coding of groups of data items into sets and packs.

The content model also uses the terminology employed by SMPTE ST 326 – SDTI-CP, which defines frame-interleaved content based on the following components:

System item: that includes system level descriptive metadata and content metadata;

Picture item: that includes one or more picture elements;

Sound item: that contains one or more audio elements;

Data item: that contains one or more data essence elements.

Compound item: that contains one or more intrinsically interleaved elements (such as an interleave of DV-DIF packets)

Link item: that links metadata in the system item to any one of the elements.

Each of these essence elements can be separately indexed by an index table and can also be mapped to a track in a file. The track file is the metadata object that controls the way in which this essence element is used.

9.3 MXF Essence Containers

MXF files use essence containers based on the content package model to encapsulate one or more essence elements. These essence elements can be intrinsically interleaved (for example, a SMPTE ST 314 DV-based stream) or can consist of a single non-interleaved essence element.

In order to support stream capability for two or more essence elements, the essence elements are interleaved over a limited duration (typically 1 frame). Each essence element can be encapsulated using KLV coding over the interleave duration to allow an MXF decoder to access the essence on these KLV boundaries.

The MXF Format does not provide the individual essence container specifications, but defines the constraints that a compliant essence container specification needs to meet in order for it to be encapsulated in an MXF file body. Constraints on the essence container are defined in the operational pattern document and the essence container document. They can be summarized as follows:

1. Must encapsulate each essence component with KLV coding using publicly registered keys;
2. Must provide for interleaving of the essence components over a limited duration (typically one frame), when inputs or outputs are used for streaming;
3. Must be standardized as an open specification, preferably through the due-process of SMPTE;
4. Must meet the SMPTE criteria for a standard.

It is expected that such essence containers will become available for the systems below.

Note that none of the compression formats is a compulsory function.

9.3.1 SMPTE ST 377-1 Requirements for Essence Container Specifications

SMPTE ST 377-1-2011 lists a number of requirements (i.e. statements using the word “shall”) for essence container specifications. These are distributed throughout the document and are copied in Table 2 for information only:

Table 2

Requirement
“For an Essence Element, there shall be an associated MXF Essence Container specification that defines how the Essence Element shall be KLV encoded in the Essence Container.”
“An Essence Container specification shall identify the Essence Descriptors that are required to describe the Essence Element.”
“The KLV coding rules for each Essence Container shall be defined in individual Essence Container specifications.”
“The byte order of Essence data as a value in a KLV packet shall be defined by the Essence Container specification.”
“Essence Container specifications shall meet all the normative criteria listed in Sections 10.2 and 10.3”
“An Essence Container shall be coded as a concatenated sequence of individually coded KLV packets where each KLV packet meets the following requirements”
– “Each KLV packet shall be coded according to SMPTE ST 336 and the values of the KLV packet Key shall be publicly registered with the SMPTE.”
– “The Essence Container shall be formed by a stream of one or more KLV packets.”

– "The Length field of each KLV packet shall be coded according to SMPTE ST 336 with a limit to the coding range provided to limit the requirements of compliant equipment for this Essence Container."
– "The Essence Container specification shall define the byte order."
– "The Essence Containers used in streaming Operational Patterns shall be capable of interleave over a defined interleaving period or shall be capable of being multiplexed in an MXF file using the Partition mechanism."
– "The Essence Container specification shall be assigned a registered SMPTE Universal Label"
– "The Essence Container specification shall define which Descriptor Set(s) shall be used to describe the contents of the Essence Container payload."
– "The Essence Container specification shall define the mechanism to create unique Track Numbers for identifying specific content within the Essence Container (SMPTE ST 379-1 and -2 provide such a specification for Generic Containers.)"
"Essence Container specifications shall identify if the Edit Unit of stored Essence and its Essence Elements are of constant size"
"Essence Container specifications shall identify the Frame Layout values"

All the quotes in the above table are replications of normative text in SMPTE ST 377-1 as a guide to users. They are categorically not to be taken verbatim from this guideline.

9.3.2 SMPTE ST 377-1:2011 Recommendations for Essence Container Specifications

SMPTE ST 377-1:2011 also makes a number of recommendations (i.e. statements using the words "should" and "may") for essence container specifications that are summarized in Table 3:

Table 3

Recommendation or Permission
"An Essence Container specification may define how the Index Table specification shall be applied for indexing the Essence Element."
"Other MXF specifications (e.g. Essence Container specifications) may define additional constraints for KLV Lengths."
"Essence Container specifications may require a non-zero (KAG) value for all Partitions"
"Individual Essence Container specifications may define which KLV elements shall be aligned to a gridline"
"Essence Container specifications may further constrain the rules of MXF encoding"
"Essence Container specifications may define additional provisions about the number of Tracks for specific Essence Containers."
"An Essence Container specification may also define how Index Tables shall be used and how the Essence Descriptor shall be coded."

“An Essence Container specification may be split over multiple documents when generic structures and specific mappings of different Essence types are required.”
“The interleave or multiplex duration … should be the period of the minimum duration of usable picture Essence, typically a picture frame period.”
“The KLV packets of each interleave period should contain Essence of essentially the same timing (for example, audio-video timing is rarely sample-accurate).”
“An MXF Essence Container specification should be a public standard from an internationally accredited standards body.”
“The Essence Container specification may define the use of Descriptor(s) for that Essence Container type.”
“The Essence Container may contain KLV packets containing separate streams of metadata in addition to Essence data.”
“The Essence Container specification should define the use of Index Tables.”
“The Essence Container specification may define any mappings required from Multi-track audio in the Header Metadata to Multi-channel audio in the Essence Container.”
“The Essence Container specification may define a KAG value (see Section 6.4.1).”
“New Essence Container specifications are recommended [i.e. *should*] to use the MXF Generic Container defined in SMPTE ST 379-1 or SMPTE ST 379-2 where applicable.”
“Essence Container specifications may define additional constraints that apply to this situation.”
“Essence Container specifications may define implementation details of Index Tables.”
“Essence Container specifications may define this term (Sample Rate) more strictly for individual Essence types.”

All the quotes in the above table are replications of normative text in SMPTE ST 377-1 as a guide to users. They are categorically not to be taken verbatim from this guideline.

Caveat - Users must not assume that the quotes in the above two tables are a complete and full summary of all essence container requirements. The tables above are provided for guidance only and users must always refer to SMPTE ST 377-1 as the defining document.

Note that individual essence container mapping standards can add further requirements specific to the mapping of the target essence kind.

9.3.3 MXF generic container

SMPTE 379M-2004 provided a generic container with intrinsic interleaving that allowed most existing formats to be mapped into the MXF Format with the minimal invention of new techniques.

Wrapping all essence variants in a common essence container format is advantageous for system design and interoperability. The MXF document suite specifies mappings of a variety of essence formats into the MXF generic container as described below.

The MXF generic container can also use essence elements and metadata items defined in SMPTE ST 331 through application of the specifications in SMPTE ST 385 (mapping SDTI-CP essence and metadata into the MXF GC).

A number of issues were found in realizations of the original MXF generic container (SMPTE 379M-2004) such as provisions that were unconstrained leading to excessive decoder requirements and provisions that could result in files that could not, in practice, be decoded. These issues could not be resolved in a revision of SMPTE 377M-2004 because this could have invalidated existing files. The solution accepted was to create a variant of SMPTE 379 that addressed these issues.

Thus the 5-year review of SMPTE 379M-2004 resulted in two variants;

- SMPTE ST 379-1 which is essentially an update of SMPTE 379M-2004 with no changes to the normative provisions.
- SMPTE ST 379-2 which is a re-write of SMPTE 379M-2004 that provides additional constraints to improve interoperability.

Users are encouraged to consider using SMPTE ST 379-2 as the MXF essence container. SMPTE ST 379-1 is provided essentially for backwards compatibility with existing files created using the original SMPTE 379M-2004.

The additional constraints provided by SMPTE ST 379-2 are not obvious without close reading. To aid the reader, the table below highlights the changes made in SMPTE ST 379-2 compared to SMPTE ST 379-1.

Table 4

Essence Element is a new term defined in SMPTE ST 379-2 but one that is interchangeable with any single essence kind (picture, sound etc.).
Section 6 (Legacy Considerations) provides new information not present in SMPTE ST 379-1.
Section 7.1 of SMPTE ST 379-2 permits KLV fill items. This facility was not present in SMPTE ST 379-1.
SMPTE ST 379-1 did not specify that the order of CI's in each CP of the GC must be consistent for all CPs. SMPTE-379-2 does require this consistency.
SMPTE ST 379-2, Section 7.4.2 (clip wrapping) requires 2 or more edit-units in a clip-wrapped GC. In SMPTE ST 379-1, single frames could be either frame or clip wrapped.
SMPTE ST 379-2 requires clip wrapping to be used only for mono-essence data. SMPTE ST 379-1 did not provide this constraint making it possible for interleaved essence data streams to be clip-wrapped.
SMPTE ST 379-2 provides for a custom wrapping facility not present in SMPTE ST 379-1.
SMPTE ST 379-2 provides a new section on indexing essence with the three different wrapping kinds. This was not provided in SMPTE ST 379-1.
SMPTE ST 379-2 lists encoder rules using the word "shall" where SMPTE ST 379-1 used "should". It also provides specific information on the rules for using long-GOP coded essence kinds.
SMPTE ST 379-2 provides specific information on the rules for using long-GOP coded essence kinds not provided for in SMPTE ST 379-1.
SMPTE ST 379-2 provides specific information in regard to the handling of missing essence elements from a given track in a CP that was not present in SMPTE ST 379-1.
SMPTE ST 379-2 Section 7.5.2 is new and provides information on how to add essence stream data after a recording has started. This was not present in SMPTE ST 379-1.

SMPTE ST 379-2 Section 7.6.1 provides an additional sentence that permits an essence container spec to further constrain the length field of the KLV coding syntax. This was not present in SMPTE ST 379-1.
SMPTE ST 379-2 adds a new paragraph in Section 9.2.2 that clarifies the linking mechanism. This was not present in SMPTE ST 379-1.
SMPTE ST 379-2 adds a new bullet point in Section 9.2.3 that separately defines the linking mechanisms from a system element to another system element and a system element to an essence element. This was not present in SMPTE ST 379-1.
SMPTE ST 379-2 Section 8.3 provides an expansion of the definition of the content package structure including the requirement that the structure of each content package within a given essence container is consistent. This was not present in SMPTE ST 379-1.
SMPTE ST 379-2 Section 10, paragraph 1 added text stipulating that a valid CE payload requires a normative specification. This was not present in SMPTE ST 379-1.
SMPTE ST 379-2 Table 3 defines the version byte (8) to be a fixed value of 1. This was left undefined in SMPTE ST 379-1.

Caveat - Users must not assume that the items in the above table provide a complete and full summary of all the differences between SMPTE ST 379-1 and SMPTE ST 379-2. The table above is provided for guidance only and users must always refer to SMPTE ST 379-1 and ST 379-2 as the defining documents.

9.4 Mapping Essence and Metadata into the MXF Generic Container

This sub-section provides an overview of the mapping of specific classes of essence and metadata into the MXF Generic Container. This is not a complete list and users need to be aware that ongoing work means more essence mappings will become available in the future.

Note that the type of essence or metadata encapsulated in each specific variant of an MXF file is defined by an individual essence container specification and is identified in the file header by one or more unique essence container labels.

9.4.1 MPEG-2 long GOP and type D-10

MPEG compressed picture essence in streams can be interleaved in several different patterns as defined by the ISO 13818-1 systems layer, including elementary streams, program streams, and transport streams. The mapping of MPEG picture essence into MXF is defined in SMPTE ST 381. As of 2010, this document was divided into multiple parts with part 1 (SMPTE ST 381-1) defining the mapping of MPEG essence into SMPTE ST 379-1 (the unconstrained MXF GC) and part 2 (SMPTE ST 381-2) defining the mapping of MPEG essence into SMPTE ST 379-2 (the refined MXF GC)

SMPTE type D-10 MPEG elementary streams are defined by SMPTE ST 356. MXF generic container specifications allowing wrapping of this constrained essence variant defines frame by frame wrapping of the elementary stream in the generic container as the essence encapsulation method.

9.4.2 DV compressed essence

SMPTE ST 383 defines the means for mapping systems employing the DV family of compression schemes defined by IEC61834-2, SMPTE ST 314 and SMPTE ST 370 in the MXF generic container.

9.4.3 Uncompressed pictures

MXF files can be used for the transfer of program material employing uncompressed video at all resolutions, including standard and high definitions. SMPTE ST 384 specifies the use of the MXF generic container for encapsulating uncompressed video and the use of a separate payload ID (as defined by SMPTE ST 352) to carry signal parameters for use by decoders and trans-coders. Like all

generic container essence element mappings, this picture element can be used on its own or can be used with appropriate sound or data essence elements.

9.4.4 Audio

An MXF mapping document for the encapsulation of AES3 audio and broadcast wave compatible audio in the generic container has been defined in SMPTE ST 382. This audio element can be used on its own or to add audio to another generic container element such as a picture element (for example, uncompressed pictures or MPEG long GOP pictures).

9.4.5 JPEG 2000 pictures

The user requirements for digital cinema (D-Cinema) required the pictures to be individually coded using MXF files and to be compression coded using JPEG-2000 compression. The MXF mapping document that encapsulates D-Cinema pictures is specified in SMPTE ST 422.

9.4.6 Other essence types

MXF files can encapsulate various other video essence compression systems. See Annex A Bibliography for a list of all mappings available at the time of publication.

9.4.7 Metadata and other data types

MXF files can encapsulate metadata and other data streams, including SMPTE ST 436 that defines how Ancillary and Vertical Blanking Interval (VBI) data packets are mapped into the MXF generic container.

Metadata from the SDTI-CP interface (SMPTE ST 326 and SMPTE ST 331) can be carried in MXF files using the mapping specification provided by SMPTE ST 385.

There is also a standard that defines how to encapsulate stream metadata into the MXF GC in SMPTE ST 394. This has a companion document (SMPTE ST 405) that defines the stream metadata elements for use with SMPTE ST 394 and provides backwards compatibility for the metadata elements defined in SMPTE ST 331 (this provides a more future-proofed approach compared to that defined in SMPTE ST 385).

9.5 How MXF Header Metadata Relates to the Essence Container

SMPTE ST 377-1 provides a physical representation of an underlying class model for structural metadata including tools for data identification and data relationships. The method of relating the structural header metadata to the contents of the essence container is logical, but can be considered complex. This sub-section provides guidance on how this method operates.

In each partition of an MXF file, there can be any, or all, of the following core components:

1. A partition pack that defines:
   – body SID for the container data stream in this partition;
   – an Index SID for the index table in this partition.
2. A primer pack.
3. Header metadata repetition that includes:
   – a content storage set at the top level;
   – one or more top-level file packages each associated with an essence container data set;
   – other metadata to describe the entire file (after all, it is a header metadata repetition).
4. An essence container (that occupies the whole file body or a part).
5. Unique IDs that link data sets together (16-byte Instance UIDs).
6. Unique material IDs (32-byte UMIDs) that identify the essence container.

These components are related as illustrated in Figure 13.

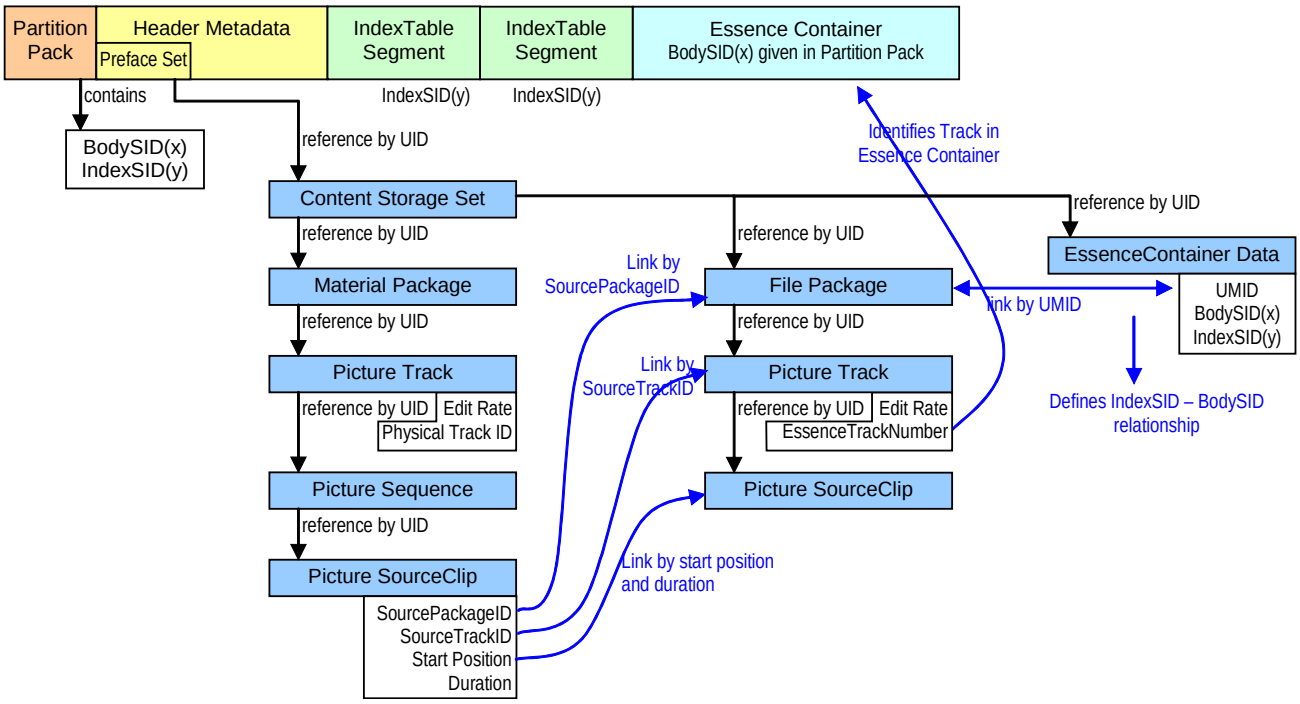

Figure 13 – MXF metadata and relationship to the essence container

The relationships are as follows:

– The partition pack includes a BodySID and an IndexSID that identify the essence container segment and index table segments in the partition. These are linked to the BodySID and IndexSID in the relevant top-level file package via the corresponding EssenceContainerData set. They are also linked to the BodySID and IndexSID in the relevant index table. When the BodySID value in a partition is zero, it indicates that there is no essence container segment in this partition. Likewise, a zero IndexSID value indicates there are no index table segments in this partition.

– The header metadata has a content storage set at the top level that contains a set of package UIDs and a set of EssenceContainerData UIDs. The content storage set strongly references every package, including each top-level file package as well as each material package. The content storage set will also reference lower-level source packages where these are present in the header metadata.

– Within the header metadata, there is also an essence container data set for every top-level file package. This set provides the linking between BodySID, IndexSID and their related package UMID value. This mechanism relates the partitions and index tables within the file body to the top-level file packages in the header metadata.

Note that the package UIDs are basic UMIDs as defined by SMPTE ST 330.

9.5.1 External files — where is the essence?

The MXF essence descriptor contains a list of properties called “locators”. MXF supports two different types of locator — network and text. The top-level file package that describes the essence (i.e., the one that is referenced by the material package) can have external essence, and the decoder needs to scan the locators *in the order they are given* to find the essence. A typical example of this might be the creation of a CD-ROM where the network locators are given as a file reference relative to the location of the MXF file, followed by other locations in which the file might be found; e.g.:

Network locator: “src/clip1.dv” — a relative file reference to clip1.dv in folder **src**;

Network locator: file://usr/~jon/clip1.dv — an absolute file reference to clip1.dv in **jon**'s home folder;

Text locator: "clip1 DV tape is on shelf 42" — a text locator intended for a human to interpret.

Even though the actual essence data is external to the file, there can be metadata describing the essence within the file. In the extreme case, all the essence could be external to the file leaving a small MXF stub that fully describes the external essence. MXF Files with internal essence can also have locators. When all the essence can be found internally, the locators need to be treated as being for information purposes only. In higher operational patterns, it is possible that some of the essence will be internal and some of it will be external. In this case, internal essence, where present, must take precedence over external references. Where there is no internal essence available from a material package SourceClip reference, the locators must be searched in their listed order to find the content. External content can be verified by checking the BodySID value in the essence container set for the appropriate UMID. A zero value indicates external essence.

9.5.1.1 Indexing External Essence

External essence can be located in an MXF file with its own index table, or it can be located in a MXF file without an index table, or in a non-MXF file. In all cases the referencing MXF file can include an index table for the external essence. Unlike an index table for internal essence, which is required to be valid, there is no guarantee that the external file has not been modified in such a way that the index table is no longer correct. This means that applications making use of an index table to access external essence need to take great care. If the external file contains some form of index table this will be more reliable, but it might not be as convenient.

9.6 Operational Patterns

Different applications produce and consume material of various degrees of complexity and structure, from a single clip to a multitude of clips and effects. Applications requiring only the simplest files need not be burdened with support of the most complex. To maximize interoperability MXF uses operational patterns (OPs) to define constrained levels of file complexity.

During the development of MXF there were many different attempts at defining the functionality of an operational pattern. The goal was to create a number of axes that allowed software and hardware developers to create products with different levels of functionality (and hence cost). These different axes had to correspond to real world ways of working, and had to provide mechanisms for a file to be "flattened" from a complex operational pattern to a simple operational pattern in a way that made sense to someone working with the multimedia content.

The description below is of the different axes followed by a non-exhaustive discussion of some applications.

9.6.1 Operational pattern "axes"

When trying to constrain the complexity of an MXF file, there are different axes or degrees of freedom that can be constrained independently. Most operational patterns have been written as a constraint on the axes in this section. However, for certain specialized applications (such as allowing audio-only WAV files to be read by non-MXF devices) there can be specialized operational patterns that constrain the specification differently (such as SMPTE ST 390, OP Atom). Regardless of the operational pattern, any MXF decoder is able to read the header and report the contents of the file and why it can or cannot process the file.

The operational pattern axes are arranged so that any operational pattern to the left, or above another operational pattern is a subset of its functionality as illustrated in figure 14. For example, operational pattern 3b is a superset of the functionality of OP1a, OP2a, OP1b, OP2b and OP3a, and includes not just the ability for each material package to access sequential top-level file packages, but also the ability to access a sequence of ganged top-level file packages.

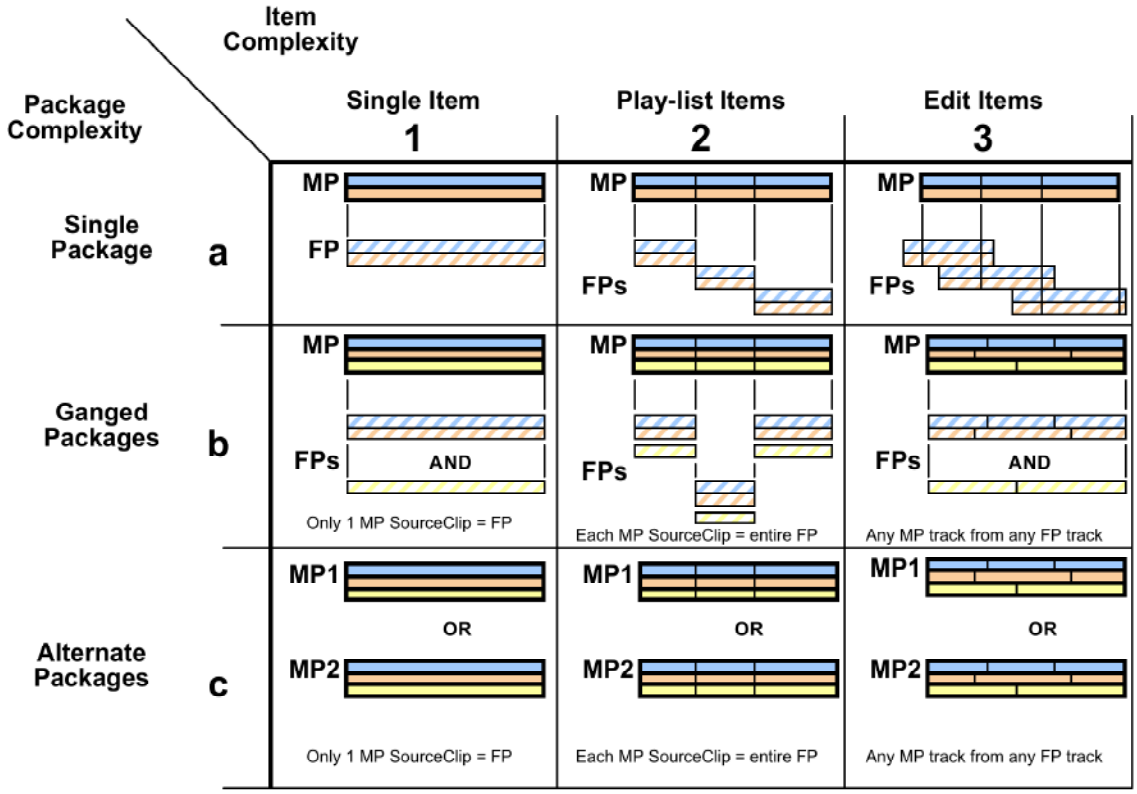

Figure 14 – Operational pattern axes

9.6.1.1 Item complexity

In principle, there are three levels of constraint to the temporal relationship between different top-level file packages within the MXF file:

Single item: The file contains top-level file packages that have the same duration as the output time-line (like a tape).

Playlist items: The file contains top-level file packages that are butted one against the other. All tracks are switched synchronously with optional audio fade out / fade in to prevent clicking. This can be likened to a playlist of tapes.

Edit items: The file contains several top-level file packages with one or more cut edits. Tracks can have independent editing to allow audio and video to be switched at different points in the timeline. This will often involve random access within the file and therefore MXF files in this column are unlikely to be stream-able.

9.6.1.2 Package complexity

Single package: The file contains only one active essence container at any point on the output timeline.

Ganged packages: The file contains two or more essence containers that share a common synchronized timeline. The MXF structure is used to wrap several essence containers and multiplex them using the KLV and partitioning rules. This could be used to gang together an MPEG picture track in one package with an uncompressed sound track in another (possibly external) package.

Alternate packages: The file contains several versions of the “program”. There are several material packages that might be used to control a browse track or different language versions of a program, or different edits of some finished material destined for

different censorship zones. For example, an OP1c file can have two continuous timelines — one for the French soundtrack and another for the English soundtrack. Another example is an OP3c file, where not only is there a choice of English or French, but the cut lists for the output tracks are different. Since this OP is a superset of the ganged package complexity, it also has the capabilities of ganged packages as well as alternate packages.

9.6.2 Operational pattern qualifiers

In addition to the axes above, there are operational pattern qualifiers that modify the behaviors above.

9.6.2.1 Internal / external flag

This is a simple flag that modifies an operational pattern. It has two states to indicate either that all the essence containers are internal to the file (internal) or that one or more of the essence containers are in an external file (eXternal). For example, an OP1b file can have internal picture data, but external sound data.

9.6.2.2 Stream / non-stream (wire / storage) flag

This is a simple flag that indicates either that the partitions in the file have been arranged so that it can be streamed on a wire (Wire file), or that some other non-streaming arrangement has been used (stored file). The streamed file representation implies that essence containers are multiplexed together and that within an essence container, any interleave that exists will allow decoding of the essence during streaming file transfer so that the pictures can be viewed and the sound heard during transfer with minimal latency. The size of buffers required to do this is an application issue and outside the scope of SMPTE ST 377-1. Any file that does not have this streaming property is just a file.

9.6.2.3 Uni-track / multi-track flag

This is a simple flag that indicates that all the essence containers in an MXF file have only a single essence track. This flag is to aid workflows where all the different essence components of a production need to be individual files. This flag helps MXF decoders know that the file meets this criterion. The flag is either uni-track or multi-track.

9.6.3 Specialized Operational Patterns

Specialized operational patterns lie outside the realm of the standard operational patterns constrained by item and package complexity. At the time of writing, the only example of a specialized operational pattern is OP Atom which is described next.

9.6.3.1 Operational Pattern “Atom”

Specialized Operational Pattern OP-Atom is a simplified operational pattern for the storage and interchange of a single item of essence. Each OP-Atom MXF file contains a single essence stream in a single partition. It also contains a complete index table for the contained essence and the file footer always contains a copy of this index table. The header is always closed and complete. Some systems, such as SMPTE ST 428 D-Cinema Distribution Master, use a number of OP-Atom MXF files to store the essence.

9.6.4 Operational pattern applications

MXF applications need, where appropriate, to be able to perform the following functions with respect to operational patterns:

- Encoders and decoders must be able to report the most complex operational pattern they can handle.
- A decoder must be able to indicate what level of operational pattern has been processed when its capabilities have been exceeded.

- Encoders must always correctly signal the operational pattern of the files they create. This means that an MXF encoder capable of creating all possible operational patterns must not signal the files it creates with the highest operational pattern code. It must signal the operational pattern to which the file complies.

An application might give a file a name depending on its functionality, for example:

Test_OP1aiwm.mxf – mxf file with internal essence, wire-file, multitrack, Operational Pattern 1a;

Test_OP3cxm.mxf – mxf file with external essence, not stream-able, multitrack, Operational Pattern 3c.

Listed below are several MXF applications. They are intended solely to give a guide on how MXF operational patterns might be used. They are not normative definitions of the operational patterns concerned.

9.6.4.1 Video tape replacement

A video tape is essentially a single container with a single item on it. Even though there can be more than one “scene” or “shot” or “clip” on the tape, no special processing is required to play the sequence. All the material is internal to the tape and it is stored in a way that can be streamed. This makes an operational pattern for video tape replacement one of the simplest operational patterns.

9.6.4.2 Archive

There are many different archive applications. Often, it is desirable to have metadata or a browse track “online” and the full-quality content in some deep store. This requires referencing of external essence as well as multiple representations of the same content. There can only be one single item in each of the representations (each having the same duration) and the content could be arranged for streaming or storage depending on the precise application.

9.6.4.3 D-Cinema

D-cinema files use MXF and the standards used are listed in Annex A Bibliography.

9.6.4.4 Adding handles to material

Handles are extra bits of material that are present before and after the desired content. There are several ways in which these could be used in MXF depending on the desired result.

The most common use of handles is to adjust edit points, and / or to provide context for production processes such as color correction. This use of handles implies that the content within the handle is not actually used in the material package, but exists within the top-level file package. The resulting file would be in the edit items column of the operational pattern axes matrix. The precise row or column of the operational pattern would depend on the construction of the essence within the file. For a mono-essence file, it would be constructed as an OP2a or OP3a file. Multi-track files would be either OP2b or OP3b depending on whether or not the cut points of the top-level file packages are synchronized on the timeline.

9.7 MXF Decoder Design

MXF decoder design is, of course, an application-specific issue. This section is intended to advise the reader on issues that could improve interoperability with other systems.

1. MXF decoders need to be able to skip over any run-in.

2. MXF decoders need to be able to **parse** (i.e., understand the syntactic structure) at least the following:

   a. The KLV packet structure of all parts of the file (including the KLV packets of any kind of essence container);

   b. The KLV structure of the header partition, any body partition and the footer partition;

c. The KLV structure of any optional index tables;

d. The optional random index pack;

e. The basic header metadata structure in any partition;

f. The SMPTE Universal labels in all the partition packs;

In addition, it is desirable that MXF decoders **decode** (i.e., interpret and act on the values within) at least:

a. The metadata sets and individual metadata items defined in the minimum use of the simplest operational pattern.

3. Decoding of other aspects, such as the compressed bit-stream or the specific essence container in the file body, depends on the ability of the decoder to support those aspects. It is desirable that MXF decoders be able to locate and present the information that identifies the contents of the MXF file as follows:

a. The MXF file identification itself (that identifies that the file is MXF compliant) through the key value of the header partition pack,

b The UL of the operational pattern (structural metadata) to which the file conforms,

c. The array of ULs that identify each essence container and its contents in the file body, and

d. The array of ULs that identify each descriptive metadata collection within the file.

None of the above is a requirement for an MXF decoder.

Annex A Bibliography — Further Reading

The following list of informative documents provides background information related to the MXF specification.

The SMPTE Data Coding Protocol and Dictionaries, Jim Wilkinson, SMPTE Journal, July 2000 Vol. 109, No 7, Engineering Report

UML information for understanding class diagrams and other aspects of data modeling and programming http://www.oreilly.com

The MXF Book – Introduction to the Material eXchange Format, Nick Wells et al, 2006, Elsevier Inc., ISBN 10: 0-240-80693-x, ISBN 13: 9780240806938

File Interchange Handbook, Brad Gilmer et al, Focal Press, ISBN 0-240-80605-0