SMPTE STANDARD

for Television — Material Exchange Format (MXF) — File Format Specification (Standard)

Table of contents

1 Scope

This standard defines the data structure of the material exchange format (MXF) for the interchange of audio-visual material. It defines the data structure for network transport and may be used on storage media. This document does not define internal storage formats for MXF compliant devices.

The standard defines all the components of the MXF file specification including all those in the file header, file body and file footer. It defines the application of partitions in the file that provide valuable features such as the ability for an MXF file to serve many application requirements and recovery of partially received files. The document also defines key features of the file structure including the partition packs, the structural metadata, the primer pack, the random index pack and index tables.

The standard does not define either the essence container or the descriptive metadata. Instead, it defines the requirements for these components to be added as a plug-in to an MXF file.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision,

Approved
September 22, 2004

and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent edition of the standards indicated below.

ANSI/SMPTE 298M-1997, Television — Universal Labels for Unique Identification of Digital Data

SMPTE 336M–2001, Television — Data Encoding Protocol Using KLV

SMPTE RP 210, Metadata Dictionary Registry of Metadata Element Descriptions

SMPTE RP 224, SMPTE Labels Registry

IEEE 754-1985, Floating Point Format

ITU-R BS.1196 (1995) (Annex 2), Audio Coding for Digital Terrestrial Television Broadcasting

ISO/IEC 11578-1:1998, Information Technology — Open Systems Interconnection — Remote Procedure Call (RPC) Annex A, Universally Unique Identifier

3 Definition of acronyms, terms and data types

Acronyms, terms and data types used in the MXF specifications are defined in this section. Many of these terms are used in other MXF documents.

3.1 Acronyms and terms

AAF: Advanced authoring format.

Aggregation: Creating a new object by accumulating the items and semantics of all of its parts. For example a "car" object is an aggregation of the "engine" object, the "chassis" object and all the other objects that make up the car.

ASN.1: ISO/IEC 8825-1 standard notation format of which BER is a part.

Audio-visual: A term used to describe any playable content comprising picture (visual), sound (audio) or data essence and includes any metadata that the content may embed. The content may comprise a single essence element or many essence elements (of the same kind or different kinds) that are interleaved at an appropriate rate.

BER: Basic encoding rules — ISO/IEC 882-1 ASN.1 for encoding KLV lengths and labels.

Best effort: Best effort metadata is a category of metadata which may not be known at the time of writing a file. Best efforts should be made to complete these values during file creation. Only metadata parameters marked specifically as best effort metadata may be treated this way. If the value of the parameter cannot be completed correctly then its distinguished value shall be used.

Big endian: A byte order where bytes of a word are transmitted most significant byte first.

BodySID: Essence container identifier. Type and use given in the specification.

CBE: Constant bytes per element.

CBR: Constant bit rate.

CDCI: Color-difference component image.

Dark: Used to describe essence and metadata items that are unknown to an application at a given time. For example a camera GPS position may be known to a camera, dark to an MXF store and forward device, needed and processed by a librarian program for archive, but discarded by the same application for broadcast.

Dictionary: A list of values with defined meanings. (e.g., SMPTE RP 210 metadata dictionary).

Distinguished value: The (invalid) value of a best effort metadata item which is used to indicate that the true value of the property is not known. This is different to the default value which is used to indicate the most common value a parameter will take.

DM: Descriptive metadata. Data which can be placed in the file to describe the essence.

Duration: An integer number that defines the length of a property in edit units.

Edit rate: A rational number that specifies the units used to specify the duration of components in a track; the edit rate is the number of units that equal one elapsed second. In a track which describes an essence container, the edit rate is usually chosen to be the number of editable units per second.

Edit unit: A period of time equal to 1/(edit rate).

Editable unit: The smallest portion of the essence container which can be edited such as a field or frame.

Essence: The raw video, audio and data streams to be contained and described by MXF.

File package: The name for a logical grouping of structural metadata sets giving information about the essence in the essence container. If a file package is not referenced by a material package then it provides historical metadata annotation and shall not contain any essence.

Framework: The name for a number of related sets of MXF descriptive metadata.

GOP: Group of pictures — an MPEG coding parameter.

Grid size: The size of the KAG.

Hex: Hexadecimal (base 16) number format. A hex value is represented in MXF documents as NNh or 0xNN or NN.NN.NN.NN where NN is a hexadecimal digit.

IndexSID: Index stream identification (index of essence container).

Index table: A lookup table which converts a desired time offset on the timeline to a byte offset within the file.

Inheritance: Creating a new object by taking the items (properties) and behaviors of a parent object and adding to them and / or overriding them.

KAG: KLV alignment grid. A notional byte spacing which may be used in the file to align KLV items

KLV: Key-length-value encoding rules in SMPTE 336M

Link: A relationship between two properties using a numerical value of a defined type. The link is made when the two properties have the same value.

Local set: A set where each element has a (short) locally unique tag value — see SMPTE 336M.

Lower-level source package: A source package (file package or physical package) which is not directly referenced by a material package in this file. These source packages are used for historical annotation (or the derivation information) of the top-level file package(s).

Material package: A collection of tracks which generally represents the output timeline of the file.

MPEG: Moving picture experts group coding standard — see references.

NDE: Number of delta entries in an index table.

NIE: Number of index entries in an index table.

NSL: Number of slices in an index table minus 1.

Origin: An integer number that specifies the chosen zero point on a track, in edit units, from which relative times are measured.

Pack: A grouping of KLV elements defined in SMPTE 336M.

Package: The name for a logical grouping of structural metadata sets.

Partition: A logical separation of the MXF file. Each partition contains a single part of the file and different partitions may be multiplexed together. A tool for dividing the file into useful portions.

Physical package: A package with a physical essence descriptor to describe a physical entity such as a tape. A physical package may only be used for historical annotation in MXF.

Raster: A defined signal scanning system. Usually a standard — may be interlaced or progressive.

RIP: Random index pack.

RGBA: Red green blue alpha.

[RP 210 text]: The definition of a term copied from the SMPTE Dictionary RP 210 at the time of balloting this specification. This text is informative in this document. RP 210 remains the defining document. This text is provided where the specific text in the MXF document constrains the RP 210 definition such that the wording might appear different between the two documents.

Sample rate: The field or frame rate of essence container (not the video clock rate).

Sequence: A metadata structure which allows components (e.g., SourceClips) to be placed on a track.

Set: A grouping of KLV elements defined in SMPTE 336M.

Set Key: A SMPTE set Universal label. A 16-byte registered value.

SourceClip: A metadata structure which allows a portion of one track to be referenced by another. For example to describe a portion of the stored video which is placed on the output timeline.

Source package: An abstract class from which file package and physical package derive.

Start position: An integer number that defines the offset into an essence track, relative to the origin, in edit units which defines the desired start point of the essence

Structural: The name for metadata which relates to the structure and capabilities of an MXF file.

Tag: Local tag value used to identify elements in a local set. The tag byte-order is big-endian.

Time code: An annotation of elapsed time along a track. This may be created to be be numerically equal to time code standards such as SMPTE 12M. It can be used to convert position along a track measured in edit units to a count of hours, minutes, seconds and frames.

Top-level file package: A file package which is directly referenced by a material package in this file. This is the only type of file package which may describe stored essence.

UID: Unique identifier — a generic term which may be a UL, UUID, UMID, etc.

UL designator: This is a reference to the SMPTE label in the SMPTE metadata dictionary (SMPTE RP 210) which defines the property or term. The dictionary is the normative reference for these items and any explanatory text in this document is informative only.

UML: Unified modeling language.

Unicode: A form of character coding that allows a wide range of characters and ideograms to represent most major languages.

Universal set: A set where each element in the set has a full Universal label value — see SMPTE 336M.

VBE: Variable bytes per element. Used to describe compression coding styles.

VBR: Variable bit rate.

3.2 Simple data types

Simple data types used in the MXF format are given in this section.

BER length: A length value in bytes used to code a KLV triplet.

Boolean: 1-byte value with the logical values: zero == FALSE, non-zero == TRUE. Note for compatibility with older AAF SDK implementations, it is recommended that MXF file writers write the value 01h for TRUE values

Int8: Signed 8-bit integer.

Int16: Signed 16-bit integer.

Int32: Signed 32-bit integer.

Int64: Signed 64-bit integer.

Length: The Int64 value of the length (duration) measured in edit units of a piece of essence such as a SourceClip. Negative values are reserved for indicating distinguished values and shall not be used to indicate valid lengths.

Package ID: A UMID to uniquely identify a package or a zero value used to terminate a reference chain.

Position: The Int64 value used to locate a specific point along a track. Since properties of type position are relative to a chosen zero point, negative values may occur.

StrongRef: "One-to-one" relationship between sets and implemented in MXF with UUIDs. Strong references are typed which means that the definition identifies the kind of set which is the target of the reference.

UInt8: Unsigned 8-bit integer.

UInt16: Unsigned 16-bit integer.

UInt32: Unsigned 32-bit integer.

UInt64: Unsigned 64-bit integer.

UL: Universal label (SMPTE 298M).

UMID: Unique material ID (SMPTE 330M).

UUID: Universally unique identifier according to ISO 11578.

Version type: A Uint16 version number (NOTE: The number is created from major and minor version numbers using major*256 + minor)

WeakRef: "Many-to-one" relationship between sets implemented in MXF with UUIDs. Weak references are typed which means that the definition identifies the kind of set which is the target of the reference

The normative definition of item ULs is given by SMPTE RP 210. The data type of certain item ULs is itself a UL which is an enumeration of known values for the Item. The SMPTE labels registry is the normative reference for these values which are listed in RP 224.

3.3 Compound data types

Compound data types that are created from combinations of simple data types are given in this section. Arrays and batches are usually preceded by a name (e.g., IndexEntryArray is an ordered array of IndexEntry values).

Array: A compound type comprising multiple individual elements. The elements are ordered, the type is defined, the count of elements is explicit and the size of each element is fixed and explicit. NOTE – MXF decoders should ensure they use the size of each element as indicated in the file, to extract individual array elements. This ensures compatibility with any future extensions of MXF.

Batch: A compound type comprising multiple individual elements. The elements are unordered, the type is defined, the count of elements is explicit and the size of each element is fixed and explicit. NOTE – MXF decoders should ensure they use the size of each element, as indicated in the file, to extract the individual batch elements. This ensures compatibility with any future extensions of MXF.

DataStream: A string of data or metadata elements. The type of data or metadata element and the length of the string is defined elsewhere.

Rational: A pair of Int32 values where the first is the numerator and the second is the denominator (e.g., for an aspect ratio of 4:3, the number would appear as 00.00.00.04.00.00.00.03 in hexadecimal format).

Strings: Strings are created from individual characters defined either as ISO 7-bit characters (as used in SMPTE RP 210) requiring 1 byte per character, or as unicode UTF-16 characters requiring 2 bytes per character. In the case of UTF-16 characters expressing ISO 7-bit characters, an inspection of every byte will show each 2-byte pair as a null byte and a character byte. Byte order is specified as fixed big endian. The number of bytes allocated to this string is given by the KLV encoding. There is no requirement to terminate each string with a zero value. However, if the length of the string information is less than the space allocated, the string shall be terminated with a zero value

Timestamp: A time and date item according to the Gregorian calendar comprising, in order, Year: [Int16], Month: [UInt8], Day: [UInt8], Hour: [UInt8], Minute: [UInt8], Second: [UInt8] and mSec/4: [UInt8]. A value of '0' for each and every field identifies a timestamp value of 'unknown'. This value should not be used unless unavoidable.

UTF-16: Unicode characters coded with 16-bits; i.e., 2-byte characters.

ProductVersion: Comprises 5 values of UInt16 indicating major, minor, patch, build and release version numbers. These describe version of the tool that created or modified the file. The specific use of the first four values shall be defined by the tool. The "release" number is enumerated as follows:

0 = Unknown version, 1 = Released version, 2 = Development version, 3 = Released version with patches, 4 = Pre-release beta version, 5 = Private version not intended for general release.

xxxBatch: An unordered batch of individual elements preceded by a header of 8 bytes. The first 4 bytes define the number of elements to follow and the second 4 bytes define the length of each element, both represented as UInt32 coded as big-endian.

xxxArray: An ordered array of individual elements preceded by a header of 8 bytes. The first 4 bytes define the number of elements to follow and the second 4 bytes define the length of each element, both represented as UInt32 coded as big-endian. The order of the individual elements in the array defines the sequence in which the referenced objects appear; for example, contiguous objects representing the timeline of a track.

3.4 Guide to the use of tables

Tables used in this standard use the following symbols to the left of the table to help identify the entries which link the metadata items together

Set key — top level;
↔ Set length — top level;
UID used as strong reference target or a strong reference to a UID;
Strong reference array or strong reference set — references to one or more UIDs.

The property names in the columns of the set tables have the following meanings:

Item name: The name of the property.

Type: The defined type of the data.

Len: The length of the value in bytes where known.

Local tag: The 2-byte tag of the data when encoded as a KLV local set.

UL designator: The designator portion of the UL as defined in SMPTE 336M that defines the SMPTE metadata dictionary entry. This does not apply in the case of the set key and set length.

Req'd: Indication of whether MXF encoder must supply, and what action MXF decoder should take.
Meaning: A description of the property.

Default: A default value which is commonly used by an encoder and should be used by the decoder if the property is not encoded in the set.

The following table defines the meanings of the required status of items in these annexes.

Entry in Table	Abbreviation	Encoder	Decoder
Required	Req	Shall encode	Shall decode
Encoder Required	E/req	Shall encode	May decode
Decoder Required	D/req	May encode	Shall decode
Best Effort	B.Effort	Should encode	Shall decode
Optional	Opt	May encode	May decode

4 Introduction

There are several parts to the MXF specification. This part is a normative definition of the format of an MXF file. MXF operational pattern specifications define how the MXF file format specification can be configured to provide a defined application. One or more normative documents define a descriptive metadata scheme as "plug-in" to the MXF file format specification. A number of individual normative documents define both the essence containers that may be used in an MXF body and the mappings of essence elements into an essence container (section 9.1 below). In addition, the MXF engineering guideline provides an introduction and general engineering description of MXF including the document layering.

4.1 Structure of this standard

This standard starts with an overview of the whole specification and then defines each of the different elements in turn.

Section 5 – Overall specification: gives an overview of the whole MXF specification and partitioning rules.

Section 6 – Partitions: gives details of the fine level structure of MXF including the header partition, the body partition(s) and the footer partition.

Section 7 – Operational patterns: shows how interchange can be aided by defining how the file is structured.

Section 8 – Header metadata: describes the header metadata including a definition of the structural metadata and the mechanism for adding descriptive metadata schemes as plug-ins.

Section 9 – Body: defines the requirements of the essence containers that can be used as an MXF file.

Section 10 – Index tables: defines how to locate essence elements within the file body.

Section 11 – Random index pack: defines how the partitions within an MXF file can be rapidly accessed.

Annexes – Detailed byte values of structural metadata sets and packages.

4.2 MXF file

An MXF file starts with a file header, is followed by a file body and is completed by a file footer. This standard defines the data structure of the MXF file, including a full definition of the file header and file footer. Associated documents are required to provide the full specification of an MXF file.

Operational pattern standards define the specific details of every component in an MXF file together with the framework needed to define the relationships between the components. At least part of these associated documents will define the structural metadata part of the header metadata such that it can be accepted as a compliant subset of the AAF metadata structure.

Essence container standards provide the specifications for the picture, sound and data essence containers that can be used in a file body. The file body may contain one or more essence containers. This document does not normatively define each individual essence container that may be placed in the file body, but leaves that to associated MXF essence container specifications. This document does, however, define the requirements for each essence container specification in order to meet the needs of program interchange. Each essence container specification is written as a stand-alone document or document set that must meet the requirements set out within this document in order for it to be accepted as a compliant MXF essence container.

Descriptive metadata standards define optional editorial metadata that enhance the usability of the essence content of an MXF file. MXF files include SMPTE Universal labels in the file header that provide early identification of the operational pattern, any descriptive metadata and the essence containers in the file body. MXF essence containers may include picture, sound and data essence as well as related metadata. All essence and metadata types may be constant data rate or variable data rate.

This document is written as though the essence data is embedded in the file body. Later parts of this document describe how essence data may be located outside the file.

5 Overall specification

This section defines the common parts of every MXF file. Further detail on each item is given later in this specification.

5.1 Overall data structure

The overall structure of an MXF file is shown in figure 1 and figure 2. Note that the relative lengths of individual components shown in the figures are not to scale.

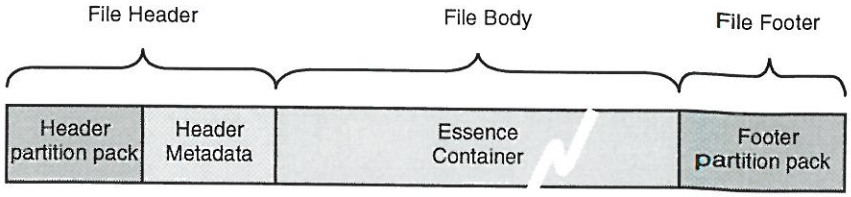

Figure 1 – Overall data structure of a simple MXF file

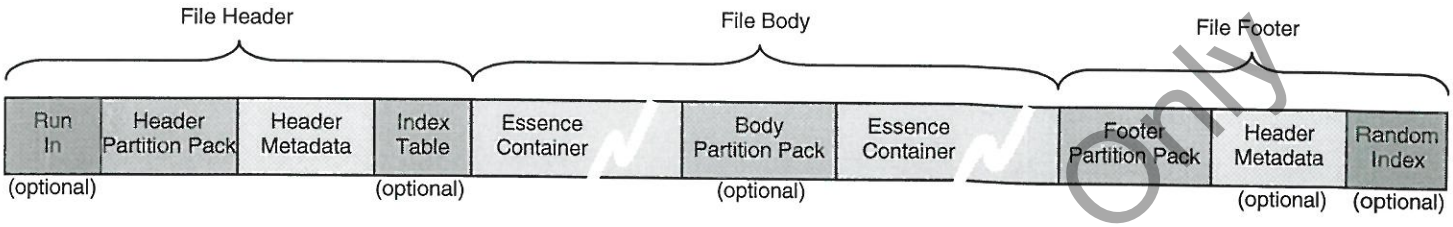

Figure 2 – Overall data structure of an MXF file with some optional components

The MXF file shall be constructed as follows:

5.1.1 File header

The file header shall always be present at the start of every MXF file. As shown in figure 2, it includes an optional run-in, a header partition pack, header metadata and optionally an index table. There may optionally be fill items in the file. The rules for insertion of fill items are given in section 5.4.

NOTE – Although the optional run-in is a part of the file header, it is not a part of the header partition (see section 5.2.1).

5.1.2 File body

The file body provides a mechanism for embedding essence data within MXF files. The file body shall have one or more essence containers. If there is more than one essence container, they shall be multiplexed together using partitions (see section 5.2). The associated MXF essence container specifications define how each essence element in the container shall be KLV encoded. Each essence container specification defines how index tables should be implemented and identifies the essence descriptors that are required to describe each essence element.

MXF applications may be created which embed all the essence data within the file. Other applications may require some or all of the essence data to be externally referenced. The mechanisms for this are further explained in section 6, section 7, and in annex B.

MXF metadata only files may have no file body and hence no essence containers.

In the operational pattern definitions, there may be constraints placed on the capabilities of the file body.

NOTE – Examples of the nature of these constraints are described in the MXF engineering guideline which is referenced in the annex F.

5.1.3 File footer

The file footer shall be located at the end of the file. As shown in figure 2, it includes a footer partition pack, an optional repetition of the header metadata sets and an optional random index pack (see section 11). The file footer may also include optional index table segments.

NOTE – Although the optional random index pack is a part of the file footer, it is not included as a part of the footer partition.

The file footer shall be present unless a specialized operational pattern is being used which defines it to be absent or optional.

5.2 Partitions

An MXF File shall be divided into a number of partitions:

one header partition which shall be followed by
zero or more body partitions, the last of which shall be followed by
zero or one footer partition

Partitions logically divide the file to allow easier parsing, to help streaming and to manage the creation of index tables (which, in turn, ease random access in a storage system).

An MXF file may contain many different essence containers and partitions help to manage them.

5.2.1 Partition rules overview

Each essence container shall have an identifier called the BodySID which is unique within each file. Each essence container may be segmented and distributed over one or more partitions. Each header or body partition shall have data from only one essence container. The footer partition shall not have essence container data. All partitions containing data from a particular essence container shall have the same value of BodySID. The order of the essence container data after segmentation into partitions shall be the same as the order of the unsegmented essence container data.

The essence container data may be indexed with an index table. Each index table shall have an identifier called the IndexSID which is unique within each file. An index table shall comprise one or more index table segments. All segments of a particular index table shall have the same value of IndexSID. Each partition shall include zero or more index table segments, each of which has the same value of IndexSID. The order of the index table segments shall present the indexing data in the same as the order as the indexing data of an unsegmented index table.

The EssenceContainerData set is defined in annex A.4. In this data set, the relationship between the index table and essence container shall be defined by linking the values of an IndexSID with a BodySID. The following three rules apply:

1. If a partition has an essence container segment of a given BodySID, then the only index table segments permitted in the same partition shall be those having the linked IndexSID value.

2. If there is no essence container segment in a partition, then index table segments of any single IndexSID value may be placed in the partition.

3. If there are no index table segments in a partition, then an essence container segment of any single BodySID value may be placed in the partition.

A partition pack shall define the start of every partition. Every partition shall be defined either as a header partition, a body partition or a footer partition depending on where it is located within the file. For each kind of partition, the required order of the components following the partition pack is defined in the text below and illustrated in figure 3.

The header partition shall be located at the start of the file. A header partition starts with a header partition pack and shall be followed by the header metadata, optional index table segments and optionally by the whole, or the first part, of an essence container. In certain specialized operational patterns, it may be preceded by an optional run-in as shown in figure 2.

If a file has body partitions, then each shall comprise a body partition pack followed optionally by a repeat of the header metadata, optional index table segments and optionally by a whole or segment of an essence container.

Where present, a footer partition shall be located at the end of the file. It shall comprise a footer partition pack followed optionally by the header metadata, and optionally by index table segments. The footer partition may optionally be followed by a random index pack.

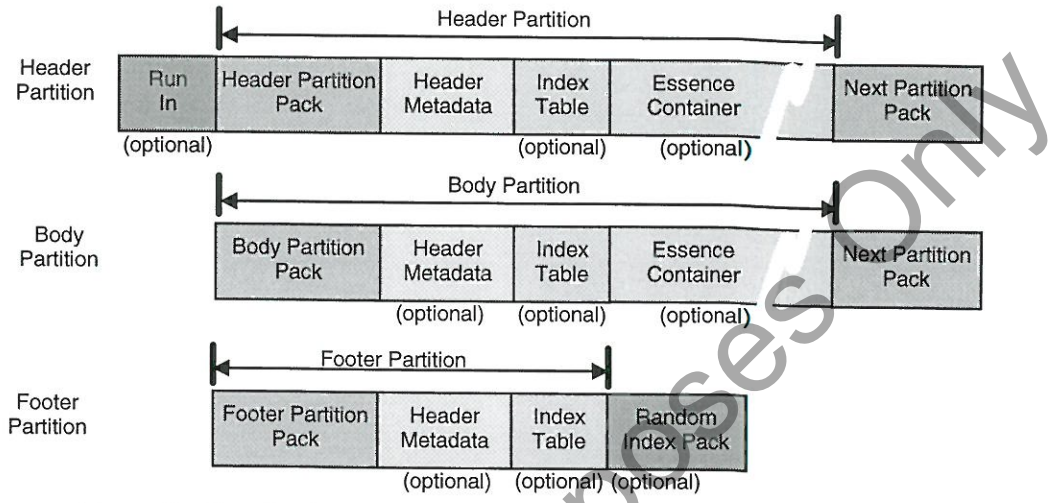

Figure 3 – Required order of file components in each partition kind

5.2.2 Partition rules summary

The logic describing partitions, essence containers and index tables can be summarized as follows:

Essence containers:

1. A file body can have zero or more essence containers.
2. Each essence container is identified with a file unique BodySID.
3. The essence container data may be segmented over one or more partitions.
4. The essence container partitions shall be in sequence within the file, though not necessarily in adjacent partitions.
5. Each partition identifies its essence container segment with the appropriate BodySID value.
6. Partitions with different BodySID values may be multiplexed together.

Index tables:

1. Each essence container has an optional index table.
2. Each index table is identified with a file unique IndexSID.
3. Each index table may be divided into index table segments which are distributed into partitions.
4. Each index table segment shall be in sequence within the file, though not necessarily in adjacent partitions.
5. Each partition identifies its index table segment with the IndexSID value of the index table.
6. Partitions with different IndexSID values may be multiplexed together.

Implementers should be aware that, as illustrated in figure 3 and figure 4, the first essence container segment in the file may be in the header partition. In files with no body partitions, the whole essence container shall be in the header partition.

NOTE – Multiplexed means putting different partitions one after the other whereas "interleaved" means that the essence container itself has different components which are interleaved on a time division basis. It is perfectly valid to create an MXF file with two interleaved essence containers (e.g., a DV generic container and a D-10 generic container). These may be placed in two large partitions in which all the type D-10 information follows all the DV information or alternatively the partitions may be made smaller and multiplexed together. Providing the above rules are met, the file will be valid.

information or alternatively the partitions may be made smaller and multiplexed together. Providing the above rules are met, the file will be valid.

5.2.3 Partition status

A partition may be open or closed. It may be complete or incomplete according to the following definitions. The intention is that open/closed indicates that the partition contains a partial or final version of the header metadata. Complete/incomplete indicates whether all the (best effort) values were known at the time of creating the file. Section 6 defines the partition pack values for all partitions.

Open or closed status —

Open: An open partition is one in which required header metadata values have not been finalized (required values may be absent or incorrect) and may not have the same values as the closed partition(s). Header metadata in open partitions may change in later repetitions.

Closed: A closed partition is one in which required header metadata values have been finalized (all required metadata is present and correct). All closed partitions that contain header metadata shall have identical header metadata sets.

Incomplete or complete status —

Incomplete: An incomplete partition is one where header metadata exists and some *best effort* metadata properties have been flagged as unknown (by setting to the appropriate distinguished value).

Complete: A complete partition is one where either header metadata is absent in this partition or where the header metadata exists and all best effort metadata properties have been correctly completed.

Closed partitions may be complete or incomplete.

5.2.4 Status of an MXF file

A closed MXF file is one in which there is either a closed header partition containing header metadata or a closed footer partition containing header metadata. An open MXF file is one which does not meet this condition. Open files may have Header Metadata values which are required, but have not been finalized; i.e., their values may not be correct. This state is defined to allow real-world applications to create and process MXF files during the recording process.

MXF files shall be closed.

Note that MXF decoders are not required to be able to decode open MXF files.

5.2.5 Header partition

The header partition shall be the first partition in any MXF file and there shall be only one such partition. The header partition may be open or closed. The header partition may be complete or incomplete. If there are no body partitions, then all of the file body shall be located in the header partition.

5.2.6 Optional body partitions

The optional body partition is a partition that may be present within the file body. There may be zero or more body partitions in an MXF file.

Any body partition may be of type open or closed. A body partition may be complete or incomplete. The FooterPartition value in the body partition pack may be zero or a non-zero (valid) value. There are many applications for body partitions, including the ability to multiplex together different essence containers. Among

these applications are to provide a segmentation mechanism for index tables and to assist recovery from incomplete transfers. Where possible, body partitions must start and end at edit unit boundaries of the essence containers.

The overall structure of file components with a single body partition is as shown in figure 4. An example with extra body partitions is shown in figure 5.

5.2.7 Footer partition

The footer partition shall be the last partition in any MXF file and there shall be zero or one such partition. A footer partition shall be of type "closed" as defined in section 5.2.3. A footer partition may be complete or incomplete as defined in section 5.2.3. The FooterPartition value in the footer partition pack shall have the correct value.

The presence of a footer partition shall be used to identify that there is no essence container data beyond this point in the MXF file. The footer partition shall be present unless a specialized operational pattern is used which requires it to be absent or optional. In this case, other mechanisms shall be used to identify that the file is closed and complete. Such mechanisms shall be defined by the specialized operational pattern specification. The presence and location of a footer partition shall be indicated by a non-zero value of the FooterPartition item in any closed partition pack within the file.

5.2.8 Essence container and partition relationships (informative)

In a simple MXF file, there is only one essence container and optionally one index table. These can be divided into a header partition and one or more body partitions as determined by the application. Figure 4 shows a file having a single embedded essence container divided into two segments (a and b) with an associated index table and optional repetitions of the header metadata.

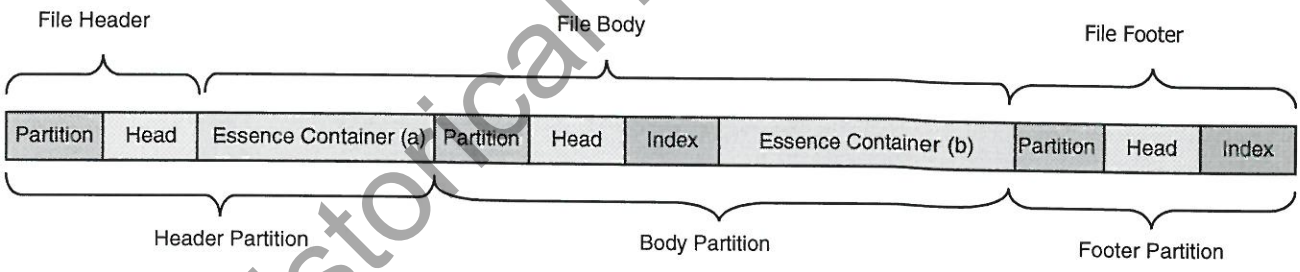

Figure 4 – MXF file containing one essence container

In more complex MXF files, there may be multiple essence containers, each with an optional index table (which may be segmented). Each essence container and its index table are mapped into partitions as defined by the application, subject to the rule that any partition may have data from only one essence container and index table segments from only one index table.

An example file containing two essence containers (1 and 2) with associated index tables (1 and 2) is shown in figure 5. Note the insertion of additional body partitions to permit header metadata repetition, the insertion of index table segments and the multiplexing of different essence containers.

The first body partition in figure 5 shows both an essence container segment and an index table segment in the same partition. The partitioning rules defined in section 5.2.1 ensure that the index table segments 1a and 1b do not appear in the partition with a segment from essence container 2. Index table segment 1b appears in a body partition of its own. Index table segment 2 appears in the footer partition.

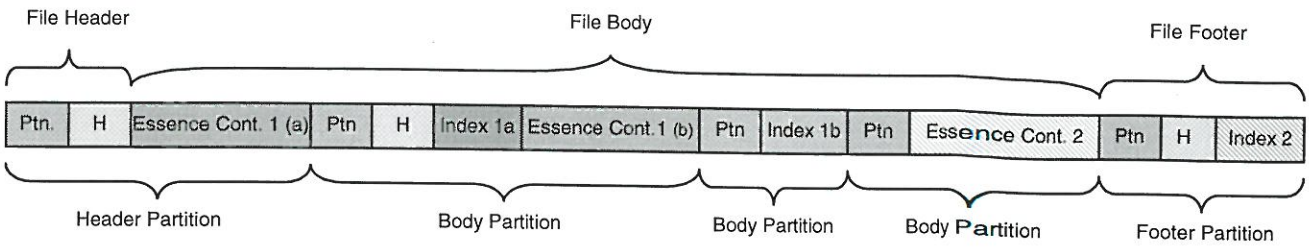

Figure 5 – MXF file containing two essence containers

5.3 KLV coding

The KLV coded data packets in an MXF file may be individually coded data items or data groups coded as sets or packs as defined in SMPTE 336M. This section defines KLV coding issues specific to the implementation of MXF.

NOTE – A brief introduction to KLV coding is given in the MXF engineering guideline and a full review of SMPTE KLV coding is given in the July 2000 issue of the SMPTE Journal.

5.3.1 KLV coding sequence

MXF files are defined by a contiguous sequence of KLV coded data packets. All data within an MXF file (except for the optional run-in) shall be KLV coded with no gaps. The number of KLV packets in an MXF file is variable and depends on the number of partitions in the file, the number of metadata sets in the header metadata, the method of coding the index tables and the length of the essence container(s) in the file body. The KLV coding rules for each essence container are given in the individual essence container specifications. Where the essence container allows, there should be at least one KLV packet for each edit unit of essence container data (typically one frame or one group of pictures in duration).

5.3.2 KLV coded dark components

Since extra components may be added to the MXF specification at any time in the future, MXF decoders must be able to parse any KLV packet and extract the recognized packets whilst ignoring KLV coded data packets that are not recognized.

In particular, some variants of MXF files may include specialized data encapsulated in KLV packets to which the keys of the KLV syntax are not specified in an MXF specification (known as dark metadata). In many cases, skipping over such data items or data sets will have little or no significant effect. However, in some cases, a KLV coded packet may be significant, for example the inclusion of a metadata set which defines (for example) audio cross-fade metadata that has no meaning to a decoder which complies only to a simple operational pattern.

The rule for MXF decoding is to always follow the KLV syntax whether or not the key is recognized. Unrecognized keys having values that cannot be determined by this decoder should, therefore, use the length field to skip over the value of the data packet. Decoders may optionally indicate such keys to the user for diagnostic purposes.

5.3.3 KLV fill items

There are a number of areas where a MXF file may use the KLV Fill item. This item is defined in SMPTE 336M (as an empty metadata item). This is a KLV coded item where the value is comprised of null or meaningless data.

This KLV fill item may be used for one of several reasons including, but not limited, to the following:

- To pad the header metadata so that the following KLV packet lies on a convenient storage boundary for ease of access.

- To pad header metadata sets to allow variable length strings in individual metadata items to be re-written in place without increasing the overall length of the header metadata.

- To pad the header metadata to allow space for an optional index table to be inserted before the first body item.

- To pad the end of a body editable unit to a known KAG boundary thereby ensuring the next editable unit is aligned to the next KAG boundary as described in section 5.4.1.

- To pad each body editable unit to a constant length in the event of a variable length compression coding scheme.

This list is not exhaustive and it does not preclude other reasons for using the KLV fill item. The shortest KLV fill item is 17 bytes long. This is constructed using a key and a short form BER coded length of zero. There are no value bytes because of the zero length. The maximum KLV fill item length is governed by the longest length field which can be coded.

KLV fill items shall not be placed adjacent to each other in a file. Any contiguous KLV fill items shall be concatenated into a single KLV fill Item

It is not possible to make a KLV fill item which is less than 17 bytes in length. If KAG alignment is required and a KLV gap of less than 17 bytes exists, it is necessary to round up the size of the fill item to the next nearest KAG so that the KLV fill item is 17 bytes or longer. For example, if 3 bytes are required to obtain alignment with a 4096 byte KAG then a KLV fill item with 16 bytes for the key, 5 bytes for the length and (4096+3-16-5)= 4078 bytes for the value could be used.

Although the length field of a KLV fill item indicates the length of the value, the value itself is provided just to fill data space and shall have no defined meaning. By default, the value field of any KLV fill item should be composed of bytes with a value of zero.

The KLV fill item is defined in the SMPTE metadata dictionary (SMPTE RP 210).

5.3.4 KLV lengths

Lengths shall be BER encoded using short or long-form encoding as specified by SMPTE 336M. In no event shall MXF encoders use long-form coding that exceeds a 9-byte BER encoded length.

MXF decoders shall be SMPTE 336M compliant and must respond to short or long-form BER encoding as received. Decoders shall not rely on a fixed number of bytes for length fields. Note that in MXF files, long form BER encoding may be used even for length values less than 128.

NOTE – The length value "80h" shall not be used for MXF encoding.

5.3.5 Local set lengths

The key of a local set shall comply with the values defined in SMPTE 336M. By default, most local sets are coded with 2-byte tags and 2-byte lengths, but decoders shall not assume that this is the only case.

NOTE – Tag and length fields of all items in local sets are always coded as big-endian (MSB first), non-BER values.

5.3.6 Variable-length pack lengths

The key of a variable-length pack shall comply with the values defined in SMPTE 336M. By default, most variable-length packs are coded with a 2-byte length field, but decoders shall not assume that this is the only case.

NOTE – The length field of items in variable-length packs is always coded as a big-endian (MSB first), non-BER value.

5.3.7 MXF keys and Universal labels

SMPTE keys and Universal labels (ULs) may be any length in steps of 4 bytes; the length being defined in the second byte of the key or UL according to SMPTE 298M. MXF files shall have keys and ULs that are 16 bytes long and no other lengths are permitted.

5.3.8 KLV recursive grouping (as defined in SMPTE 336M)

SMPTE 336M places no limits on the depth of recursive grouping allowed in KLV coded data sets (i.e., sets contained within sets) to provide a hierarchical structure. However, MXF files shall only contain sets or packs with individual data items unless otherwise specifically defined in a descriptive metadata or essence container specification. Note that the use of recursive group structures could prevent backwards compatibility with existing implementations and is therefore prohibited in MXF defined sets. Dark metadata is not constrained.

Parts of an MXF file (with particular reference to the header metadata) may contain a logical grouping of sets or packs that defines a hierarchical structure. The method of coding this structure in the MXF file format specification is to provide each data set with an instance UID metadata item that provides each set in the structure with a unique identification. This allows any set to be referenced from any other set. It provides the same logical effect as recursive grouping, and results in all sets being coded in a single layer as a contiguous sequence of sets.

Section 8.3 describing the header metadata provides more details of coding logical data structures.

5.3.9 Primer Pack – KLV local sets, dark metadata and MXF extensions

MXF local sets in the header metadata use 2-byte local tags as a shorthand for the 16-byte Universal label which fully identifies any item within a set. In MXF these local tags are defined as having the scope of the partition in which they occur. This means that all local tags are unique within a partition. To ensure the uniqueness of these tags, a primer pack is provided which defines the mapping between the local tags and their 16-byte Universal label equivalents.

Implementers who insert metadata extensions in MXF defined sets shall use the primer mechanism for generating local tags. This will ensure that no two extended metadata items in a set will use a common local tag for different data items. Metadata extension schemes may be added using other mechanisms such as Universal sets, Universal items, local sets with different sized tags and lengths.

Full details of the primer pack are given in section 8.2.

5.4 MXF Encoding Requirements

The contents of an MXF file may be viewed as a multiplex of data items or objects:

Run-in — only used in specialized operational patterns.
Partition metadata — the metadata used to describe the structure of each partition in the file and defines the header partition, any body partitions and the footer partition.
Header metadata — metadata describing the file body, its essence, its structure and possibly its meaning.
Index tables — to convert from a position in a track to a byte offset within the essence container.
Essence containers (when present) — the essence data of the file body.

Random index pack — used to rapidly locate the partitions in the file.

All MXF data types shall be KLV coded (with the sole exception of the optional run-in).

The ordering of the various components of a partition and the use of the fill item between these components shall follow the rules given below.

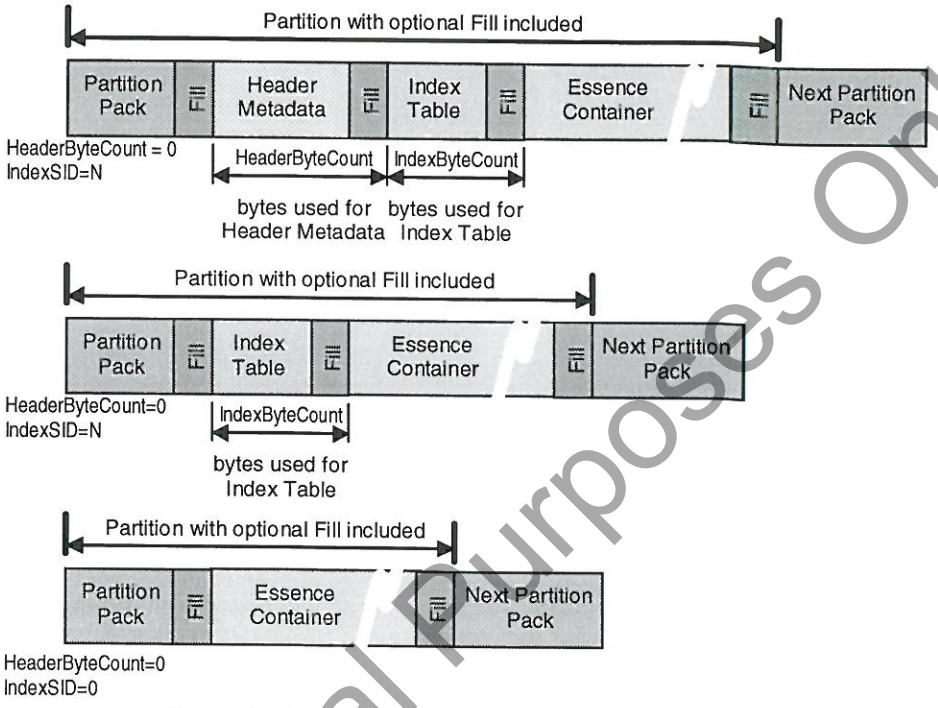

Figure 6 – Ordering in a partition and the use of fill

Each partition shall comprise the following items (in the order defined):

1. A partition pack optionally followed by one KLV fill item to ensure grid alignment when required or to provide padding.

2. Optional header metadata. If present, it may be followed by one KLV fill item to ensure grid alignment when required or to provide padding.

3. Zero or more index table segments. If present, each may be followed by one KLV fill item to ensure grid alignment when required or to provide padding.

4. Zero or more bytes of essence container data. If present, KLV fill items may be added to provide grid alignment when required or padding as required by the essence container.

5.4.1 KLV alignment grid (KAG)

In certain operations, it may be desirable to align certain KLV elements to specific byte boundaries. This can be achieved with the insertion of KLV fill items to ensure the desired items are aligned. One example might be a VBR picture stream with a peak rate of 1-Mbyte/picture. It might be desirable to align each picture on a 1-Mbyte boundary despite the increase in file size. KLV Fill items can be used to create an alignment grid for this purpose.

Each partition may align certain KLV elements to a KLV alignment grid (KAG). The KLV alignment grid defines the number of bytes between gridlines to which the first byte of the Key of certain KLV elements may be aligned. The first gridline in any partition is the first byte of the key of the partition pack that defines that partition. Specific KLV items that may be aligned to a gridline include the preface set (of the header metadata) and the index table segment set. Individual essence container specifications shall define which KLV elements shall lie on a gridline. All partitions that use the KAG shall ensure that the end of that partition is padded to a gridline defined by that partition.

A KAG value of "0" defines that the grid value is undefined; e.g., there may once have been a KAG but it may no longer be valid for some reason. A KAG value of "1" defines that there is no grid; i.e., byte alignment. Any KAG value between, and including, "2" and "1048576" defines a valid grid size. A KAG value in excess of "1048576" (1 Mbyte, 2^{20}) is not permitted.

NOTE – The upper limit of 1 Mbyte is provided to minimize receiver buffer requirements.

If grid alignment is used, then all partitions of the same type in the file should use grid alignment. The KAG size is specified in every partition pack and is valid for that partition only. The KAG size shall be constant for all partitions with segments from the same essence container. However, specific applications may require a different KAG size value for different essence containers (for example, an MXF file that has a high resolution primary video and an alternate low resolution preview video might use different KAG sizes for the essence container of each video).

If the KAG size of any partition is changed, then the values of appropriate items within the partition (e.g., partition pack, index tables) should be changed accordingly. If an MXF application modifies the KAG size in a partition and the dependant partition items are not updated, then the KAG value should be set to 0. The KAG size parameter is a performance enhancement parameter. It is possible that some files may have the KAG value incorrectly set.

Essence container specifications may define a KAG value which MXF encoders should use, however, all decoders shall correctly parse a file whether the KAG parameter is correct or not.

NOTE – Application writers should be aware that incorrect values of KAG size may lead to performance degradation, especially when MXF files are transferred to and from certain classes of machine. Applications may also be degraded by the requirements of KAG implementation. Its use is therefore optional.

5.4.2 MXF byte order

This section defines the byte order of the value fields in KLV packets used in MXF files. The byte order of essence data as a value in a KLV packet shall be defined by the appropriate specification.

For all other parts of the MXF specification, all multi-byte values in any KLV packet shall be coded as big-endian (most significant byte first), wherever the value is sensitive to the byte order. Examples of simple multi-byte values affected by byte order are: UInt16, Uint32, Uint64 and UTF-16.

Compound data types shall have all component multi-byte values coded as big-endian.

5.4.3 Encoding constraints

This document constrains several aspects of MXF encoding. Other documents, such as operational pattern and essence container specifications, may further constrain aspects of MXF encoding for reasons particular to their requirements.

5.5 Run-in sequence

In certain specialized operational patterns, the header partition may be preceded by a non-KLV coded run-in. This is to allow synchronization bytes or "camouflage" bytes to be added at the front of the file in specialized applications.

One example of this use is when a broadcast WAV header is added to the start of the MXF file to make it look like a WAV file to a WAV machine and an MXF file to an MXF machine with no change in byte structure. In all other circumstances, there will be no run-in and the entire file must consist of only of KLV packets with NO gaps.

MXF decoders shall ignore the run-in sequence and parse the input data until either the first 11 bytes of the partition pack label has been located or the maximum run-in length has been exceeded.

The run-in sequence shall be less than 65536 bytes long and shall not contain the first 11 bytes of the partition pack label. The maximum length of the run-in prevents a decoder from searching through an excessively large non-MXF file if incorrectly applied to an MXF decoder.

The default run-in sequence shall have a length of zero.

MXF encoders may insert any necessary run-in sequence providing it conforms to the rules given above, and any rules given in the appropriate specialized operational pattern document.

5.6 Minimum decoder

The concept of a minimum decoder is useful in defining the minimum behavior of any device which claims to be MXF aware. The following is a list of required functionality:

1. It must locate the first Key in the file and determine that it is the key of the header partition pack (open or closed).

2. It must then locate, in the value fields of the header partition pack, the ULs for the operational pattern and the essence container, where present (some commissioning files may have no essence container and be metadata only).

3. It must determine if the operational pattern UL is compatible with the capabilities of this decoder.

4. It must further determine if the essence container(s) UL is compatible with the capabilities of this decoder.

5. It must have a defined behavior (such as reporting an error) if the operational pattern or essence containers(s) are incompatible with its capabilities.

Useful decoders that are compatible with the indicated operation pattern and essence container(s) will then perform additional operations as needed such as to verify the KLV syntax, verify the remaining partition packs, locate the header metadata in a closed partition, decode the contents of that header metadata, decode the contents of selected compatible essence container(s) and decode the contents of other parts of the file such as primer pack, index tables, descriptive metadata and the RIP (where present).

5.7 Strong/weak reference integrity (informative)

This section is a guidance note to point out that MXF decoders may not be able to find the target of all the references in an MXF file. This may be because the target is a dark set. This situation can occur for many reasons including additions to the MXF specification (extensions, new essence types etc.) or it may be because private additions to the specification have been included in specific files. In general, the following points should be observed:

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) shall be strongly referenced (directly or indirectly) to the preface set.

3. All strong references in a file shall match one and only one set.

4. Weak references may be made to "global definitions" which may be inside or 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, then it can be regarded as a global definition.

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

6 Partitions

6.1 Partition pack

Partition packs shall be KLV coded, fixed length packs as defined in SMPTE 336M. Header partition packs, body partition packs and footer partition packs, whether open or closed, complete or incomplete, shall be variants of partition pack.

The key of the partition pack is given in table 1.

Table 1 – Partition pack key value

Byte No.	Description	Value (hex)	Meaning
1	Object Identifier	06h	
2	Label size	0Eh	
3	Designator	2Bh	ISO, ORG
4	Designator	34h	SMPTE
5	Registry Category Designator	02h	KLV Sets & Packs
6	Registry Designator	05h	Fixed Length Packs (no length fields)
7	Structure Designator	01h	Set / Pack registry
8	Version Number	01h	Registry Version 1
9	Item Designator	0Dh	Organizationally registered
10	Organization	01h	AAF
11	Application	02h	MXF File Structure
12	Structure Version	01h	Version 1
13	Structure Kind	01h	MXF File Structure sets & packs
14	Set / Pack Kind	Sections 6.2, 6.3, 6.4	Header Partition, Body Partition or Footer Partition
15	Partition Status	Sections 6.2, 6.3, 6.4	Open and Incomplete (01h) Closed and Incomplete (02h) Open and Complete (03h) Closed and Complete (04h)
16	Reserved	00h	

Byte 15 provides four alternate partition pack key values for the combinations provided by an open or closed partition and a complete or incomplete header metadata as defined in section 5.2.3.

The data in a partition pack is defined in table 2 and the notes which follow. Note that the values shall appear in the order presented in table 2 because this is a pack.

Table 2 – Partition pack

Item Name	Type	Len	UL Designator	Meaning	Default
Partition Metadata	Pack Key	16	Table 1	Identifies a Partition Pack	
Length	BER Length	var (see 8.3)		Overall Length of Partition Pack	
Major Version	UInt16	2	03.01.02.01.06	Major Version number of MXF byte-level format (non-backwards compatible version number) [RP 210 A major version number. A change in a major version implies non-backwards compatibility]	0001h
Minor Version	UInt16	2	03.01.02.01.07	Minor Version number of MXF byte-level format (backwards compatible version number) [RP 210 A minor version number. A change in a minor version implies some measure of backwards compatibility]	0002h
KAGSize	UInt32	4	03.01.02.01.09	Size of the KLV Alignment Grid (KAG) for this partition, in bytes [RP 210 Size of the KLV Alignment Grid (KAG) for this partition, in bytes]	
ThisPartition	UInt64	8	06.10.10.03.01	The number of this partition in the sequence of partitions (as a **byte count** relative to the start of the Header Partition). [RP 210 The current number in a sequence]	
PreviousPartition	UInt64	8	06.10.10.02.01	The number of the previous partition in the sequence of partitions (as a **byte offset** of the start of the Previous Partition, relative to the start of the Header Partition). [RP 210 The previous number in a sequence]	0
FooterPartition	UInt64	8	06.10.10.05.01	The number of the last partition in the sequence of partitions (i.e. a **byte offset** of the Footer Partition, relative to the start of the Header Partition). [RP 210 The last number in a sequence]	0
HeaderByteCount	UInt64	8	04.06.09.01	Count of Bytes used for Header Metadata and Primer Pack. This starts at the first byte of the key of the Primer pack and ends after any trailing filler which is included within this HeaderByteCount. See Figure 6 [RP 210 Count of bytes used for the metadata in a file Header]	0
IndexByteCount	UInt64	8	04.06.09.02	Count of Bytes used for Index Table Segments. This starts at the first byte of the key of the first Index Table Segment and ends after any trailing filler which is included in the IndexByteCount. See Figure 6 [RP 210 Count of bytes used for index table segments]	0
IndexSID	UInt32	4	01.03.04.05	Index Table Segment Identifier in this partition. The value 0 defines that there are no Index Table segments in this partition. [RP 210 Index table stream ID]	0
BodyOffset	UInt64	8	06.08.01.02.01.03	Byte offset of the start of the Essence Container segment in this partition, relative to the start of the Essence Container [RP 210 Indicator for the position of a packet in a stream of packets]	
BodySID	UInt32	4	01.03.04.04	Identifier of the Essence Container segment found in	0

Item Name	Type	Len	UL Designator	Meaning	Default
				this partition. The value 0 indicates there is no Essence Container data in this partition. [RP 210 Essence (or its container) stream ID]	
Operational Pattern	UL	16	01.02.02.03	Universal Label of the Operational Pattern to which this file complies (copy of Preface Set value) [RP 210 Specifies the SMPTE Universal Label that locates an Operational Pattern]	
EssenceContainers	ULBatch (Essence Containers)	8+ 16n	01.02.02.10.02.01	An unordered Batch of Universal Labels of internal Essence Containers used in or referenced by the top level File Package this file Individual UL values are listed in RP224 [RP210 Batch of universal labels of all essence containers in the file]	

The first two items (**Major Version, Minor Version**) shall be the same in every partition pack in a file. Minor version is used to indicate the precise revision of this document, both prior-to and post standardization.

ThisPartition specifies the byte offset of the start of this partition relative to the start of the header partition. Regardless of any run-in, the "ThisPartition" value in the header partition is zero.

INFORMATIVE NOTE – This definition means that a change in the size of any run-in will not affect the addressing of the rest of the file.

PreviousPartition specifies the byte offset of the start of the previous partition relative to the start of this partition.

FooterPartition specifies the byte offset of the start of the footer partition relative to the start of the header partition. The rules for setting this parameter are given in the sections on header partitions, body partitions and footer partitions (see section 5.2).

HeaderByteCount specifies the number of bytes used for the header metadata in this partition. This starts at the first byte of the key of the primer pack and ends after any trailing KLV fill item which is included within this HeaderByteCount (see figure 6). The values shall be zero if there is no header metadata in this partition.

IndexByteCount specifies the number of bytes used for index table segments in this partition. This starts at the first byte of the key of the first index table segment and ends after any trailing KLV fill item which is included within this IndexByteCount (see figure 6).

IndexSID specifies the identifier of the index table segment(s) in this partition. The value shall be zero if there are no index table segments in this partition.

BodyOffset specifies the byte offset of the essence container segment in this partition relative to the start of the essence container with the specified BodySID. This value includes any KLV fill items that are part of the essence container. An example is shown in figure 7. This value provides a tool for building index tables and for recovering partial file transfers.

BodySID specifies the identifier of the essence container in this partition. The value shall be zero if there is no essence container segment in this partition.

OperationalPattern is a UL that identifies the operational pattern for this MXF file (see section 7).

EssenceContainers is a batch of ULs that identifies the different essence container types used in or referenced by this MXF file. This should be complete where possible. Note that for any given container type, there may be more than one label if different mappings are used for picture, sound, data, etc. If this partition is closed, the values shall be complete and correct. There shall be no duplicate values. The length the partition

pack includes the length of this batch. MXF files with different EssenceContainers may have different lengths of partition pack.

(NOTES **(i)**. According to the terminology in SMPTE 336M this is still classed as a fixed length pack; **(ii)** It is perfectly valid for this batch to have an overall length of 8 bytes with its Item count set to zero indicating that a file has metadata but no essence or that the essence is external; **(iii)** It may not be possible for this information to be fully provided by all MXF encoders and hence decoders should not fail if this information is empty or missing).

The essence container ULs placed in this batch should be for those essence containers which appear internally in the file. These essence container ULs shall have the same value as those found in the essence descriptors for the file package(s) directly referenced by the material package(s) in the file.

6.2 Header partition pack

In the default case of a run-in sequence length of zero, the file shall start with the header partition pack. The header partition shall be the first partition in an MXF file.

6.2.1 Header partition pack key

The 16-byte SMPTE Universal label of the header partition pack both identifies the file as an MXF file and acts as a Key for KLV coding of the header partition pack.

The header partition pack key shall have the following value:

Table 3 – Specification of the header partition pack key

Byte No.	Description	Value (hex)	Meaning
1-13	See Partition Pack (Table 1)	-	Refer to Table 1
14	Partition Kind	02h	MXF Header Partition
15	Partition Status	01h 02h 03h 04h	Open and Incomplete Closed and Incomplete Open and Complete Closed and Complete
16	See Partition Pack (Table 1)	-	Refer to Table 1

The header partition pack status byte 15 shall be set according to the definitions in section 5.2.3.

NOTE – Implementers are advised that future extensions to MXF may introduce new types of partition and that these, if not known by the decoder, should be treated as dark.

6.2.2 Header partition pack value

The value of the **FooterPartition** property shall be filled in correctly if known, otherwise it shall be zero. If the footer partition is not present in the file then this property shall be zero.

The value of the **PreviousPartition** property shall be zero.

Header metadata is required in the header partition. The value of the **HeaderByteCount** property shall be correct and shall not be zero.

An index table is optional in the header partition. The value of the **IndexByteCount** Item shall be zero if there is no index table, or be the correct non-zero value where an index table is present.

An essence container is optional in the header partition. The **BodySID** value shall be zero if there is no essence container in this partition or non-zero if an essence container is present in this partition.

6.3 Body partition pack

Zero or more body partitions may be embedded at intervals throughout the file. Where a body partition is used, the partition shall start with a body partition pack defined as follows:

6.3.1 Body partition pack key

The body partition pack key shall have the following value:

Table 4 – Specification of the body partition pack key

Byte No.	Description	Value (hex)	Meaning
1-13	See Partition Pack (Table 1)	-	Refer to Table 1
14	Partition Kind	03h	MXF Body Partition
15	Partition Status	01h 02h 03h 04h	Open and Incomplete Closed and Incomplete Open and Complete Closed and Complete
16	See Partition Pack (Table 1)	-	Refer to Table 1

The body partition pack status byte 15 shall be set according to the definitions in section 5.2.3.

NOTE – Implementers are advised that future extensions to MXF may introduce new types of partition and that these, if not known by the decoder, should be treated as dark.

6.3.2 Body partition pack value

The value of the **FooterPartition** property shall be filled in correctly if known, otherwise it shall be zero. If the footer partition is not present in the file then this property shall be zero.

The value of the **PreviousPartition** property shall be correctly completed. If it is the first body partition, then the value will be zero as the previous partition will be the header partition.

Header metadata is optional in the body partition. The value of the **HeaderByteCount** Item shall be zero if there is no header metadata or be the correct non-zero value where header metadata is present.

An index table is optional in a body partition. The value of the **IndexByteCount** Item shall be zero if there are no index table segments, or be the correct non-zero value where index table segments are present.

An essence container segment is optional in a body partition. The **BodySID** value shall be zero if there is no essence container segment in this partition or non-zero if an essence container segment is present in this partition.

6.4 Footer partition pack

The footer partition shall be the last partition in an MXF file and shall not contain any essence container data. The footer partition shall start with a footer partition pack.

6.4.1 Footer partition pack key

The footer partition pack key shall have the following value:

Table 5 – Specification of the footer partition pack key

Byte No.	Description	Value (hex)	Meaning
1-13	See Partition Pack (Table 1)	-	Refer to Table 1
14	Partition Kind	04h	MXF Footer Partition
15	Partition Status	02h 04h	Closed and Incomplete Closed and Complete
16	See Partition Pack (Table 1)	-	Refer to Table 1

The footer partition pack status byte 15 shall be set according to the definitions in section 5.2.3. Open footer partitions are not permitted.

NOTE – Implementers are advised that future extensions to MXF may introduce new types of partition and that these, if not known by the decoder, should be treated as dark.

6.4.2 Footer partition pack value

The item **FooterPartition** shall be equal to the value of **ThisPartition**.

The value of the item **BodyOffset** shall be zero.

The **BodySID** item shall be set to zero (as there is no essence container in a footer partition).

The value of the **PreviousPartition** property shall be correctly completed. If there are no body partitions in the file, then the value shall be zero as the previous partition will be the header partition.

Header metadata is optional in the footer partition. The value of the **HeaderByteCount** Item shall be zero if there is no header metadata or be the correct non-zero value where header metadata is present.

An index table is optional in a footer partition. The value of the **IndexByteCount** Item shall be zero if there is no index table, or be the correct non-zero value where an index table is present.

6.4.3 Header metadata repetition in footer partition

The header metadata may optionally be repeated in the file footer, following the rules set out in section 6.6.

6.5 Using the partition pack data for indexing the essence container (informative)

Figure 7 shows an example of the use of the **KAG, BodyOffset, IndexSID** and **BodySID**. In this example, two essence containers A and B are multiplexed into a file which has a total of 17 units aligned to the KAG. The file is divided into six partitions. Stream A consists of six CBE pictures (v0 to v5) and stream B consists of four VBE pictures (u0 to u3). These pictures are mapped into units (A0 to A5 and B0 to B2) with the addition of KLV Fill items to align to the KAG.

Note the different strategies employed. The CBR stream aligns each picture to a gridline, whereas the VBR stream is more storage efficient, but needs to employ an index table to provide random access.

The MXF file is then created by adding the partition pack information, header metadata information and index table information. The gray diagonal lines show how the multiplexing algorithm has used the grid to place the grid units of each stream in the file. Finally in the lower part of the diagram, the absolute grid numbers of the resulting file have been shown.

In figure 7, the **'Essence'** shows the actual essence encoded in the body, both CBR in BodySID(A) (not indexed) and VBR in BodySID(B) (indexed), plus filler to complete the grid alignment of each stream.

'IndexSID' shows the stream identifiers of the (one and only) index table segment and shows the index entries as relative byte offsets within the BodySID(B). The **'BodySID(A)'** and **'BodySID(B)'** tags show the relative addresses within the given essence container. The line **IndexSID(B)** tag shows the index entries as relative byte offsets within the BodySID(B) essence container. The **"BodySID and BodyOffset'** tag show the entries in the partition pack for each partition.

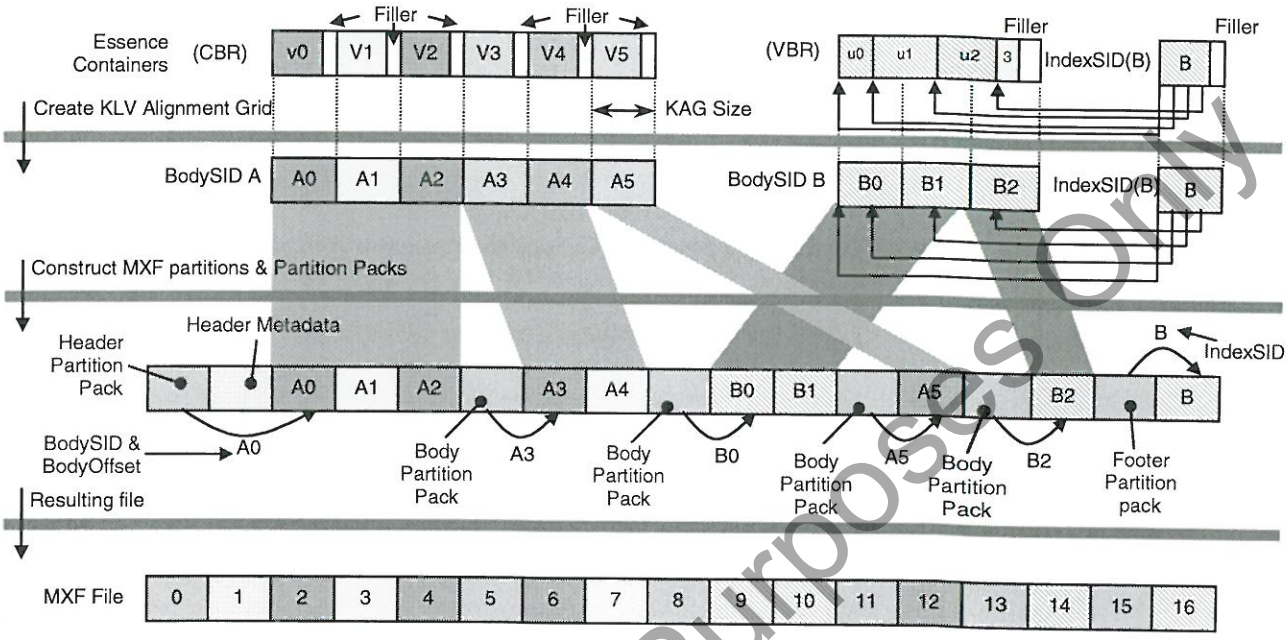

Figure 7 – Example of BodyOffSet, IndexSID and BodySID

6.6 Header metadata repetition in body and footer partitions

Header metadata may be repeated in body partitions and in the footer partition.

Repetition of the header metadata is dependant on the application. Such applications are to be found in the transfer of an MXF file as a stream over a unidirectional link and in tape streamers. The purpose of header metadata repetition in the file is to support the recovery of critical metadata in applications where the file may be interrupted, stored on tape, or where the decoder receives data in mid-transfer.

The KLV coding protocol allows the easy repetition of the header metadata as the entire file is KLV coded (with the exception of the optional run-in). The insertion of the UL identifying a body partition pack is easily detected. The component parts of partitions are shown in figure 3.

If the header partition is open, each repetition of the header metadata in a body partition or the footer partition shall be an updated copy. In the repetition, element durations and other header metadata properties may be changed to reflect conditions at the time of writing and the last closed partition (which may be the footer partition) shall contain the last updated copy. Note that in section 5.2.3, it is stated that there shall be at least one closed partition in a closed MXF file. The closed partition(s) may be Incomplete according to the definition in section 5.2.3.

If the header partition is closed, then each repetition of the header metadata in a closed body partition or the footer partition shall be identical to that in the header partition. In this case, repetition in the last closed partition (which may be the footer partition) is optional. See section 5.2.3 for more information.

Updated repetitions shall be distinguished from the original, and from one another, by updating the generation number item in the identification set together with the reference in the set or sets in which the change(s) was made.

If there have been no changes to the header metadata, the generation number in the copy shall be the same as the generation number in the original.

In all cases, the header metadata with the latest generation number shall be considered as the master version and all other header metadata versions shall be considered outdated.

6.6.1 Guidelines for using header metadata repetition

It is not practical to list all the situations under which the MXF footer may not be available to an MXF application. Guidance is therefore given in this subsection on how an MXF application should behave when using header metadata repetitions. In the following text an application may be an MXF encoder, MXF decoder or some other MXF modification process.

If an MXF file has an open header partition then the footer partition, if present, should contain updated metadata. Therefore, MXF decoders should, if possible, use the header metadata in the footer partition for files with open header partitions. MXF files shall be closed. This means that failure to repeat the header metadata in the footer, when the header is open, results in an open MXF file which is not a valid file. In the event that the footer header metadata is not available or is not easily accessible, MXF decoders may use a repeated header metadata from a body partition or from the header partition. This may occur in the case of partial transfer or preplay (where a file is played or transferred before recording is finished).

When devices are creating MXF files, every effort should be made to ensure that closed and complete partitions are created wherever possible.

In the case of recording on streaming linear media, where header metadata repetition may increase the speed of data recovery, header metadata repetitions may be recorded as part of the file but should be updated to match the most recent header metadata instance when practicable.

When the incoming file is received in complete form, the header metadata repetitions should be updated, where practical, to match the closed partition values or they should be removed (possibly by overwriting with a KLV fill item) from the data stream prior to recording on nonstreaming media. Users and manufacturers should be aware that repetition of header metadata may result in inconsistencies if any values in the closed and complete header metadata are not copied to all other instances of header metadata in the file. MXF decoders should indicate, where practical, by an appropriate means, that the header metadata in use is a copy (from a body partition) and not from a closed or complete, header or footer partition.

6.6.2 Tracking changes with generation UID

When an MXF file is initially created, a single identification set shall be added giving details of the device that created the file. Each subsequent modification to the metadata shall add a new identification set that identifies the device that made the modification. Each metadata set modified shall have its "generation UID" item modified to match the "This Generation UID" value in the identification set relating to the device that modified it. Metadata sets that do not have a "Generation UID" value are unchanged from the initial file creation and are associated with the first identification set in the "Identification" array in the preface set.

7 Operational patterns

7.1 General

MXF defines operational patterns to define constrained levels of file complexity.

It is intended that the operational patterns be written and standardized as separate documents as they are needed. Most operational patterns will be written as a constraint on the axes defined in the next section. These are referred to as generalized operational patterns. However, for certain specialized applications (such as allowing audio-only WAV files to be read by non-MXF devices) there may be specialized operational

patterns which constrain the specification in a different way. Regardless of the operational pattern, any MXF decoder should be able to read the partition pack of the file header and report the contents of the file and why it can or cannot process the file.

It is possible that a file of any operational pattern may be created which has no essence containers. These metadata only files should correctly report the complexity of their timeline with the mechanisms defined below.

7.2 Generic Universal label for all operational patterns

The value of the operational pattern Universal label used to identify any MXF operational pattern shall be defined in the table below.

Table 6 – Value of the MXF operational pattern identification Universal label

Byte No.	Description	Value (hex)	Meaning
1	Object Identifier	06h	
2	Label size	0Eh	
3	Designator	2Bh	ISO, ORG
4	Designator	34h	SMPTE
5	Registry Category Designator	04h	Labels
6	Registry Designator	01h	Labels
7	Structure Designator	01h	Labels
8	Version Number	01h	Registry Version 1
9	Item Designator	0Dh	Organizationally Registered
10	Organization	01h	AAF Association
11	Application	02h	Operational Patterns
12	Structure Version	01h	Version 1
13	Operational Pattern Definition	xxh	Item Complexity
14~16	Definition depends on byte 13	xxh	

7.3 Generalized operational patterns

Generalized operational patterns comprise two components:

1. Operational pattern axes that define the file complexity in two dimensions of item complexity and channel complexity (see below).

2. Operational pattern qualifiers that define file parameters that are common to all operational patterns.

7.3.1 Item complexity

Single item: The file contains only one item. There is a single material package SourceClip which is the same duration as the top-level file package(s).

Playlist items: The file contains several items that are butted one against the other. Each material package SourceClip is the same duration as an entire top-level file package.

Edit items: The file contains several items with one or more edits. Any material package SourceClip may come from any part of any appropriate top-level file package.

7.3.2 Package complexity

Single package: The material package can only access a single top-level source package at a time.

Ganged packages: The material package can access one or more top-level source packages at a time.

Alternate packages: There are two or more alternative material packages, each of which can access one or more top-level file packages at a time. **Informative example:** These different material packages might be used to provide different language versions or special edits destined for different censorship zones.

These axes are summarized in figure 8.

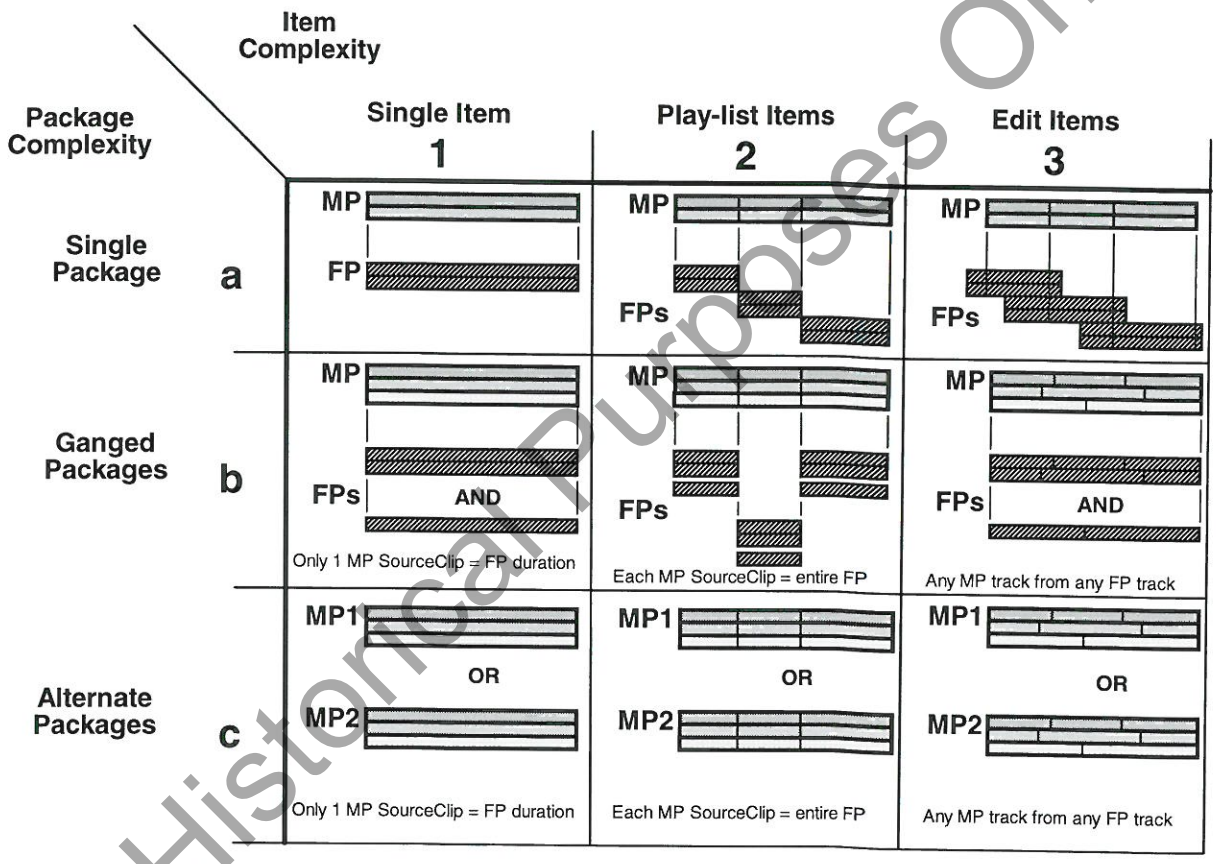

Figure 8 (informative) – Illustration of the operational pattern axes

Note that for generalized operational patterns (OPs), an application shall identify the correct values of the item and package complexity through the operational pattern UL in the partition pack. Applications creating files conforming to generalized operational patterns shall ensure that the simplest operational pattern to which a file conforms is always reported in the operation pattern UL. For example, if a file conforms to OP-1b, it shall not identify the file as OP-2b or OP-1c.

7.3.3 Universal label for generalized operational patterns

For generalized operational patterns, certain axes of flexibility have been defined and the normative value ranges are tabulated here. For specific values of bytes 13 and 14, the associated operational pattern document should be consulted.

Table 7 – Universal label byte ranges for generalized operational patterns

Byte No.	Description	Value (hex)	Meaning
13	Operational Pattern Definition	01h –03h	Item Complexity 01h = Single Item 02h = Play-list Items 03h = Edit items
14	Operational Pattern Definition	01h –03h	Package Complexity 01h = Single Package 02h = Ganged Packages 03h = Alternate Versions
15	Operational Pattern Definition	zzh	Qualifier bits see Table 8
16	Operational Pattern Definition	nnh	Reserved for specification in OP documents

The table below gives the values of the qualifier byte 15. This byte qualifies the operational pattern in a way which is general to all generalized operational patterns.

NOTE – If the essence container in table 8 is the generic container, then the bits described in table 8 refer to (one of) the generic container(s) in the file.

Table 8 – Byte 15 values of the generalized MXF operational pattern UL

Bit number	Values and Descriptions
0	Value = 1 (Marker bit to prevent a zero value)
1	0 = internal essence. (No Essence Container is externally referenced) 1 = external essence. (One or more Essence Containers are externally referenced)
2	0 = stream file. (All Essence Container are multiplexed and/or interleaved to allow streaming of the file) 1 = non-stream file. (No support for streaming of the file).
3	0 = uni-track. (Every Essence Container has one and only one essence track) 1 = multi-track. (One or more Essence Containers have more than one essence track)
7-4	Reserved for future use, encoder should set to zero

7.4 Specialized operational patterns

For specialized operational patterns, it is likely that very detailed and specialized constraints will apply. These will be defined in the associated specialized operational pattern specification.

7.4.1 Universal label byte values for specialized operational patterns

Values for byte 13 of the operational pattern UL in the range "10h" to "7Fh" are reserved to allow specialized operational patterns to be uniquely identified. The meaning of the final 3 bytes of the label will defined in the operational pattern specification.

Table 9 – Byte ranges of the specialized MXF operational pattern UL

Byte No.	Description	Value (hex)	Meaning
13	Operational Pattern Definition	10h –7Fh	Item Complexity: specialized pattern number
14-16	Operational Pattern Definition	xxh	Reserved for specification in OP documents

7.5 Package hierarchy in operational patterns

Where present, the primary package property of the preface set identifies the package which an MXF application treats as the default or primary package.

In certain specialized operational patterns, a material package may reference a source package which may be a file package or a physical package. However, **unless** specifically permitted by the relevant operational pattern document a material package may **only** reference a file package. For clarity, text and figures in this document show the constrained hierarchy. The difference between these package types is detailed in annex C.5.

8 Header metadata

The header metadata is broadly split into two categories: structural and descriptive. Both categories shall be a sequence of KLV coded packets.

This specification defines the structural metadata packages and sets as a single scheme. There shall be no other structural metadata schemes in MXF. The structural metadata scheme shall occupy the first part of the header metadata.

Any MXF descriptive metadata is defined as a "plug-in" in order to accommodate one or more descriptive metadata schemes. Where present, any MXF descriptive metadata scheme shall use the plug-in mechanism provided by the structural metadata scheme. Where there is more than one descriptive metadata scheme in the header metadata, any order of schemes is beyond the scope of this specification.

8.1 Parsing the header metadata

The header metadata shall start with the primer pack. There may be one KLV Fill item that separates the partition pack and the header metadata. No other KLV packet is permitted between the partition pack and the header metadata.

The header metadata may optionally be followed by a single KLV fill item as the last KLV packet. This KLV fill item shall be included in the value of HeaderByteCount as defined in the preceding partition pack. There may be KLV fill items anywhere within the header metadata.

8.2 Primer pack

The primer pack is intended to be a look up table which ensures that all local tags in all public sets are unique within a partition. This is a constraint on the scope rules of SMPTE 336M. Specifically it provides a mapping from all local tags to their respective UIDs for all MXF defined sets; i.e., all those defined in any of the documents in the MXF specification. The primer pack does not cover local sets within essence containers.

The key of the primer pack is given in table 10.

Table 10 – Primer pack key value

Byte No.	Description	Value (hex)	Meaning
1	Object Identifier	06h	
2	Label size	0Eh	
3	Designator	2Bh	ISO, ORG
4	Designator	34h	SMPTE
5	Registry Category Designator	02h	KLV Sets & Packs
6	Registry Designator	05h	Fixed Length Packs (no length fields)
7	Structure Designator	01h	Set / Pack registry
8	Version Number	01h	Registry Version 1
9	Item Designator	0Dh	Organizationally registered
10	Organization	01h	AAF Association
11	Application	02h	MXF File Structure
12	Structure Version	01h	Version 1
13	Structure Kind	01h	MXF File Structure sets & packs
14	Set / Pack Kind	05h	Primer Pack
15	Primer version	01h	Version of the Primer Pack
16	Reserved	00h	

The value of the primer pack is defined in table 11.

Table 11 – Primer pack

	Item Name	Type	Len	UL Designator	Meaning	Default
	Primer Pack	Pack Key	16	Table 10	Identifies a Primer Pack	
↔	Length	BER Length	var (see 8.3)		Overall Length of Primer Pack	
	LocalTagEntry Batch	Local Tag Entry Batch	8+ 18n	06.01.01.07.15	An unordered Batch of Local Tag to UL mappings (see Table 12)	

The LocalTagEntry batch is a batch of local tag to UID mappings with the format below. The pairs are in no particular order. There shall be only one entry for each local tag in the LocalTagEntry Batch. There shall be only one dynamic entry for each UID in the LocalTagEntry batch (see 8.2.2). The scope of the pack is one partition. The packs do not accumulate with every new partition, they are replaced by the pack in the new partition.

Table 12 – LocalTagEntry batch

#	Item Name	Type	Len	UL Designator	Meaning	Default
	Number of items	Uint32	4	N/A	The Number of Items in the Batch	N
	Item Length	Uint32	4	N/A	The Length of each Item	18
N	Local Tag	Uint16	2	01.03.06.02	The value of the Local Tag. [RP 210: A locally unique registry identifier]	
	UID	UL or UUID	16	01.03.06.03	The UID of which the local tag is an alias	

8.2.1 Contents of the primer

The primer shall contain all local tags used in MXF sets within the partition in which it occurs. The primer may also contain private local tags used in sets defined outside the scope of the MXF defined sets. The primer may also contain local tags which are not used within the partition

8.2.2 Local tag values

Local tags shall be allocated in the following ranges:

00.00h	Shall not be used
00.01h to 00.FFh	Reserved for compatibility with AAF
01.00h to 7F.FFh	Statically assigned Tags – assigned by MXF specifications
80.00h to FF.FFh	Dynamically allocated tags.

The statically allocated tags are part of a public specification and the mapping between a static 2-byte local tag and the UID is permanently defined. This allows implementers to have a fast look up strategy for all MXF defined tags.

The dynamically allocated tags are allocated on a partition by partition basis. This means that when a file is re-written, the mapping between a dynamic 2-byte local tag and a UID may change.

The algorithm for allocating the dynamic tags is not specified. The algorithm shall ensure that the restrictions to the table written in this specification are met.

8.2.3 Dark metadata support

Metadata values inserted into a file which are intended to be preserved, but are unknown to some decoders are known as dark metadata (see section 5.3.2). Metadata extensions to MXF sets shall place their tags in the primer pack in order to prevent tag clashes. An MXF decoder should not attempt to interpret local tag values within sets whose set key is not understood by that decoder.

Note that any implementer inserting metadata in MXF which does not conform to the MXF constraints upon SMPTE 336M, must recognize that this may jeopardize future public specification within MXF of that metadata.

Metadata sets outside the scope of this specification may have local set coding. The existence of a local tag in a set with an unknown set key does not imply that the local tag in the primer enables the translation of this local tag into a UID.

The processing of dark metadata by MXF decoders is an application issue. MXF Decoders are not required to handle dark metadata, but they should preserve any dark metadata in the file in case later processes require it.

8.3 Header metadata coding

The header metadata is a sequence of KLV coded metadata sets which describe the contents of the file body.

Decoders shall comply with section 5.3.4. It is preferred that the length field of each metadata set shall be BER long-form encoded using 4 bytes. However, header metadata decoders shall not assume that this is the only value used.

An example of KLV packets is shown in figure 9.

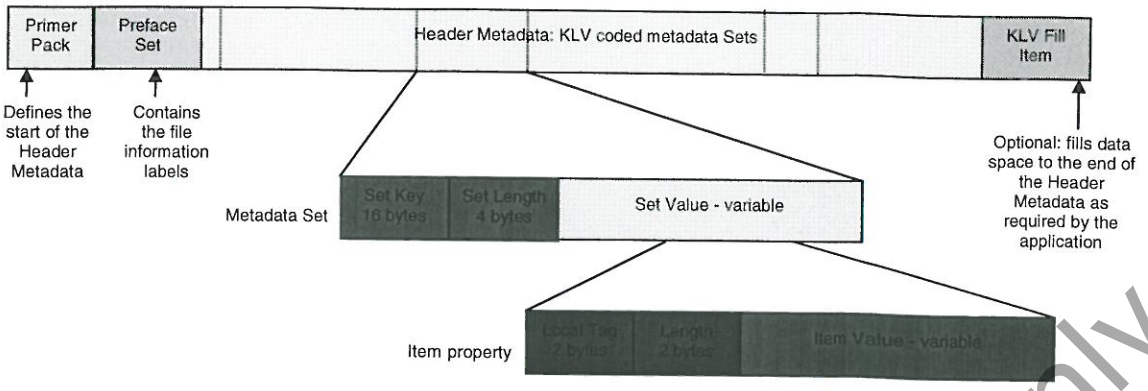

Figure 9 – KLV data coding in the header metadata

8.3.1 Data model

The header metadata contains metadata sets which describe the contents of the file body as a whole. The metadata is broadly split into two parts: structural metadata and descriptive metadata. The structural metadata is defined in this section and defines the capabilities of the file and how it is constructed. The definitions of the operational patterns are given in section 7 and these define the structural metadata items that needs to be included for a complete definition of the file body. The descriptive metadata is used to define various editorial aspects of the file, for example production and clip information. This is defined in the descriptive metadata specification plug-ins in section 8.7.

8.3.2 Strong and weak references

References are merely a way of allowing one item to refer to another. This may be a one to one relationship implying ownership (strong reference) or a many to one relationship (weak reference). Strong references shall be made to the Instance UID of another set and weak references shall be made either to the instance UID of another set or to a UL. This is known as the referencing method. The alternate method of embedding each strongly referred data set into the referring data set (as allowed by the recursive grouping mechanism in SMPTE 336M) is not used in MXF defined header metadata. It use is discouraged, but not prohibited in dark metadata.

Figure 10 illustrates a sequence of KLV packets, each with a unique identification (UID) and the connections between packets.

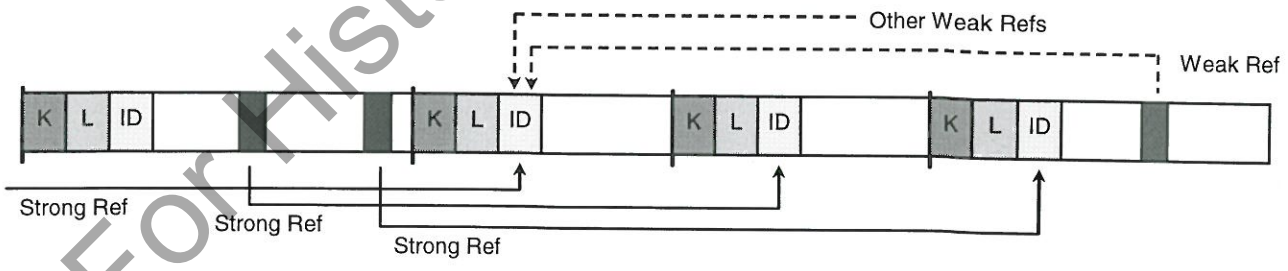

Figure 10 – Using UIDs to connect metadata sets in a data stream

A guide to the use and implementation of strong and weak references is given in the MXF engineering guideline.

The logical model that results from the use of references and other syntactic constructs is defined in section 8.5 of this specification. Logical models for descriptive metadata schemes must follow the same broad guidelines as used in this specification.

Uniqueness of UIDs —

The UIDs used in the MXF header metadata are globally unique values. However, the scope of their application shall be limited to any one instance of the header metadata. Thus, simple repetition of the header metadata does not require new UIDs to be created. Likewise, no processing is required if an MXF file is copied. Updates to any header metadata set can be detected using the Generation UID Item within that set.

NOTE – A fuller description on the difference between UUIDs and ULs is given in the MXF engineering guideline.

8.4 Structural metadata semantics

The header metadata is composed of a sequence of metadata sets which shall be positioned adjacent to each other. These sets are connected to each other using strong or weak references (section 8.3.2) in order to provide a logical data model. Figure 11 and figure 13 broadly illustrate the semantics of the structural metadata and how it relates to the content of the file body.

Figure 11 shows the main synchronization and referencing features of an MXF file. The material package (at the top) is a metadata structure which generally represents the "output" timeline of the file. If the MXF file were played then this timeline represents the output that would be seen and heard.

- The material package owns a track which defines the start and edit unit rate of the output.
- The track owns a sequence which defines the duration of the output.
- The sequence, in turn, may be divided into one or more SourceClips.

The material package SourceClip is linked to a track within a top-level source package. In generalized operational patterns, the material package shall specifically reference a top-level file package. The top-level source package contains a file descriptor variant of an essence descriptor which identifies the essence container and may also contain a link to another source package. A lower-level source package contains information about how a file was created and possibly contains a physical descriptor which may describe the video tape or other physical medium from where the material came. This is historical annotation and may give rise to lower-level source packages referencing other source packages to any depth as long as there is no recursion. See annex C.5 for more details.

An **informative example** of the use of this mechanism could be in the creation of a promotional program by a broadcaster. The material package defines the output. The top-level file packages contain material from several television programs which are being promoted. The first lower-level source packages in turn describe the MXF files used to make the television programs. The next lower-level of source package may define the tapes used to capture the original material.

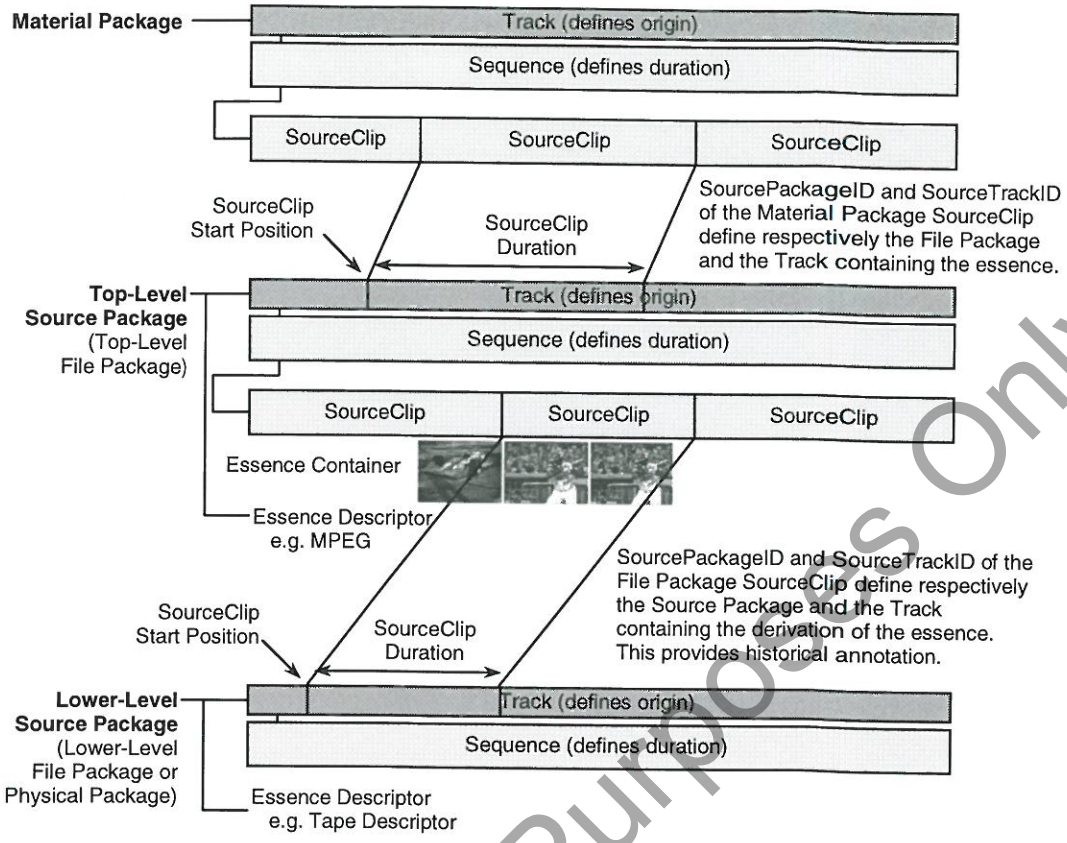

Figure 11 – Header metadata packages

Figure 11 illustrates the relationship between the track and the actual essence in its container. In the center of the figure, it shows the essence data which would be KLV encapsulated in MXF using one or more of the essence container specifications. In the upper part of the figure is a representation of the structural metadata which generally describes the output timeline of the file through the material package. The material package governs the synchronization and play order of the SourceClips defined in the top-level source package. This is achieved using a number of parallel metadata tracks each comprising a sequence of one or more SourceClips. Each SourceClip in a material package track can describe the location of the essence item in the top-level source package. The top-level source package in turn identifies the actual essence container through the essence container data set using the BodySID value to link to the appropriate file partition(s) [see figure 18].

Each SourceClip in a material package has a SourcePackageID value and a SourceTrackID value that identify respectively the top-level source package and the track which is to be accessed. The SourceClip also has a start position value and a duration value that access the portion of the track required. The start position value is the number of material package track edit units along the top-level source package track with the first edit unit number on the track set by the top-level source package track origin value. Clearly, the SourceClip being extracted from the top-level source package must fall within the total duration of the referenced track. This reference chain shall end when a zero valued SourcePackageID is encountered.

The same mechanism is used within a source package to access tracks in other lower-level source packages. However, whereas a top-level source package represents essence data in the file body, lower-level source packages are used to annotate the derivation of that essence. Annex C defines the two sorts of source package available in MXF: a file package and a physical package.

Metadata tracks such as those used for time code and descriptive metadata refer to the package which contains them. Thus the time code track in a material package defines the time code for the file playout and will therefore be continuous. A time code track in a file package represents the time code in the essence

container associated with that file package. Since the time code in an essence container may be discontinuous, the file package may require one or more time code SourceClips to represent the desired time code values in the essence container.

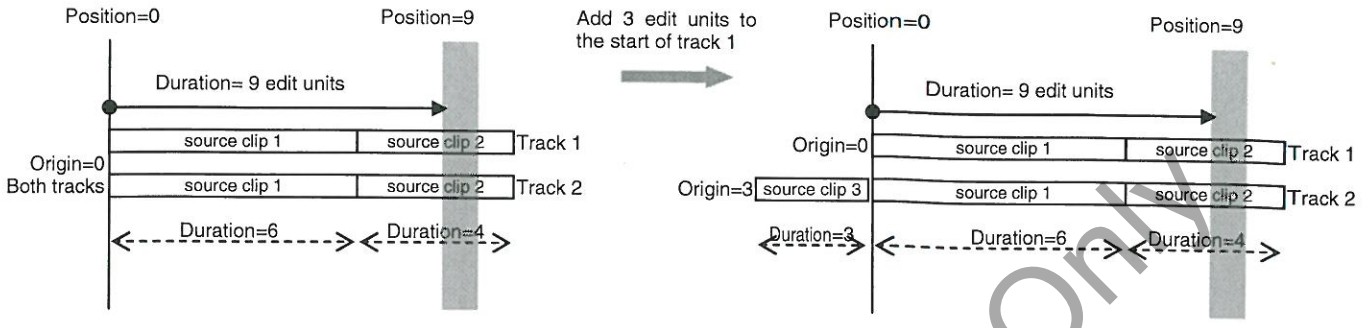

Figure 12 – MXF timing model example where all tracks have the same edit rate

Synchronization between tracks is achieved by using an idealized timing model within any package.

Figure 12 shows the relationship between the position and origin parameters. In most tracks, essence containers will start at Position=0, with an origin of 0. If the edit units of each track are the same then any essence content on the tracks with the same value of position will be synchronized.

If material is added to the **start** of one of the tracks then an offset is required in order to maintain synchronization. In the example above, three edit units are added to the beginning of track 2. To maintain synchronization, the origin of the track now starts at the third edit unit of the track. In this example, the edit units in both tracks are identical. In general, the concept of idealized time can be introduced to determine synchronization.

Time offset from start of essence T_o on track n is given by $T_{o,n} = \frac{Position_n + Origin_n}{EditRate_n}$

Position=0 within track n ($P_{0,n}$) equates to the numerical $Origin_n$ position of the essence in that track. Samples which have the same normalized value of Position on any track are synchronized and shall be played at the same time in MXF.

Essence on tracks n and m are synchronized when $\frac{Position_n}{EditRate_n} = \frac{Position_m}{EditRate_m}$

The normalized position on all tracks shall be aligned even when they have different values of "edit rate" and origin. Time code is represented by a track and can be used to annotate the value of position at any point in the of the file. When using time code to synchronize different streams, it is important to realize that in MXF, time code is a metadata annotation and a time code value gets converted to an edit unit count value (and hence a position value) which is the underlying synchronization mechanism.

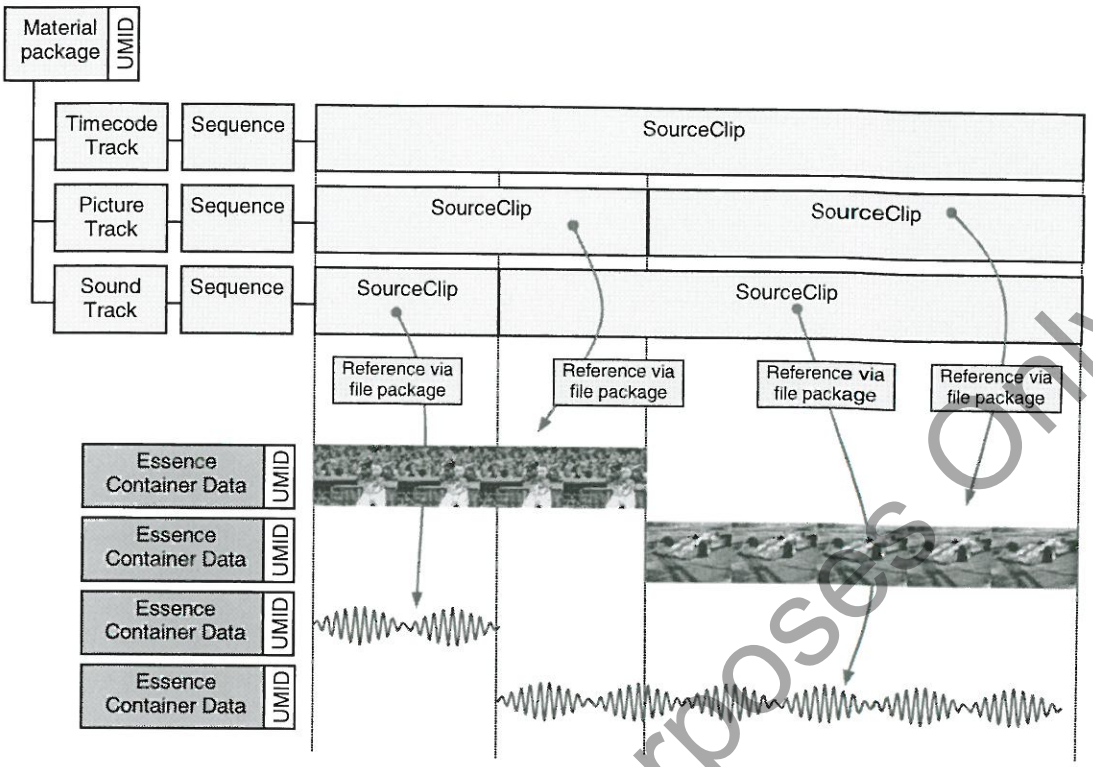

Figure 13 – Logical header metadata structure and its relation to essence data

8.4.1 Relationship between file packages and essence containers

Each file package is related to one essence container through an EssenceContainerData set as illustrated in figure 13 and in more detail in figure 18. There is one EssenceContainerData set for each file package and they are linked through a common package UID value. Each EssenceContainerData set has a BodySID value that identifies any partition(s) in this file that contain the essence container. Each EssenceContainerData set may also have an IndexSID value that identifies the partition(s) that contain the index table used to index the contents of that essence container.

8.5 Structural metadata definition

This section specifies the MXF class structure so that it can be used to define all operational patterns. The normative annexes of this document define the structural metadata sets and their property definitions.

NOTE – The class structure of the MXF structural metadata is based on a simplified instance of the AAF metadata class model.

8.5.1 Header metadata start

The header metadata sets shall start with a preface set that includes a value of the operational pattern UL, a batch of zero or more descriptive metadata scheme ULs and a batch containing zero or more essence container ULs. These properties provide an early indication of the complexity of the file structure and its essence containers.

8.5.2 Generic class diagram

The structural metadata class model is represented below in diagrammatic form. Each class shall be implemented as KLV coded metadata set.

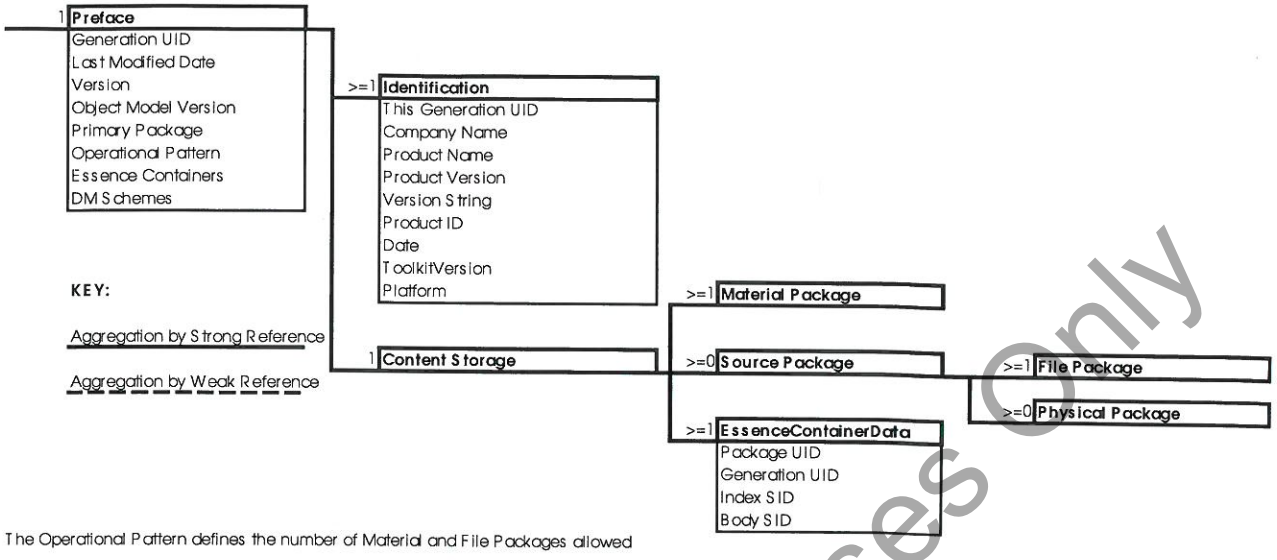

Figure 14 – Root metadata sets for structural header metadata

Figure 14 shows the root metadata set structure for an MXF file. An MXF parser would find a partition pack within the file and then look to see if there was header metadata in that partition. It would then look for the preface set which defines the version and contents of the structural metadata. The preface set references one or more identification sets which contain human readable information about the tool(s) used to create or modify the MXF file. A new instance of this identification set shall be created every time the file is modified. The preface set also references the content storage set which identifies all the packages (material, file or physical) and the EssenceContainerData sets in the structural metadata. The relationship between these packages is given in section 8.4 and shown diagrammatically in figure 11. In order to define these packages (material, file and physical), a generic package class has been defined. Subclasses of this generic package are then defined.

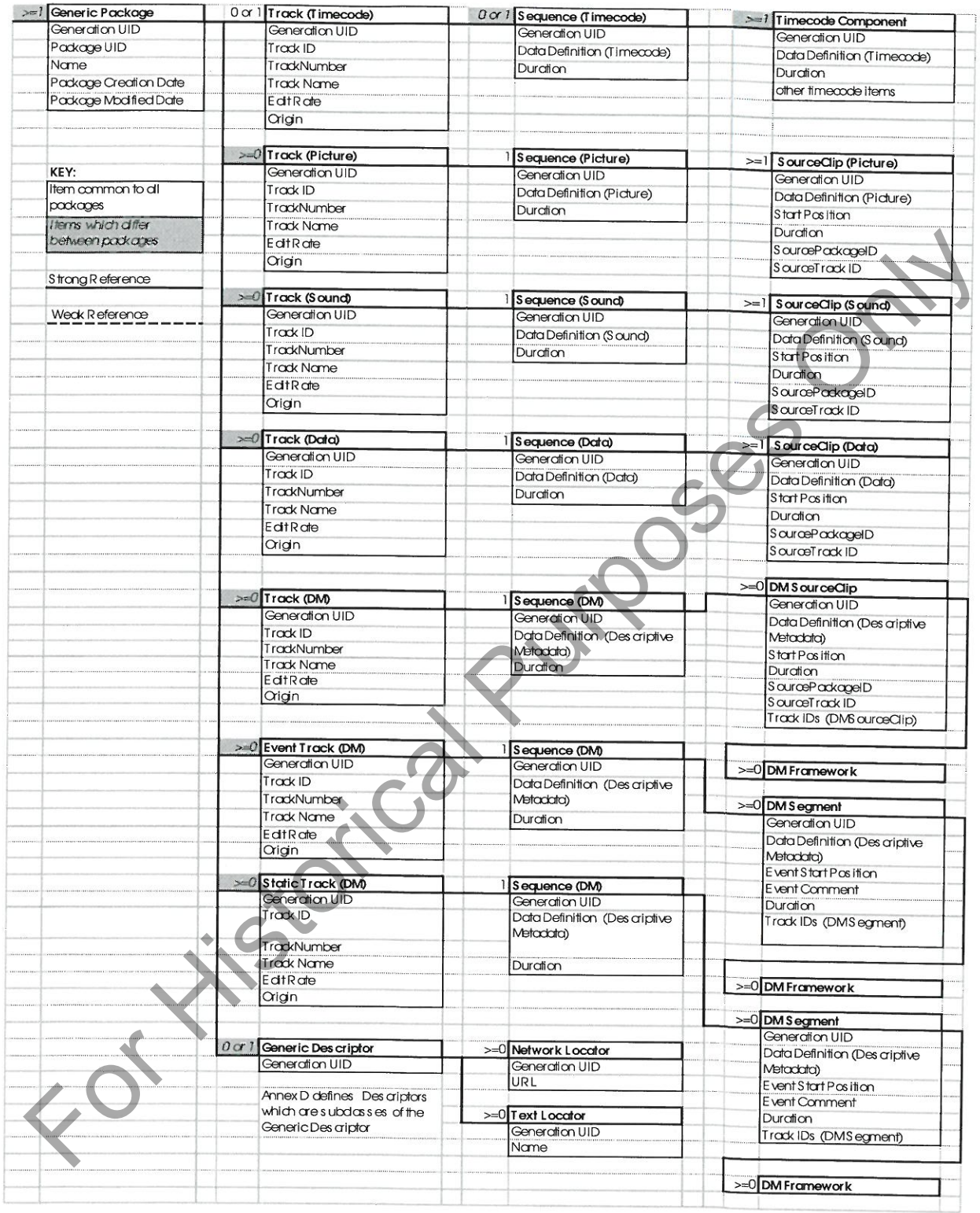

Figure 15 – Diagram of a generic package in the structural header metadata

8.5.3 Material package

The material package shall be an instance of the generic package with the following semantics:

1. The number of material packages in a file shall be defined by an operational pattern specification.
2. The tracks of a material package generally represent the "output timeline" of the file (figure 11).
3. The number of picture tracks, sound tracks and data tracks shall be controlled by the operational pattern specification and by the top-level source package(s) which are associated with the material package.
4. The SourceClips of any track in a material package shall link to a track in a top-level source package.
5. There shall be only one time code SourceClip in a material package (i.e., continuous output time code).
6. There shall be no descriptor sets in the material package.
7. There shall be no EssenceContainerData linked to the material package.
8. Any descriptive metadata referenced by the material package generally relates to the "output timeline".

8.5.4 Source package

The source package is an instance of the generic package which has a strong reference to a descriptor to describe essence container. Source packages are known as file packages or physical packages depending on the application. Further details are given in annex C.4.

8.5.5 Top-level source packages

These source packages have the following semantics:

1. The number of these top-level source packages and the descriptors they contain shall be controlled by the operational pattern specification.
2. The tracks of these top-level source packages shall represent the "input timeline" of the file (figure 11).
3. The number of picture tracks, sound tracks and data tracks shall be controlled by the operational pattern specification and by the essence container which is associated with these top-level source packages.
4. The SourceClips of any track in these top-level source packages may associate with a track in another source package.
5. There shall be zero or more time code components in a top-level source package.
6. There shall be zero or one file descriptor set in a file package. (This may be a multiple descriptor linking to zero or more file descriptors.)
7. Each file descriptor shall define a different essence element in the essence container.
8. The top-level source package may optionally use essence locator sets to locate external essence.
9. Any descriptive metadata referenced by the top-level source package relates to the content of the essence container associated with this source package.

8.5.6 Lower-level source packages

These source packages have the following semantics:

1. The tracks of these lower-level source packages document the derivation or history of SourceClips of the source essence (figure 11).
2. The number of lower-level source packages in a file is controlled by the application writing the file and is not constrained by any operational pattern.
3. There may be several time code components in a lower-level source package
4. The number of picture tracks, sound tracks and data tracks is not bounded as this is historical annotation only.
5. Any descriptive metadata in the lower-level source package relates to the content of the source essence.
6. If a file package has a descriptor, it shall be a file descriptor. (This may be a multiple descriptor linking to zero or more file descriptors.)
7. A physical package shall have a physical descriptor.

8.5.7 Relationship between the packages and SourcePackageID / SourceTrackID

The content storage set contains a batch of strong references to every package in the file. At least one of these will be a material package. This package will contain a number of tracks, each of which has a sequence, which in turn will have a number of SourceClips. File package tracks have a track number whose value is defined by the essence container specification and a Track ID to which a SourceClip can refer.

The SourceClips contain:

- a SourcePackageID which identifies the package with which this SourceClip is associated;
- a SourceTrackID which identifies the track within the package with which this SourceClip is associated;
- a start position which identifies the start of the essence element in the package track (measured in edit units defined in the referencing package track);
- a duration which identifies the length of the essence element in the package track (measured in edit units defined in the referencing package track).

The same mechanism shall be used to relate each SourceClip in a material package to a source package, and each SourceClip in a source package to another source package. These relationships are shown diagrammatically for the material package and file package in figure 16 and figure 17.

Terminating the package referencing chain

The chain of source package references shall be terminated by the following method:

- Every SourceClip in the last package shall set the sourcePackageID to zero.

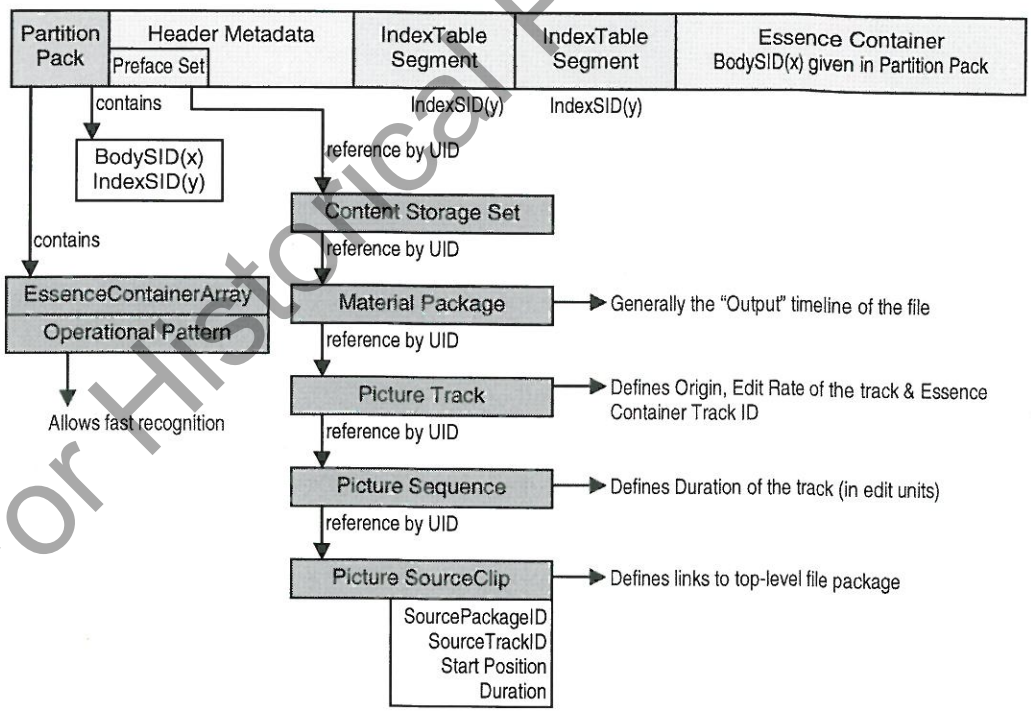

Figure 16 – Relationship between material package references

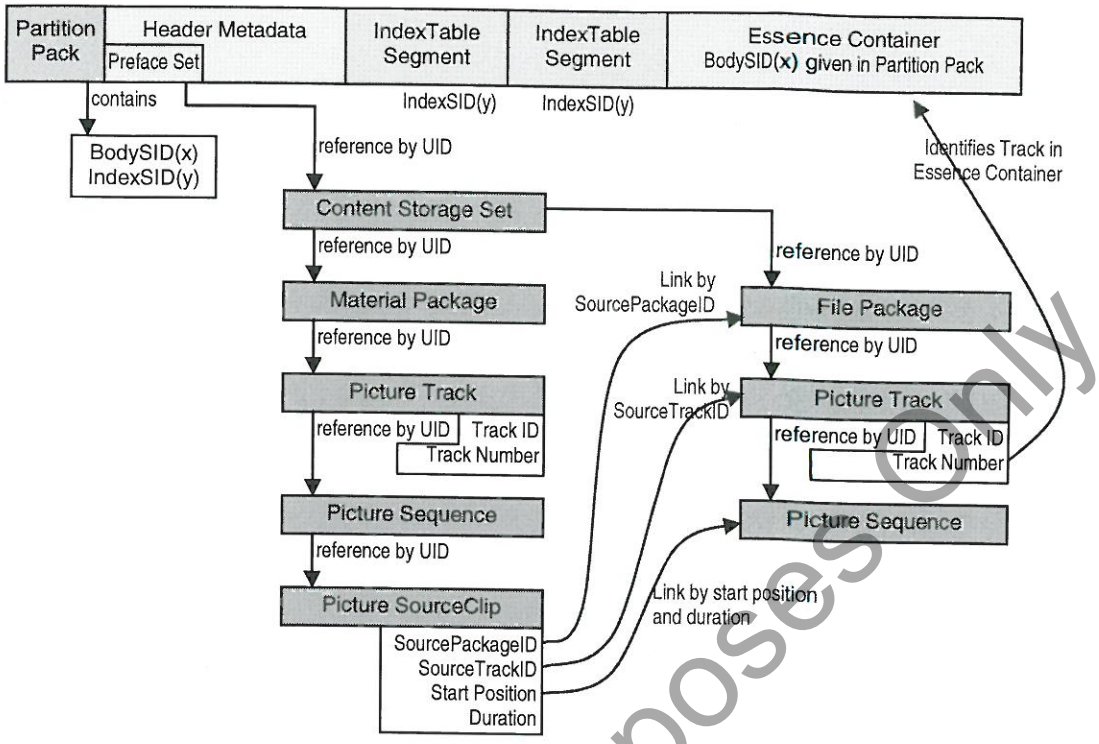

Figure 17 – Relationship between material package SourcePackageID / SourceTrackID and file package

8.5.8 Relationship between the BodySID and IndexSID

The content storage set contains a batch of UIDs which reference every package in the file. The content storage set also contains a batch of essence container data sets, each of which defines the relationship between BodySID and IndexSID for each of the packages. The UMID of the file package is used to link it to one of the EssenceContainerData sets which in turn defines the BodySID and IndexSID values which must be used for that package. This is shown diagrammatically in figure 18.

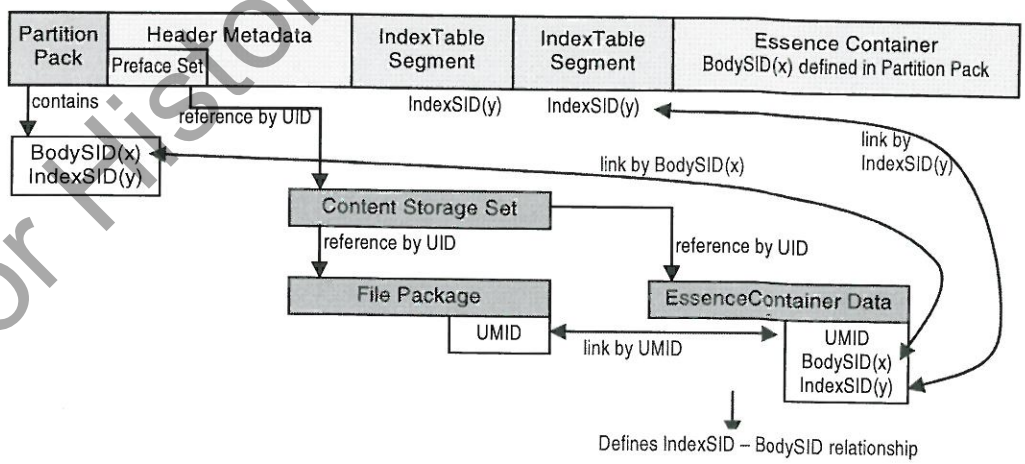

Figure 18 – Relationship between BodySID and IndexSID

8.5.9 Scope of the track ID values

The scope of all track ID values is limited to the package in which the track occurs. Therefore, a material package and a source package may use the same track ID value for the picture track, but each track ID is valid only in the appropriate package.

8.6 Structural header metadata Implementation

All structural header metadata objects shall be implemented as KLV coded local sets according to SMPTE 336M. All object properties in KLV coded local sets are encoded according to section 5.3.5.

The common key structure for all the structural header metadata objects coded as local sets in this document, shall be defined as follows:

Table 13 – Common key value for the all structural metadata sets

Byte No.	Description	Value (hex)	Meaning
1	Object Identifier	06h	
2	Label size	0Eh	
3	Designator	2Bh	ISO, ORG
4	Designator	34h	SMPTE
5	Registry Category Designator	02h	Sets & packs
6	Registry Designator:	53h	Local Sets: 2-byte local tags, 2 byte lengths
7	Structure Designator	01h	Set/Pack Dictionary
8	Version Number	01h	Registry Version 1
9	Item Designator	0Dh	Organizationally Registered
10	Organization	01h	AAF Association
11	Application	01h	MXF / AAF Association Structural Metadata Sets
12	Structure version	01h	Structure Version 1
13	Structure Kind	01h	MXF / AAF Association compatible sets & packs
14	Set Kind (1)	xxh	MXF Set Definition (see Table 14)
15	Set Kind (2)	yyh	MXF Set Definition (see Table 14)
16	Reserved	00h	Reserved

The definition of bytes 14 and 15 of the keys for all objects coded as local sets is given in table 14.

Table 14 – Key values for structural metadata sets

Set Name	Byte 14	Byte 15
Preface	01h	2Fh
Identification	01h	30h
Content Storage	01h	18h
Essence Container Data	01h	23h
Material Package	01h	36h
Source Package (File / Physical)	01h	37h
Track (Timeline)	01h	3Bh
Event Track	01h	39h
Static Track	01h	3Ah
Sequence (all cases)	01h	0Fh
SourceClip (Picture, Sound, Data)	01h	11h
Timecode Component	01h	14h
DM Segment	01h	41h
DM SourceClip	01h	45h
File Descriptor	01h	25h
Generic Picture Essence Descriptor	01h	27h
CDCI Essence Descriptor	01h	28h
RGBA Essence Descriptor	01h	29h
Generic Sound Essence Descriptor	01h	42h
Generic Data Essence Descriptor	01h	43h
Multiple Descriptor	01h	44h
Network Locator	01h	32h
Text Locator	01h	33h

8.7 Descriptive metadata plug-ins

8.7.1 General

This section defines the normative descriptive metadata plug-in mechanism and generic label used to identify the descriptive metadata sets.

NOTE – The MXF descriptive metadata plug-in is very simple. Guidelines on its use can be found in the MXF engineering guideline SMPTE EG 41. General guidelines for adding descriptive metadata can be found in the engineering guideline for MXF descriptive metadata, SMPTE EG 42.

8.7.2 Generic Universal label for the MXF descriptive metadata sets

In MXF, descriptive metadata schemes are identified by Universal labels in the value field of the DM schemes batch property. Table 15 shows a Universal label structure which has been pre-allocated for this purpose. All schemes which are called "MXF defined schemes" shall be identified by using Universal labels with this structure. There may be other schemes which use other label structures.

NOTE – Implementers are advised to inspect the appropriate SMPTE registries to determine the precise meaning of the values in the DM schemes batch property.

Table 15 – Generic Universal label for MXF descriptive metadata schemes

Byte No.	Description	Value (hex)	Meaning
1	Object Identifier	06h	
2	Label size	0Eh	
3	Designator	2Bh	ISO, ORG
4	Designator	34h	SMPTE
5	Registry Category Designator	04h	Labels
6	Registry Designator:	01h	Labels
7	Structure Designator	01h	Labels
8	Version Number	01h	Registry Version
9	Item Designator	0Dh	Organizationally Registered
10	Organization	01h	AAF Association
11	Application	04h	MXF / AAF compatible Descriptive Metadata Labels
12	Label Version	01h	Version 1 of the MXF / AAF DM labels
13	Scheme Kind	xxh	Defined by the scheme specification
14~16	Reserved	00h	Reserved for use by each scheme

8.7.3 Generic MXF descriptive metadata keys

The generic key below identifies MXF descriptive metadata sets and MXF descriptive metadata frameworks. Individual key values are defined in the associated MXF descriptive metadata specification.

Table 16 – Generic key for MXF descriptive metadata schemes

Byte No.	Description	Value (hex)	Meaning
1	Object Identifier	06h	
2	Label size	0Eh	
3	Designator	2Bh	ISO, ORG
4	Designator	34h	SMPTE
5	Registry Category Designator	02h	Sets & packs
6	Registry Designator:	53h (default)	Local Sets: 2-byte local tags, 2 byte lengths see section 5.3.5 and appropriate DM Specification
7	Structure Designator	01h	Set/Pack Dictionary
8	Version Number	01h	Registry Version 1
9	Item Designator	0Dh	Organizationally Registered
10	Organization	01h	AAF Association
11	Application	04h	MXF / AAF Descriptive Metadata sets
12	Structure version	01h	Version 1
13	Structure / Scheme Kind	xxh	See scheme for definition
14~16	Reserved	00h	Reserved for use by each scheme

8.7.4 Plug-in mechanism

An MXF file may contain zero or more descriptive metadata (DM) schemes. These schemes may or may not be MXF DM schemes.

A descriptive metadata scheme comprises one or more descriptive metadata frameworks in which each DM framework has several metadata items and sets grouped together (generally for semantic reasons).

A DM scheme may have several DM frameworks. A DM framework comprises a number of metadata items and sets, each set containing further metadata items (or more sets). These items define the metadata values of the DM framework.

In order to synchronize the metadata to the essence, each DM framework shall have a track in a Material package, a file package or a source package (see annex B.17).

NOTE – If the track is in a material package then the metadata refers to the essence content as it is to be presented (the generally the output timeline). If the track is in a top-level file package then the metadata refers to the essence content of each essence container associated with that file package (the input streams). If the track is in the lower-level source package, then the metadata refers to the content associated with that source package.

Unless specified in a descriptive metadata document, all DM frameworks can be contained by any package.

Some DM frameworks will relate to an entire package. In this case the duration of the metadata track shall be set to be the duration of the entire package.

NOTE – Descriptive metadata tracks may be event tracks. This means that DM segments may overlap. It also means that segments with zero duration are possible in this case. This allows annotation of instantaneous events occurring within the package.

9 File body

Immediately following the file header is the file body, consisting of one or more essence containers. Each essence container contains KLV encoded essence data (which may also include system data and embedded metadata). Each partition contains a single segment of essence container (see below) which, through its essence container specification, defines the KLV encoding. The principle of KLV encapsulation of a typical essence container is shown in figure 19.

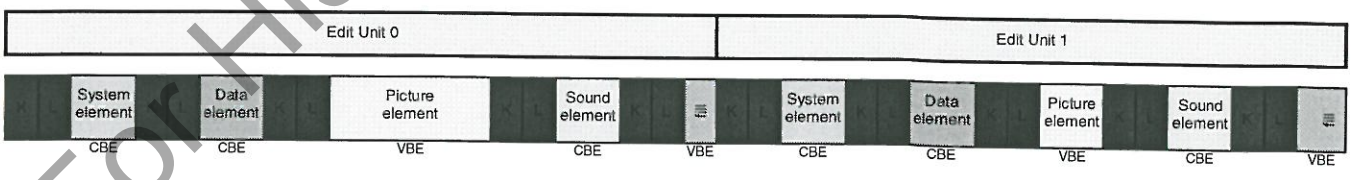

Figure 19 – Typical KLV coding in an essence container wrapped every edit unit

9.1 Essence containers

An essence container specification shall meet the requirements given below. An essence container specification may also define how index tables shall be used and how the essence descriptor shall be coded.

The individual essence container specifications are defined in other documents not included here. All compliant essence container specifications shall meet all the normative criteria listed below. These criteria are

listed under two broad headings grouping the technical requirements and the standardization requirements. An essence container specification may be split over multiple documents when generic structures and specific mappings of different essence types are required into those structures.

9.2 Technical requirements

An essence container specification 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 336M 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 336M with a limit to the coding range provided to limit the requirements of compliant equipment for this essence container.
- The specification shall include the byte order for the correct interpretation of multi-byte values.
- The essence containers used in streaming operational patterns must be capable of interleave over a defined interleaving period or must be capable of being multiplexed in an MXF file using the partition mechanism. The interleave / multiplex duration is dependent upon the application, but 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). Special timing arrangements may be needed in the case of, for example, long GOP coding using B-frame coding. This requirement allows simple editing and switching of the interleaved essence.
- The essence container specification shall be assigned a registered Universal label value, which shall be used by the partition pack, the descriptor set and the preface set to identify the 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 for that essence container type.
- The essence container specification may define the use of descriptor(s) for that essence container type including any mappings required from multitrack audio in the header metadata to multichannel audio in the essence container.
- The essence container specification shall define the mechanism to identify track numbers for the purpose of identifying specific content within the essence container.
- The essence container specification may define an optimal KAG value in compliance with section 5.4.1.

9.3 Standards requirements of an MXF essence container document

An MXF essence container specification should be a public standard from an internationally accredited standards body. This is to ensure that all essence container documents can be used for interchange. New essence types should use the MXF generic container. Doing so will minimize the implementation burden for equipment makers and increase the chances of interoperability.

9.4 General information (informative)

The size of each element (in bytes) may be determined by the length value of the KLV packet according to SMPTE 336M in conjunction with the defining essence container specification.

The essence container specification may comprise more than one document. This is the case with the generic container which requires associated essence and metadata mapping documents together with application documents to provide a complete specification.

The interleaving of elements of different essence types is fully described in each essence container document.

The description of each essence type is defined in the appropriate essence container specification.

9.5 Descriptors

The header metadata describes the essence and its container(s). Packages are concerned with temporal characteristics whereas descriptors are concerned with parametric properties. These properties may be to do with sampling, such as sampled screen size. They may be to do with the organization of the data such as the number of audio channels. They may also give information such as where external essence may be located.

All descriptors are derived from the generic descriptor set which is shown in table 17. This generic set defines the base functionality of all descriptor sets. It is never used directly and the UL designator and set keys are defined by the individually derived descriptor sets.

Table 17 – Generic descriptor set

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Generic Descriptor	Set Key	16		Defined by descriptor type	Req	Defines the Generic Descriptor set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [RP 210 The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [RP 210 Specifies the reference to an overall modification]	
Locators	StrongRefArray (Locators)	8+ 16n	2F.01	06.01.01.04.06.03	Opt	Ordered array of strong references to Locator sets If present, essence may be located external to the file. If there is more than one locator set an MXF Decoder shall use them in the order specified. [RP 210 Specifies a vector of references to essence locators]	

10 Index table

10.1 Overview

Index tables can be used to speed up the location of individual editable units of essence in a timeline. The essence itself may be interleaved, as in the generic container, or may be a single essence element. An index table may be placed in the header partition, in the footer partition or optionally distributed throughout body partitions of an MXF file.

Index tables should be implemented wherever possible. They can be used to satisfy a number of the user requirements, particularly those to do with the handling of partial files (for example reading part of a large file from the middle of a data tape, indexed by the timeline). When index tables are used, they shall conform to this specification.

An index table can be created "on the fly" during file creation from an input signal and is notionally placed in the file footer on recording. In practice, its placement in a server file system may be anywhere for storage convenience. During transfer of a complete file, an application may choose to replace a distributed index table with one placed immediately after the header. It should be noted, however, that streaming devices may have limited index table memories and a single large index table may be less useful, in this circumstance, than a distributed one.

Index values and offsets do not change if the index table is relocated within the file – this is handled by the low-level byte-stream format described in section 6, in which the index table and essence container are treated as separate streams linked together by properties in the partition packs. All index table entries use offsets relative to the start of the essence container stream, and not absolute offsets within the file.

An index table shall be used to index a single essence container. Each index table shall be indexed by edit units. The edit unit rate of an index table is defined by the tracks for the essence container that it indexes. An edit unit is usually a field for video, but could equally well be a frame (comprising 2 interleaved fields, or perhaps a single progressive frame). An edit unit for an audio-only application could be an AES3 audio frame comprising 192 audio samples (having a duration of 4 msec when using a 48-kHz sampling rate).

The index tables are created in such a way that they can provide indexing support for picture, sound, field coded, frame coded, interleaved and variable bit rate essence. In the section below, the word "picture" will be used to refer to an essence image which has been field or frame coded and may be interleaved. This interleave period could be a field or a frame depending on the application. A frame is composed of two interlaced fields.

The index tables are also able to cope with temporally re-ordered content such as Long GOP MPEG. Each of the essence container documents contains the implementation details of index tables for a particular essence container. The examples below are provided to aid the designer build a generic index table handler for applications ranging from non-interleaved simple essence to Interleaved and temporally reordered essence.

NOTE – Application writers should be aware that absence of index tables may lead to performance degradation, especially when MXF files are transferred to and from certain classes of machine. Application performance may also be degraded by the requirements of index table implementation. Their use is therefore optional.

10.1.1 Interleaved streams

Interleaved streams consist of a regular sequence of several essence elements as a group, usually interleaved every picture frame. Examples of essence elements are video, audio, system and auxiliary data. For convenience, in this section we refer to any streams of element groupings as interleaved streams in an essence container.

10.1.2 Constant bytes per element (CBE) and variable bytes per element (VBE)

Within an interleaved essence container, each element may be either a constant bytes per element (CBE) or a variable bytes per element (VBE). Whenever there is at least one VBE in an element grouping, a separate index table value is required to locate each element grouping in the interleaved stream. The purpose of an index is to locate an element grouping based on a temporal reference, such as a timeline position value.

Note that for most purposes, an uncompressed audio element has the constant bytes per element property even though some video field rates lead to a small variations (e.g., at a 59.94-Hz field or frame rate, the number of audio samples varies between 800 and 801 in a regular sequence).

Figure 20 shows the first three edit units in an interleaved essence container (which was KLV coded in figure 19). In this example, each interleaved essence container consists of a system, data, picture and sound element. The system, data and sound elements are CBE, while the picture element is VBE. The figure shows the concept of an index table slice. This is zero or more CBE elements followed by a VBE element or the end of the edit unit.

NOTE – The term "slice" as used above should not be confused with the same term as used by ISO/IEC 13818-2 (MPEG2).

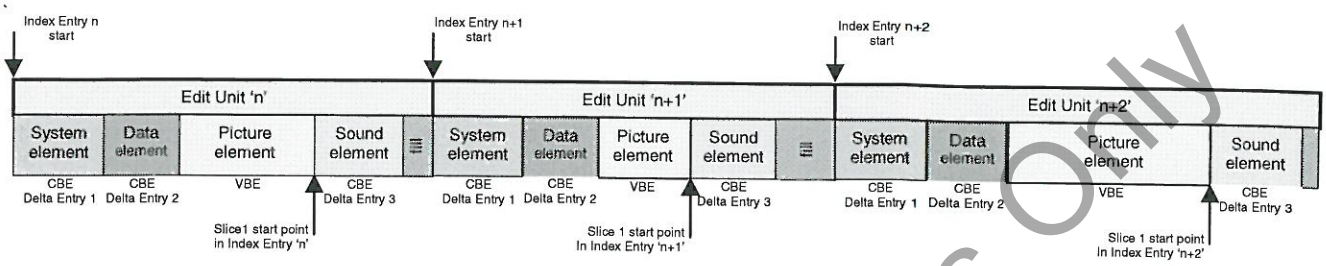

Figure 20 – Index table relationships for interleaved CBE/VBE stream

10.1.3 Complex interleaves of compressed audio

There are certain conditions in which the distribution of sound and data elements can be uneven. This can occur, for example, when compressed sound with large, indivisible frames is interleaved with picture having a different frame duration. In order to correctly index the synchronization points of the picture and sound, it is helpful to know the temporal offset of the first sound or data element in the indexed edit unit.

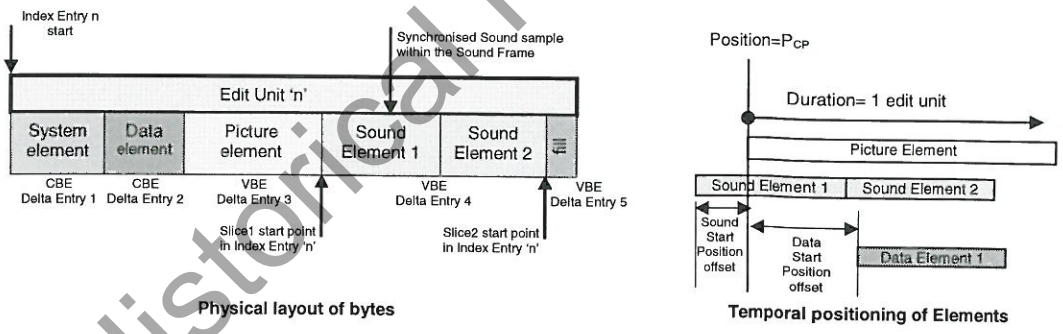

Figure 21 – Content package with interleaved large sound frames

Figure 21 shows a content package containing five elements and some fill. The interleaving of the data, picture and sound leads to different numbers of sound elements in the content package and therefore turns it into a VBE Element. The dark arrow at the top of the picture indicates the sound sample within the indivisible sound frame which is synchronized with the picture element. A temporal offset is recorded in the index table as a measure of the displacement from the start of the content package to the first sound sample in the element. When the content is KLV wrapped every edit unit (the most common wrapping format), this will result in a fractional position offset. This fractional measure allows precise synchronization of audio and data elements.

In figure 21, the sound offset is a negative fractional value and the data offset is a positive fractional value.

Note that this mechanism is optional and quantizing the synchronized components to a single frame may be sufficient in many applications.

10.1.4 Description of operation

An index table provides byte offset information within an essence container for a given time offset from the start of that essence container. If the essence container has interleaved data within in it, then extra mechanisms are provided for finding the offsets to the individual essence elements once the correct time offset is located. Each index entry provides the byte offset within the file for a given time offset measured in edit units. To locate the individual elements within the index entry, the delta entries and possible slice offsets are required. The extent to which the essence is indexed depends on an application. For many applications, simple indexing of the start of each edit unit will suffice. It is then up to the decoder within the application to find the start of each element by parsing.

Index entries –

An index entry provides the byte offset to the start of each edit unit within the essence container. Each index entry value marks the start of the key for the KLV packet of the first element in each edit unit. The temporal distance between index entries within an index table segment set is one edit unit. An index entry does not provide information about any interleave within the essence container. The essence type of the index entry shall be determined by inspection of the key of the essence (e.g., picture, sound, fill, etc.).

Slice offsets –

Each index entry value may have zero or more slice offset values that provide the byte offset within the edit unit to the end of any elements which are VBE. As can be seen in figure 20, each slice starts with a number of CBE elements and ends with a single VBE element. If the arrangement of elements in an essence container is such that the only VBE is the last element, then no slice offset values are needed. The start of slice zero corresponds to the start of the index entry.

Delta entries –

A single delta entry array provides information to find the elements within an interleaved essence container. There shall be one delta entry for each and every indexed element within the essence container. Any fill item may be indexed if desired. For example, a multiplex with three elements in the order CBE, VBE, CBE must have three delta entries:

1. CBE element: Delta entry (slice number=0, delta from start of slice= 0);
2. VBE element: Delta entry (slice number =0, delta from start of slice= sizeof(first_CBE_element));
3. CBE element: Delta entry (slice number =1, delta from start of slice= 0).

Any element that has minor sample variations (e.g., the 800/801 525-line audio sequence) must be padded to a constant size if it is to be regarded as a CBE, otherwise it is a VBE element and must use the slice mechanism. The essence type of the delta entry shall be determined by inspection of the key of the essence (e.g., picture, sound, fill, etc.).

10.1.5 Generalization using element delta

For each element within the interleaved essence container, the element delta defines whether the element is reordered temporally, which slice contains the element, and the (fixed) offset from the start of the slice to the start of the element. In addition, the delta entry allows the synchronization of an element relative to other elements to be calculated.

There is only one delta entry array per essence container stream, but may be different for each essence container stream. For reliability, the element delta is repeated in every segment of an index table stream. Elements which only appear occasionally in a stream (e.g., a DVB subtitle PES stream) shall be indexed as though they were a VBE element which is always present, but whose length is zero when no data exists in given edit unit.

10.1.6 Temporal reordering

In the particular case of MPEG long-GOP video, the compressed video pictures may be reordered from their display order according to the MPEG specification. This reordering is applied only to the video elements. An example is shown in figure 22.

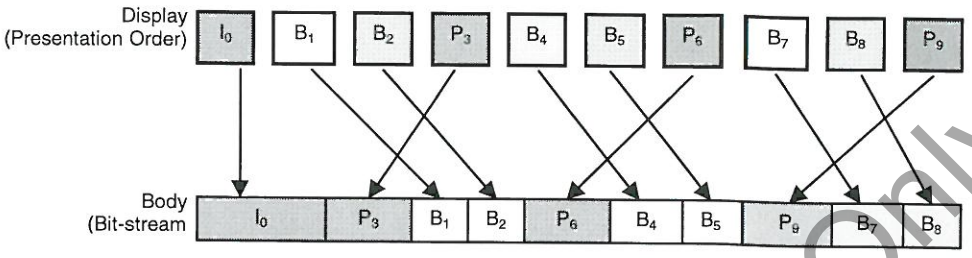

Figure 22 – Temporal reordering of MPEG-2 frame code video

MXF index tables provide a rapid conversion from the edit unit value into byte offsets within the essence container including display order offsets ("temporal offsets") for long GOP MPEG elements. For long GOP MPEG elements, the edit unit should be the video field, even when frame coding is used. This allows accurate and consistent indexing, even in the presence of irregular field sequences which can be created when film material has been edited in the video domain.

As an example, consider the sequence in figure 22 and figure 23. This is a typical frame coded Long GOP MPEG sequence with two B frames. As a result of the frame coding, each field within the frame has the same start position. The following example will access field 3, the second field of the first B frame.

The first step of the process is to look up the field number (in display order). From this, the reordering temporal offset of this field is found. This value is added to the original field number and then used as a second lookup in the same table to find the picture type and offset to the element within the stream. Note that the offset to element is calculated from the delta entry and index entry information as shown below.

This is represented schematically as follows:

Field Number	Index Entry
0 (I)	0
1 (I)	1
2 (B)	2
3 (B)	3
4 (B)	4
5 (B)	5
6 (P)	6
7 (P)	7
8 (B)	8

Temporal Offset
0
0
+2
+2
+2
+2
-4
-4
+2

Field Number	Picture Type	Offset to Element
0	I	00000100h
1	I	00000100h
2	P	0000c350h
3	P	0000c350h
4	B	0011b340h
5	B	0011b340h
6	B	0014c080h
7	B	0014c080h
8	P	0017ff40h

Figure 23 – Conversion from field number to offset with temporal offset

In the case of non-reordered elements such as sound, the temporal offset in the index entry for that edit unit is simply ignored when calculating the offset to element from the delta entry and index entry information. This is illustrated in figure 24.

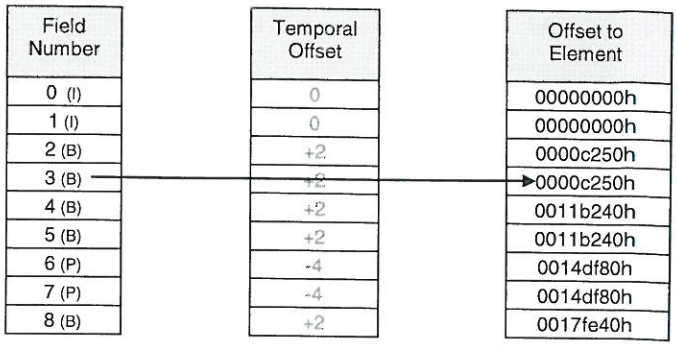

Figure 24 – Conversion from field number to offset ignoring temporal offset

10.1.7 Indexing KLV fill packets

KLV fill packets may be used to pad the essence elements to a given grid size. Where present, the KLV fill packets should have their own entry in the appropriate part of the index table (either as a delta entry or a slice offset). This will allow bit rates within the essence container to be (roughly) calculated when index tables do not index every single element in the stream. When index tables are fully populated, it will allow recalculation of the index table without parsing the stream when the KLV fill items are removed or changed to support a different grid size.

10.1.8 Constant edit unit size

In some simpler essence containers, the size of the edit units is constant. In this case, one of the properties in the index table segment allows a constant size value to be entered. This then requires no index entry array as the index entry for a given essence container is a simple multiplication of the edit unit size value multiplied by the edit unit count value.

The delta entry array may be equally simplified in such cases where the elements in the edit unit are of constant size.

The individual essence container specifications shall identify if the edit unit and its elements are of constant size and define appropriate values. Index table segments with these values may appear in any relevant partition, but each appearance of the index table segment shall have identical values.

A value of zero in the edit unit byte offset item shall identify that the size of the essence elements in each edit unit is not constant and that the indexing of edit units is to be found in the index entry array.

10.2 Index table specification

Index tables are specified as index table segments. A complete index table shall comprise one or more index table segments.

10.2.1 Index table segments

The size of index table segments is specific to the application. Typically, the size of index table segments including optional fill items is chosen to be a multiple of the KAG size.

Every index table segment shall specify the identifier of the index table (IndexSID) of which it is part, and the identifier of the essence container (BodySID) that it indexes.

An index table is intended to provide a conversion between edit units and byte offsets within an essence container in the MXF file. In an interleaved essence container, there may be several different components

such as picture, sound and data. Each of these components will have an associated track in the package which describes it. In order to construct a valid index table, the edit rate item in each indexed track in the package shall have the same value.

Zero or more index table segments may be inserted into any partition according to the rules in section 5.2. One or more index table segments may be included in each partition. An entire index table that comprises multiple index table segments may be put in a single partition by placing all the segments one after the other, in order in the partition.

Index tables may be repeated in the file. The manner in which this is done is application specific, but the following guidelines may be helpful:

- A repeated index table segment may accumulate prior index table segments in the file.
- When complete index tables are used, they should be placed in the header partition and/or the footer partition.
- A complete index table may be divided in order to provide distributed index table segments throughout the partitions in the file.
- MXF encoders shall ensure that for each repeated edit unit offset within the file, the byte offset values for each repeated indexed element shall be identical.

10.2.2 Index table segment key

A 16-byte SMPTE Universal label shall be used to identify an index table segment and act as a key for KLV coding of the index table segment. The index table segment key shall have the following value:

Table 18 – Specification of the index table segment key

Byte No.	Description	Value (hex)	Meaning
1	Object Identifier	06h	
2	Label size	0Eh	
3	Designator	2Bh	ISO
4	Designator	34h	SMPTE
5	Registry Category Designator	02h	Sets & Packs
6	Registry Designator	53h	Local Sets (2 byte tags, 2 byte lengths)
7	Structure Designator	01h	Set/Pack registry
8	Version Number	01h	Registry Version 1
9	Item Designator	0Dh	Organizationally registered
10	Organization	01h	AAF Association
11	Application	02h	MXF File Structure
12	Structure Version	01h	Version 1
13	Structure Kind	01h	File Structure sets & packs
14	Set/Pack Kind	10h	Index Table Segment
15	Version	01h	Index Table Specification version
16	reserved	00h	

10.2.3 Index table segment

Table 19 – Index table segment set

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Index Table Segment	Set Key	16		Table 18	Req	An Index Table Segment set	
Length	BER Length	var			Req	Set Length(see 8.3)	
Instance ID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Index Edit Rate	Rational	8	3F.0B	05.30.04.06	Req	Edit Rate copied from the tracks of the Essence Container [**RP 210** Specifies the indexing rate in hertz]	
Index Start Position	Position	8	3F.0C	07.02.01.03.01.0A	Req	The first editable unit indexed by this Index Table segment measured in File Package Edit Units [**RP 210** Specifies the position relative to start of essence, in edit units, where indexing starts]	
Index Duration	Length	8	3F.0D	07.02.02.01.01.02	Req	Time duration of this table segment measured in Edit Units of the referenced Package [**RP 210** Specifies the duration of an Index table in content units]	
Edit Unit Byte Count	UInt32	4	3F.05	04.06.02.01	D/Req	Defines the byte count of each and every Edit Unit in the Essence Container. A value of 0 defines the byte count of Edit Units is only given in the Index Entry Array [**RP 210** The length of an edit unit (in Bytes) in the container]	0
IndexSID	UInt32	4	3F.06	01.03.04.05	D/Req	Stream Identifier (SID) of Index Table [**RP 210** Index table stream ID]	
BodySID	UInt32	4	3F.07	01.03.04.04	Req	Stream Identifier (SID) of the indexed Essence Container [**RP 210** Essence (or its container) stream ID]	
Slice Count	UInt8	1	3F.08	04.04.04.01.01	D/Req	Number of slices minus 1 (NSL) [**RP 210** Number of sections indexed, per edit unit, minus one]	0
PosTableCount	UInt8	1	3F.0E	04.04.04.01.07	Opt	Number of PosTable Entries minus 1 (NPE) [**RP 210** Number of position offsets indexed, per edit unit, minus one]	
Delta Entry Array	Array of DeltaEntry	var	3F.09	04.04.04.01.06	Opt	Map Elements onto Slices (see Table 20) [**RP 210** Array of values used to identify elements of Essence within an edit unit]	{0,6}
Index Entry Array	Array of IndexEntry	var	3F.0A	04.04.04.02.05	D/Req	Index from Edit Unit number to stream offset (see Table 21) [**RP 210** Array of values used to index elements from edit unit to edit unit]	

NOTES

1. Where the edit unit byte count value is non-zero, the duration value may be set to zero to indicate that this index table segment applies to the entire essence container identified by this BodySID. In this case, identical index table segments may be repeated in partitions where required by the application.

2. All index table segments in each partition shall have the same index SID value
3. All index table segments in one index stream (i.e., with the same index SID) shall have the same values of body SID, slice count and delta entry array.
4. For clarification from table above, the size of the DeltaEntry Array is 8 + (NDE*Sizeof(DeltaEntry)).
5. For clarification from table above, the size of the IndexEntry Array is 8 + (NIE*Sizeof(IndexEntry)).
6. The edit rate value in an index table segment shall be the same as the edit rate value in all the tracks that are indexed by that index table segment. All index table segments with the same index SID value shall have the same edit rate value.
7. The first index entry in the index entry array within an index table segment, shall give information about the byte offsets for the temporal location given by the start position item within that segment.
8. The PosTableCount gives the total number of elements which have fractional positions recorded according to section 10.1.3.

Delta entry array is a value where the order of the entries is significant and defines byte offset values along an incrementing timeline. The first 4 bytes shall be the number of deltas entries (NDE), the second 4 bytes shall be the length of each entry. The structure of each delta entry is defined in table 20.

Table 20 – Structure of delta entry array

	N	Item Name	Type	Len	UL Designator	Req ?	Meaning	Default
	1	NDE	UInt32	4	N/A	Req	Number of delta entries	
	1	Length	UInt32	4	N/A	Req	Length of each delta entry	
Delta Entry	N D E	PosTableIndex	Int8	1	04.04.04.01.04	Req	-1=Apply Temporal Reordering 0= No temporal Reordering N=Index into PosTable [RP 210 value identifying that the element indexed is subject to temporal reordering or offsetting of edit units]	0
		Slice	UInt8	1	04.04.04.01.02	Req	Slice number in IndexEntry [RP 210 The number of the indexed section in the edit unit]	0
		Element Delta	UInt32	4	04.04.04.01.03	Req	Delta from start of slice to this Element [RP 210 The number of bytes from the start of the section to this element.]	

PosTableIndex is used to discover if this element has been temporally reordered or not. If the value is negative then the temporal offset property of the IndexEntryArray shall be used to determine the difference between presentation order and storage order of the indexed element. If the value is positive then this indexed element has a fractional temporal offset stored in the PosTable within the IndexEntry array. The value of PosTableIndex is the index into this table (1 is the first entry, 2 is the second, etc.). If the value is zero, there is no reordering and no temporal offsetting for this element.

Index entry array is a value where the order of the entries is significant and defines values along an incrementing timeline. The first 4 bytes shall be the number of index entries (NIE), the second 4 bytes shall be the length of each entry, which in turn shall depend on the number of slices (NSL) in this index table. Note that the number of slices shall be fixed. The structure of each index entry is defined in table 21.

Table 21 – Structure of index entry array

N	Item Name	Type	Len	UL Designator	Req ?	Meaning	Default
1	NIE	UInt32	4	N/A	Req	Number of index entries	
1	Length	UInt32	4	N/A	Req	Length of each index array entry	
Index Entry: NIE	Temporal Offset	Int8	1	04.04.04.02.03	Req	Offset in edit units from Display Order to Coded Order [**RP 210** The number of edit units by which this edit unit has been moved in the bitstream for the purpose of temporal reordering (e.g. MPEG)]	0
	Key-Frame Offset	Int8	1	04.04.04.02.04	Req	Offset in edit units to previous Key-Frame. The value is zero if this is a Key-Frame. [**RP 210** The offset in edit units from this edit unit to the previous Key-Frame edit unit (e.g. previous I-frame in MPEG-2)]	0
	Flags	EditUnitFlag	1	04.04.04.02.02	Req	Flags for this Edit Unit Bit 7: Random Access Bit 6: Sequence Header Bit 5: forward prediction flag Bit 4: backward prediction flag e.g. 00= I frame (no prediction) 10= P frame(forward prediction from previous frame) 01= B frame (backward prediction from future frame) 11= B frame (forward & backward prediction) Bits 0-3: reserved [**RP 210** Flags to indicate coding of elements in this edit unit]	80h
	Stream Offset	UInt64	8	04.04.04.02.01	Req	Offset in bytes from the first KLV element in this Edit Unit within the Essence Container Stream [**RP 210** The offset of the edit unit within the container stream relative to the start of that container stream]	
	SliceOffset	NSL x UInt32	4* NSL	04.04.04.01.05	Opt	The offset in bytes from the Stream Offset to the start of this slice. [**RP 210** List of the offsets within the edit unit of each indexed section (except the first)]	
	PosTable	NPE *Rational	8* NPE	04.04.04.01.08	Opt	The fractional position offset from the start of the content package to the synchronized sample in the Content Package (Figure 21) [**RP 210** List of the fractional temporal offsets of indexed elements relative to the indexed position]	

NOTES

1. Temporal offset shall be only applied for elements whose PosTableIndex value in the element delta is -1.
2. Temporal offset shall be zero (0) if the elements are not reordered.
3. Key-frame offset shall be only applied for predictive coding schemes where a key-frame must be decoded in order to decode the indexed frame. It is the offset index to find the key-frame required to decode the indexed frame.
4. The random access flag bit shall be 1 if this edit unit is a bona fide random access point in the stream.
5. The sequence header flag bit shall be 1 if this edit unit includes an MPEG sequence header. It shall be FALSE if the edit unit does not include a sequence header or if the compression scheme does not make use of such a construct.
6. The forward and backwards prediction flags shall be set to zero if the flags are not appropriate for the compression scheme in the indexed essence.
7. The PosTable is an ordered list of signed fractional position offsets for Indexed Elements. If an Element has an entry in this array then the PosTableIndex property in its DeltaEntry will be a positive Integer. Figure 21 and its associated text shows the meaning of the sign of this property.

10.3 Partial / sparse index tables

The most densely populated index table will have an index entry for every edit unit. This specification allows an MXF encoder to create several IndexTable segments, each containing a single IndexEntry where the temporal distance between the start positions of the segments is greater than one edit unit. This creates sparse index tables and MXF decoders must cope with this eventuality.

The temporal location of each sparse entry is given by the start position property of the index table segment. The byte offset to the data is calculated the same way that the first entry of any index table segment would be calculated. Applications control whether these sparse index table entries should be regularly spaced along the time axis or whether they have no fixed pattern.

10.4 To find the byte offset for an element

To find a particular element within a given edit unit, first locate the index table segment which contains the desired edit unit. Apply the temporal offset to the edit unit number if appropriate, which will give you the required index entry for this edit unit. Inspecting the DeltaEntry Array for the desired element will provide the slice number and element delta (i.e., offset from the start of the slice for the element). The byte offset of the element is given by adding:

	Element Delta	from DeltaEntry array
+	Slice Offset	from Index Entry array of Slice Offsets. Slice Number in the DeltaEntry array
+	Stream Offset	from Index Entry array for the Edit Unit

10.5 Look-up algorithm (informative)

10.5.1 Conversion from presentation time to stream offset

The pseudo-code presented here is an informative aid to understand the algorithm which returns the stream offset of the T'th edit unit of a given sub-stream K. Variable names are the same as used in the text above. The example assumes that the index table is fully populated.

The code first locates the correct index table segment by checking to see if the desired edit unit T is within the scope of the current index table segment. Once that has been found, the conversion (if any) between coded order and display order is performed according to the example in section 10.1.6. The stream offset for this temporal location can then be found. The substream (e.g., CBE sound) slice is located in the element delta. If it is not the first slice then the offset to that element's slice is added, and finally the element delta is added onto the previously calculated stream location to locate the start of this substream. Note that the offsets refer to the byte address within a stream, including all the Ks, Ls and Vs of each KLV item.

```
typedef struct {
            int8 PosTableIndex;     /* <0 when this sub-stream is temporally reordered */
                                    /* >0 to indicate that start positions of this substream
                                    are recorded in PosTable */
            uint8 Slice;
            uint32 ElementDelta;
      } DeltaEntry;

typedef struct {
            int8 TemporalOffset;
            int8 KeyFrameOffset;
            uint8 EditUnitFlags;
            uint64 StreamOffset;
            uint32 SliceOffset[SliceCount];
                  Rational PosTable[PosTableCount];
      } IndexEntry;

typedef struct {
            UUID InstanceID;
            Rational IndexEditRate;
            Position StartPosition;
            UInt64 Duration;
            UInt32 CBESize;
            UInt32 IndexSID;
            UInt32 BodySID;
            UInt8 SliceCount;
            UInt8 PosTableCount;
            DeltaEntry ElementDelta[NEL];
            IndexEntry EditUnitIndex[Duration];
      } IndexTableSegment;

/* external function to skip to previous table segment */
extern IndexTableSegment* Previous( IndexTableSegment *I);

/* external function to skip to next table segment */
extern IndexTableSegment* Next( IndexTableSegment *I);

IndexTableSegment* I;   /* Current Index Table Segment */
Position T;             /* T is edit unit number counted from start of body */
Substream K;            /* K is substream number, e.g. Sys==0,V==0,A==1,Dat==2 */

UInt64 StreamOffset( Position T, Substream K, IndexTableSegment *I) {
      /* T is edit unit number counted from start of Essence Container */
      /* K is substream number, for example Sys==0, V==0, A==1, Dat==2 */

      while ( T < I->StartPosition ) I = Previous(I);
      while ( T>= I->StartPosition + I->Duration) I = Next(I);

      T -= I->StartPosition;
      #DEFINE REORDER -1
      if (I->ElementDelta[K].PosIndex <= REORDER)
      T += I-> EditUnitIndex[T].TemporalOffset;
      UInt64 Result = I-> EditUnitIndex [T].StreamOffset;

      uint8 Slice = I->ElementDelta[K].Slice;
      if (Slice>0) Result += I->EditUnitIndex [T].SliceOffset[Slice-1];

      Result += I->ElementDelta[K].ElementDelta;

      /* Now retrieve the offset to the start position - if any */
      unit8 PosTableIndex = I->ElementDelta[K].PosIndex;
      Rational pos = 0;
      if (PosTableIndex>0) pos= I->EditUnitIndex [T].PosTable[PosTableIndex-1];

      /* pos now contains fractional component of the Subframe for synchronisation */
      Result= process_fractional_Index(T, K, I, Result, pos)

      Return Result;
}
```

11 Random index pack

The random index pack is a device to help find partitions scattered throughout an MXF file. It is a fixed length pack which defines the BodySID and byte offset to the start of each partition (i.e., the first byte of the partition pack key). This pack can be used by decoders to rapidly access index tables and to find the partitions to which an index table points. The random index pack is optional and if it exists it shall follow the footer partition and shall be the last KLV item in a file.

11.1 Random index pack key

The 16-byte SMPTE Universal label of the random index pack is defined below.

Table 22 – Random index pack key value

Byte No.	Description	Value (hex)	Meaning
1	Object Identifier	06h	
2	Label size	0Eh	
3	Designator	2Bh	ISO, ORG
4	Designator	34h	SMPTE
5	Registry Category Designator	02h	Sets & Packs
6	Registry Designator	05h	Fixed length pack (no length fields)
7	Structure Designator	01h	Set/Pack Registry
8	Version Number	01h	Registry Version 1
9	Item Designator	0Dh	Organizationally registered
10	Organization	01h	AAF
11	Application	02h	MXF File Structure
12	Structure Version	01h	Version 1
13	Structure Kind	01h	File Structure sets & packs
14	Set/Pack kind	11h	Random Index Pack
15	Version	01h	Random Index Pack Version
16	Reserved	00h	

11.2 Random index pack value

The structure of the random index pack value is given below.

Table 23 – Data structure of the random index pack value

N	Item Name	Type	Len	UL Designator	Meaning	Default
	Random Index Metadata	Set Key	16	See Table 22		
	Length	BER Length	n		Overall Length of Pack	
N pairs	BodySID	UInt32	4	01.03.04.04	Stream ID of the Body in this partition [RP 210 Essence (or its container) stream ID]	
	Byte Offset	UInt64	8	06.09.02.01.01	Byte offset from file start (1st byte of the file which is numbered 0) to the 1st byte of the Partition Pack Key [RP 210 Byte offset from start of file (byte 0) to 1st byte of partition pack key]	
	Length	UInt32	4	04.06.10.01	Overall Length of this Pack including the Set Key and BER Length fields [RP 210 Big-endian overall length of set or pack]	

If an MXF file contains 'N' partitions (including header partition and footer partition) then the table will contain 'N' pairs of values. Each pair of values relates the byte offset of the first byte of the key of the partition pack of a partition to the BodySID in that partition. The pairs shall be stored in ascending byte offset order and every partition shall be indexed if the random index pack exists.

Note that when partitions are relocated within a file (e.g., when moving index table segments) the random index pack must be recomputed or deleted.

11.3 Algorithm for using the random index pack (informative)

The following algorithm in pseudo-code allows the random index pack to be found and read. Its use is application specific and dependent on the use of index tables.

```
Seek_to_(MXF_FILE, END_OF_FILE-4);      //go to end of the MXF file
L= read_UInt32(MXF_FILE);               //read the length
If (L < UPPER_LIMIT)                    //check for silly values
{
Seek_to_(MXF_FILE, END_OF_FILE-L);      //Go to start of Random Index Pack
RIP= Read_RIP(MXF_FILE);                //Read the Random Index Pack
RIP_EXISTS= Check_Key(RIP);             //Final check that it was a valid RIP
}
```

Annex A (normative)
Specifications for root metadata sets

A.1 Preface

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Preface	Set Key	16		see Table 13	Req	Defines the Preface set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [**RP 210** Specifies the reference to an overall modification]	
Last Modified Date	Timestamp	8	3B.02	07.02.01.10.02.04	Req	Date & time of the last modification of the file [**RP 210** Identifies date and time at the point of most recent modification of any item in the container]	
Version	Version Type	2	3B.05	03.01.02.01.05	Req	See section 3.2. The value shall be 258 (i.e. v1.2)	
Object Model Version	UInt32	4	3B.07	03.01.02.01.04	Opt	Simple integer version number of Object Model The value, if present, shall be 1 [**RP 210** Specifies the Internal Object Storage Mechanism Version Number]	
Primary Package	WeakRef (Package)	16	3B.08	06.01.01.04.01.08	Opt	The primary Package in this file see 7.5 [**RP 210** Specifies a reference to the primary package in this file]	UID of the primary Material Package
Identifications	StrongRefArray (Identification)	8+16n	3B.06	06.01.01.04.06.04	Ereq	Ordered array of strong references to Identification sets recording all modifications to the file [**RP 210** Specifies a vector of references to modification identifiers]	
Content Storage	StrongRef (ContentStorage)	16	3B.03	06.01.01.04.02.01	Req	Strong reference to Content Storage object [**RP 210** Specifies a reference to the packages and essence in a file]	
Operational Pattern	UL	16	3B.09	01.02.02.03	Req	Universal Label of the Operational Pattern which this file complies to (Partition Pack copies this value) [**RP 210** Specifies the SMPTE Universal Label that locates an Operational Pattern]	
EssenceContainers	ULBatch (Essence Containers)	8+ 16n	3B.0A	01.02.02.10.02.01	Req	an unordered Batch of ULs of Essence Containers used in or referenced by this file [**RP 210** Batch of universal labels of all essence containers in the file]	
DM Schemes	ULBatch (DMSchemes)	8+ 16n	3B.0B	01.02.02.10.02.02	Req	An unordered Batch of Universal Labels of all the Descriptive Metadata schemes used in this file [**RP 210** An unordered Batch of Universal Labels of all the Descriptive Metadata schemes used in this file]	

NOTE – EssenceContainers is a batch of ULs that identifies the different essence container types used in this MXF file. This should be complete where possible. If the partition in which this set is located is closed, the values shall be complete and correct. There shall be no duplicate UL values in the batch of ULs. The essence container ULs placed in this batch

should be for those essence containers which appear internally in the file. These essence container ULs shall have the same value as those found in the essence descriptors for the file package(s) directly referenced by the material package(s) in the file.

A.2 Identification

NOTE – The first Identification set is generated when the file is created and a new set is generated and added whenever the file is modified. Each Identification set has a "This Generation UID" property value to which any other set may refer via its own "generation UID" property. This is used to indicate at what stage the set and its associated sets were created or modified. The identification set with the matching generation UID shall contain the details corresponding to the moment when those sets were created or modified.

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Identification	Set Key	16		see Table 13	Req	Defines the Identification set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
This Generation UID	UUID	16	3C.09	05.20.07.01.01	Req	Generation Identifier to be referenced by other sets (see section 6.6.2 for use of Generation IDs) [**RP210** Specifies the reference to a particular modification.]	
Company Name	UTF-16 string	var	3C.01	05.20.07.01.02.01	Req	Manufacturer of the equipment or application that created or modified the file [**RP 210** Specifies the name of the application provider]	
Product Name	UTF-16 string	var	3C.02	05.20.07.01.03.01	Req	Name of the application which created or modified this file [**RP 210** Specifies the name of the application product]	
Product Version	Product Version	10	3C.03	05.20.07.01.04	Opt	Maj.min.tweak.build.rel version number of this application (see 3.3) [**RP 210** Specifies version information for the application]	
Version String	UTF-16 string	var	3C.04	05.20.07.01.05.01	Req	Human readable name of the application version [**RP 210** Specifies version information for the application in textual form]	
Product UID	UUID	16	3C.05	05.20.07.01.07	Req	A unique identification for the product which created this file (defined by the manufacturer) [**RP 210** Specifies a reference to the application product definition]	
Modification Date	Timestamp	8	3C.06	07.02.01.10.02.03	Req	Time & date an application created or modified this file and created this Identification set (see 3.3 [**RP 210** Identifies date and time at the point of modification]	
ToolkitVersion	Product Version	10	3C.07	05.20.07.01.0A	Opt	Maj.min.tweak.build.rel version of software or hardware codec used (see 3.3) [**RP 210** Specifies the SDK version number for a modification]	
Platform	UTF-16 string	var	3C.08	05.20.07.01.06.01	Opt	Human readable name of the operating system used [**RP 210** Specifies the platform on which the application was run]	

A.3 Content storage

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Content Storage	Set Key	16		see Table 13	Req	Defines the Content Storage set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [RP210 The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [RP210 Specifies the reference to an overall modification]	
Packages	StrongRef Batch (Packages)	8+ 16n	19.01	06.01.01.04.05.01	Req	Unordered Batch of strong references to all packages used in this file [RP210 Specifies a unordered set of references to Packages]	
EssenceContainer Data	StrongRef Batch (Container Data)	8+ 16n	19.02	06.01.01.04.05.02	Opt	Unordered Batch of strong references to Essence Container Data sets used in this file [RP210 Specifies a unordered set of references to Essence Data]	

A.4 Essence container data

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Essence Container Data	Set Key	16		see Table 13	Req	Defines the Essence Container Data set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [RP 210 The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Linked Package UID	UMID	32	27.01	06.01.01.06.01	Req	Identifier of the Package to which this set is linked as a UMID [RP 210 Specifies a reference to a package associated with Essence Data]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation identifier [RP 210 Specifies the reference to an overall modification]	
IndexSID	UInt32	4	3F.06	01.03.04.05	Opt	ID of the Index Table for the Essence Container to which this set is linked [RP 210 Index table stream ID]	
BodySID	UInt32	4	3F.07	01.03.04.04	Req	ID of the Essence Container to which this set is linked The value 0 indicates there is no Essence Container data in this partition. [RP 210 Essence (or its container) stream ID]	

Annex B (normative)
Specifications for the generic package

B.1 Generic package

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Generic Package	Set Key	16		Defined by Package type	Req	Defines a Generic Package set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [RP210 The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Package UID	UMID	32	44.01	01.01.15.10	Req	Unique Package Identifier as a UMID [RP 210 Identifies the Metadata Object with a unique identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [RP 210 Specifies the reference to an overall modification]	
Name	UTF-16 string	var	44.02	01.03.03.02.01	Opt	Human readable package name [RP 210 Identifies the AAF metadata object by name]	
Package Creation Date	Timestamp	8	44.05	07.02.01.10.01.03	Req	The date & time of creation of this package (see section 3.3) [RP 210 Identifies date and time at the point of creation.]	
Package Modified Date	Timestamp	8	44.04	07.02.01.10.02.05	Req	The date & time of last modification of this package (see section 3.3) [RP 210 Identifies date and time at the point of most recent modification of the package]	
Tracks	StrongRefArray (Tracks)	8+ 16n	44.03	06.01.01.04.06.05	Req	Ordered Array of Strong References to Tracks [RP 210 Specifies a vector of references to tracks]	

B.2 Generic descriptor

The properties of the generic descriptor are given in table 17.

B.3 Network locator

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Network Locator	Set Key	16		see table 13	Req	Defines the Network Locator set for location with a URL	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [RP 210 The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [RP 210 Specifies the reference to an overall modification]	
URL String	UTF-16 string	var	40.01	01.02.01.01.01	Req	A URL indicating where the essence may be found. [RP210 Unique Resource Locator String]	

B.4 Text locator

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Text Locator	Set Key	16		see Table 13	Req	Defines the Text Locator set for location with a human-readable text string	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [**RP 210** Specifies the reference to an overall modification]	
Locator Name	UTF-16 string	var	41.01	01.04.01.02.01	Req	Value of a human-readable locator text string for manual location of essence [**RP 210** A description of the physical location of media - e.g. which archive, place, rack, shelf, position on shelf]	

B.5 Track (time code)

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Track	Set Key	16		see Table 13	Req	Defines the Track set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [**RP 210** Specifies the reference to an overall modification]	
Track ID	UInt32	4	48.01	01.07.01.01	D/Req	ID of the track in this package (for linking to a SourceTrackID in a SourceClip) [**RP 210** Specifies the particular track within a package]	
Track Number	UInt32	4	48.04	01.04.01.03	Req	Number used to link to the Timecode track within the Essence Container if it exists [**RP 210** Specifies a reference to the codec used to create Essence Data]	0
Track Name	UTF-16 string	var	48.02	01.07.01.02.01	Opt	Human readable name of the track type [**RP 210** Specifies the particular track within a package by a name]	"Master Timecode"
Edit Rate	Rational	8	4B.01	05.30.04.05	Req	Edit Rate of Track [**RP 210** Specifies the timeline rate in hertz]	
Origin	Position	8	4B.02	07.02.01.03.01.03	Req	An Offset used to resolve timeline references to this track. The start of the track has this timeline value measured in Edit Units. [**RP 210** Specifies the point, in edit units, in a track from which relative times are measured.]	
Sequence	StrongRef (Sequence)	16	48.03	06.01.01.04.02.04	Req	Strong reference to the Time Code Sequence set [**RP 210** Specifies a reference to a segment of a track]	

B.6 Sequence (time code)

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Sequence	Set Key	16		see Table 13	Req	Defines the Sequence set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [**RP 210** Specifies the reference to an overall modification]	
Data Definition	UL	16	02.01	04.07.01	Req	Specifies the data type of this set. (value= SMPTE Time Code – see SMPTE RP 224) [**RP 210** Specifies the basic essence type of a component]	
Duration	Length	8	02.02	07.02.02.01.01.03	B.Effort	Duration of Sequence (in units of edit rate) Distinguished Value = -1 [**RP 210** The absolute duration of a compositional component - e.g. clip, effect, sequence etc in units of edit rate]	
Structural Components	StrongRefArray(Structural Component)	8+16n	10.01	06.01.01.04.06.09	Req	Ordered array of strong references to Time Code Structural Components [**RP 210** Specifies a vector of references to the clips and transitions in the sequence]	

B.7 Time code component

NOTE – This set is used to define continuous time code over the duration of this component. This set may be used in any package.

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Time Code Component	Set Key	16		see Table 13	Req	Defines the Time Code Component set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [**RP 210** Specifies the reference to an overall modification]	
Data Definition	UL	16	02.01	04.07.01	Req	Specifies the data type of this set. (value= SMPTE Time code - see SMPTE RP224) [**RP 210** Specifies the basic essence type of a component]	
Duration	Length	8	02.02	07.02.02.01.01.03	B.Effort	Duration of SourceClip (in units of edit rate) Distinguished Value = -1 [**RP 210** The absolute duration of a compositional component - e.g. clip, effect, sequence etc in units of edit rate]	

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Rounded Time Code Base	Uint16	2	15.02	04.04.01.01.02.06	Req	Nearest Integer frames per second [**RP 210** e.g. 24, 25, 30, 48, 60]	
Start Time Code	Position	8	15.01	07.02.01.03.01.05	Req	Starting timecode (converted to integer frame count from 00:00:00:00) [**RP 210** The time code within a track at the starting point of the essence]	
Drop Frame	Boolean	1	15.03	04.04.01.01.05	Req	True = Drop Frame Time Code in use [**RP 210** Specifies whether time code is drop frame (Non-drop Frame = 0)]	

NOTES
1. An algorithm for determining the rounded time code base = integer_part_of(frame rate + 0.5).
2. This set is a structural component.

B.8 Track (picture)

NOTE – The number of picture track sets is determined by the number of editable picture tracks in the essence container.

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Track	Set Key	16		see Table 13	Req	Defines the Track set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation identifier [**R 210** Specifies the reference to an overall modification]	
Track ID	UInt32	4	48.01	01.07.01.01	D/Req	ID of the picture track in this package (for linking to a SourceTrackID in a SourceClip) [**RP 210** Specifies the particular track within a package]	
Track Number	UInt32	4	48.04	01.04.01.03	Req	Number used to link to the Picture track in the Essence Container. Its use is specified in the appropriate Essence Container document [**RP 210** Specifies a reference to the codec used to create Essence Data]	
Track Name	UTF-16 string	var	48.02	01.07.01.02.01	Opt	Human readable name of the track type [**RP 210** Specifies the particular track within a package by a name]	"Picture"
Edit Rate	Rational	8	4B.01	05.30.04.05	Req	Edit Rate of Track [**RP 210** Specifies the timeline rate in hertz]	
Origin	Position	8	4B.02	07.02.01.03.01.03	Req	An Offset used to resolve timeline references to this track. The start of the track has this timeline value measured in Edit Units. [**RP 210** Specifies the point, in edit units, in a track from which relative times are measured.]	
Sequence	StrongRef (Sequence)	16	48.03	06.01.01.04.02.04	Req	Strong reference to the Sequence set [**RP 210** Specifies a reference to a segment of a track]	

B.9 Sequence (picture)

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Sequence	Set Key	16		see Table 13	Req	Defines the Sequence set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [**RP 210** Specifies the reference to an overall modification]	
Data Definition	UL	16	02.01	04.07.01	Req	Specifies the data type of this set. (value= Picture Essence - see SMPTE RP224) [**RP 210** Specifies the basic essence type of a component]	
Duration	Length	8	02.02	07.02.02.01.01.03	B.Effort	Duration of Sequence (in units of edit rate) Distinguished Value = -1 [**RP 210** The absolute duration of a compositional component - e.g. clip, effect, sequence etc in units of edit rate]	
Structural Components	StrongRefArray (Structural Component)	8+16n	10.01	06.01.01.04.06.09	Req	Array of strong references to Picture Structural Components [**RP 210** Specifies a vector of references to the clips and transitions in the sequence]	

B.10 SourceClip (picture)

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
SourceClip	Set Key	16		see Table 13	Req	Defines the SourceClip set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [**RP210** Specifies the reference to an overall modification]	
Data Definition	UL	16	02.01	04.07.01	Req	Specifies the data type of this set. (value= Picture Essence - see SMPTE RP224) [**RP210** Specifies the basic essence type of a component]	
Start Position	Position	8	12.01	07.02.01.03.01.04	Req	Offset into Essence measured in edit units of the track containing this SourceClip	
Duration	Length	8	02.02	07.02.02.01.01.03	B.Effort	Duration of SourceClip (in units of edit rate) Distinguished Value = -1 [**RP 210** The absolute duration of a compositional component - e.g. clip, effect, sequence etc in units of edit rate]	
SourcePackageID	Package ID	32	11.01	06.01.01.03.01	Req	ID of referenced Package as a UMID The value shall be 32 zero valued bytes to terminate the source reference chain [**RP 210** Specifies the reference to a precursor]	

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
SourceTrackID	Track ID (UInt32)	4	11.02	06.01.01.03.02	Req	Track ID of the referenced Track within the referenced Package. Zero indicates end of reference chain. [**RP 210** Specifies the track within the referenced precursor]	

NOTE – This set is a structural component.

B.11 Track (sound)

NOTE – The number of sound track sets is determined by the number of editable sound tracks in the essence container.

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Track	Set Key	16		see Table 13	Req	Defines the Track set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation identifier [**RP 210** Specifies the reference to an overall modification]	
Track ID	UInt32	4	48.01	01.07.01.01	D/Req	ID of the sound track in this package (for linking to a SourceTrackID in a SourceClip) [**RP 210** Specifies the particular track within a package]	
Track Number	UInt32	4	48.04	01.04.01.03	Req	Number used to link to the sound track in the Essence Container. Its use is specified in the appropriate Essence Container document [**RP 210** Specifies a reference to the codec used to create Essence Data]	
Track Name	UTF-16 string	var	48.02	01.07.01.02.01	Opt	Human readable name of the track type [**RP 210** Specifies the particular track within a package by a name]	"Sound"
Edit Rate	Rational	8	4B.01	05.30.04.05	Req	Edit Rate of Track [**RP 210** Specifies the timeline rate in hertz]	
Origin	Position	8	4B.02	07.02.01.03.01.03	Req	An Offset used to resolved timeline references to this track. The start of the track has this timeline value measured in Edit Units. [**RP 210** Specifies the point, in edit units, in a track from which relative times are measured.]	
Sequence	StrongRef (Sequence)	16	48.03	06.01.01.04.02.04	Req	Strong reference to Sequence set [**RP 210** Specifies a reference to a segment of a track]	

B.12 Sequence (sound)

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Sequence	Set Key	16		see Table 13	Req	Defines the Sequence set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [**RP210** Specifies the reference to an overall modification]	
Data Definition	UL	16	02.01	04.07.01	Req	Specifies the data type of this set. (value= Sound Essence - see SMPTE RP224) [**RP210** Specifies the basic essence type of a component]	
Duration	Length	8	02.02	07.02.02.01.01.03	B.Effort	Duration of Sequence (in units of edit rate) Distinguished Value = -1 [**RP210** The absolute duration of a compositional component - e.g. clip, effect, sequence etc in units of edit rate]	
Structural Components	StrongRefArray (Structural Components)	8+16n	10.01	06.01.01.04.06.09	Req	Ordered array of strong references to Sound Structural Components [**RP210** Specifies a vector of references to the clips and transitions in the sequence]	

B.13 SourceClip (sound)

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
SourceClip	Set Key	16		see Table 13	Req	Defines the SourceClip set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [**RP 210** Specifies the reference to an overall modification]	
Data Definition	UL	16	02.01	04.07.01	Req	Specifies the data type of this set. (value= Sound Essence - see SMPTE RP224) [**RP 210** Specifies the basic essence type of a component]	
Start Position	Position	8	12.01	07.02.01.03.01.04	Req	Offset into Essence measured in edit units of the track containing this SourceClip	
Duration	Length	8	02.02	07.02.02.01.01.03	B.Effort	Duration of SourceClip (in units of edit rate) Distinguished Value = -1 [**RP 210** The absolute duration of a compositional component - e.g. clip, effect, sequence etc in units of edit rate]	
SourcePackageID	Package ID	32	11.01	06.01.01.03.01	Req	ID of referenced Package as a UMID The value shall be 32 zero valued bytes to terminate the source reference chain [**RP 210** Specifies the reference to a precursor]	
SourceTrackID	Track ID (UInt32)	4	11.02	06.01.01.03.02	Req	Track ID of the referenced Track within the referenced Package . Zero indicates end of reference chain. [**RP210** Specifies the track within the referenced precursor]	

NOTE – This set is a structural component.

B.14 Track (data)

NOTE – The number of data track sets is determined by the number of editable data tracks in the essence container.

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Track	Set Key	16		see Table 13	Req	Defines the Track set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation identifier [**RP 210** Specifies the reference to an overall modification]	
Track ID	UInt32	4	48.01	01.07.01.01	D/Req	ID of the Data track in this package (for linking to a SourceTrackID in a SourceClip) [**RP 210** Specifies the particular track within a package]	
Track Number	UInt32	4	48.04	01.04.01.03	Req	Number used to link to the Data track in the Essence Container. Its use is specified in the appropriate Essence Container document [**RP 210** Specifies a reference to the codec used to create Essence Data]	
Track Name	UTF-16 string	var	48.02	01.07.01.02.01	Opt	Human readable name of the track type [**RP 210** Specifies the particular track within a package by a name]	"Auxiliary Data"
Edit Rate	Rational	8	4B.01	05.30.04.05	Req	Edit Rate of Track [**RP 210** Specifies the timeline rate in hertz]	
Origin	Position	8	4B.02	07.02.01.03.01.03	Req	An Offset used to resolved timeline references to this track. The start of the track has this timeline value measured in Edit Units. [**RP 210** Specifies the point, in edit units, in a track from which relative times are measured.]	
Sequence	StrongRef (Sequence)	16	48.03	06.01.01.04.02.04	Req	Strong reference to Sequence set [**RP 210** Specifies a reference to a segment of a track]	

B.15 Sequence (data)

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Sequence	Set Key	16		see Table 13	Req	Defines the Sequence set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [**RP 210** Specifies the reference to an overall modification]	
Data Definition	UL	16	02.01	04.07.01	Req	Specifies the data type of this set. (value= Data Essence - see SMPTE RP224) [**RP 210** Specifies the basic essence type of a component]	
Duration	Length	8	02.02	07.02.02.01.01.03	B.Effort	Duration of Sequence (in units of edit rate) Distinguished Value = -1 [**RP 210** The absolute duration of a compositional component - e.g. clip, effect, sequence etc in units of edit rate]	
Structural Components	StrongRefArr (Structural Components)	8+16n	10.01	06.01.01.04.06.09	Req	Ordered array of strong references to Data Structural Components [**RP 210** Specifies a vector of references to the clips and transitions in the sequence]	

B.16 SourceClip (data)

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
SourceClip	Set Key	16		see Table 13	Req	Defines the SourceClip set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [**RP 210** Specifies the reference to an overall modification]	
Data Definition	UL	16	02.01	04.07.01	Req	Specifies the data type of this set. (value= Data Essence - see SMPTE RP224) [**RP 210** Specifies the basic essence type of a component]	
Start Position	Position	8	12.01	07.02.01.03.01.04	Req	Offset into Essence measured in edit units of the track containing this SourceClip	
Duration	Length	8	02.02	07.02.02.01.01.03	B.Effort	Duration of SourceClip (in units of edit rate) Distinguished Value = -1 [**RP 210** The absolute duration of a compositional component - e.g. clip, effect, sequence etc in units of edit rate]	
SourcePackageID	Package ID	32	11.01	06.01.01.03.01	Opt	ID of the linked Package as a UMID The value shall be 32 zero valued bytes to terminate the source reference chain [**RP 210** Specifies the reference to a precursor]	

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
SourceTrackID	Track ID (UInt32)	4	11.02	06.01.01.03.02	Req	Track ID of the Track within the linked Package. Zero indicates end of reference chain. [RP 210 Specifies the track within the referenced precursor]	

NOTE – This set is a structural component.

B.17 DM tracks

Descriptive metadata tracks may be one of three types; a standard (timeline) track, an event track or a static track. A timeline DM track represents a contiguous sequence of SourceClips or segments. A timeline DM track shall not have overlapping DM SourceClips or DM segments. An event DM track represents a number of, possibly independent, events occurring in time. The events may occur simultaneously have may be instantaneous or may have a duration. An event DM track may have overlapping DM SourceClips or DM segments and the DM SourceClips or DM segments may be zero duration. A static DM track holds unchanging metadata. A static DM track shall represent a static event and DM SourceClips and DM segments in a static track shall not have a duration property.

NOTE – DM tracks are only present when there is descriptive metadata in the file.

B.17.1 Track (DM)

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Track	Set Key	16		see Table 13	Req	Defines the Track set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [RP 210 The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation identifier [RP 210 Specifies the reference to an overall modification]	
Track ID	UInt32	4	48.01	01.07.01.01	D/Req	ID of the descriptive metadata track in this package (for linking to a SourceTrackID in a SourceClip) [RP 210 Specifies the particular track within a package]	0
Track Number	UInt32	4	48.04	01.04.01.03	Req	Number used to link to the descriptive metadata track in the Essence Container if it exists. Its use is specified in the appropriate Essence Container document [RP 210 Specifies a reference to the codec used to create Essence Data]	0
Track Name	UTF-16 string	var	48.02	01.07.01.02.01	Opt	Human readable name of the track type [RP 210 Specifies the particular track within a package by a name]	"Descriptive Metadata"
Edit Rate	Rational	8	4B.01	05.30.04.05	Req	Edit Rate of Track [RP 210 Specifies the timeline rate in hertz]	
Origin	Position	8	4B.02	07.02.01.03.01.03	Req	An Offset used to resolved timeline references to this track. The start of the track has this timeline value measured in Edit Units. [RP 210 Specifies the point, in edit units, in a track from which relative times are measured.]	
Sequence	StrongRef (Sequence)	16	48.03	06.01.01.04.02.04	Req	Strong reference to Sequence set [RP 210 Specifies a reference to a segment of a track]	

B.17.2 Event track (DM)

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Event Track	Set Key	16		see Table 13	Req	Defines the Track set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation identifier [**RP 210** Specifies the reference to an overall modification]	
Track ID	UInt32	4	48.01	01.07.01.01	D/Req	ID of the descriptive metadata track in this package (for linking to a SourceTrackID in a SourceClip) [**RP 210** Specifies the particular track within a package]	0
Track Number	UInt32	4	48.04	01.04.01.03	Req	Number used to link to the descriptive metadata track in the Essence Container if it exists. Its use is specified in the appropriate Essence Container document [**RP 210** Specifies a reference to the codec used to create Essence Data]	0
Track Name	UTF-16 string	var	48.02	01.07.01.02.01	Opt	Human readable name of the track type [**RP 210** Specifies the particular track within a package by a name]	"Descriptive Metadata"
Event Edit Rate	Rational	8	49.01	05.30.04.02	Req	Edit Rate of Track [**RP 210** Specifies the timeline rate in hertz]	
Event Origin	Position	8	49.02	07.02.01.03.01.0B	Opt	An Offset used to resolved timeline references to this track. The start of the track has this timeline value measured in Edit Units. [**RP 210** Specifies the point, in edit units, in an event track from which relative times are measured.]	0
Sequence	StrongRef (Sequence)	16	48.03	06.01.01.04.02.04	Req	Strong reference to Sequence set [**RP 210** Specifies a reference to a segment of a track]	

B.17.3 Static track (DM)

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Static Track	Set Key	16		see Table 13	Req	Defines the Track set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation identifier [**RP 210** Specifies the reference to an overall modification]	
Track ID	UInt32	4	48.01	01.07.01.01	D/Req	ID of the descriptive metadata track in this package (for linking to a SourceTrackID in a SourceClip) [**RP210** Specifies the particular track within a package]	0

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Track Number	UInt32	4	48.04	01.04.01.03	Req	Number used to link to the descriptive metadata track in the Essence Container if it exists. Its use is specified in the appropriate Essence Container document [RP210 Specifies a reference to the codec used to create Essence Data]	0
Track Name	UTF-16 string	var	48.02	01.07.01.02.01	Opt	Human readable name of the track type [RP210 Specifies the particular track within a package by a name]	"Descriptive Metadata"
Sequence	StrongRef (Sequence)	16	48.03	06.01.01.04.02.04	Req	Strong reference to Sequence set [RP210 Specifies a reference to a segment of a track]	

B.18 Sequence (DM)

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Sequence	Set Key	16		see Table 13	Req	Defines the Sequence set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [RP 210 The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [RP 210 Specifies the reference to an overall modification]	
Data Definition	UL	16	02.01	04.07.01	Req	Specifies the data type of this set. (value= Descriptive Metadata - see SMPTE RP 224) [RP 210 Specifies the basic essence type of a component]	
Duration	Length	8	02.02	07.02.02.01.01.03	B.Effort **	Duration of Sequence (in units of edit rate) Distinguished Value = -1 [RP 210 The absolute duration of a compositional component - e.g. clip, effect, sequence etc in units of edit rate] ** Note: Duration is Best Effort if the Sequence is in a standard (timeline) Track, is optional in an Event Track and shall be omitted in a Static Track.	
Structural Components	StrongRefArray (Structural Components)	8+16n	10.01	06.01.01.04.06.09	Req	Ordered array of strong references to Descriptive Metadata Structural Components [RP 210 Specifies a vector of references to the clips and transitions in the sequence]	

B.19 DM segment

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
DM Segment	Set Key	16		see Table 13	Req	Defines the Descriptive Metadata Segment set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [**RP 210** Specifies the reference to an overall modification]	
Data Definition	UL	16	02.01	04.07.01	Req	Specifies the data type of this set. (value= Descriptive Metadata - see SMPTE RP224) [**RP 210** Specifies the basic essence type of a component]	
Event Start Position	Position	8	06.01	07.02.01.03.03.03	Req **	Offset into the descriptive metadata track in edit units ** Note: Event Start Position is Required if the DM Segment is on an Event Track, but shall be omitted if the DM Segment is on a standard (timeline) Track or a Static Track.	
Duration	Length	8	02.02	07.02.02.01.01.03	B.Effort **	Duration of segment (in units of edit rate). At least as long as the total timespan of this DM Framework, if applicable. Distinguished Value = -1 [**RP 210** The absolute duration of a compositional component - e.g. clip, effect, sequence etc in units of edit rate] ** Note: Duration is Best Effort if the DM Segment is in a standard (timeline) Track, is optional in an Event Track and shall be omitted in a Static Track.	
Event Comment	UTF-16 String	Var	06.02	05.30.04.04.01	Opt	Description of the Descriptive Metadata Framework [**RP 210** User-provided Comment Text for an event]	
Track IDs (DM Segment)	TrackID Batch	8+ 4*n	61.02	01.07.01.05	D/req	Specifies an unordered list of track ID values that identify the tracks in this Package to which this DM Framework refers (if omitted, refers to all essence tracks) [**RP 210** Specifies an unordered list of track ID values that identify descriptive metadata tracks by containment]	
DM Framework	StrongRef (DMFramework)	16	61.01	06.01.01.04.02.0C	D/req	Strong Reference to the Descriptive Metadata Framework [**RP 210** Strong Reference to the Descriptive Metadata Framework]	

NOTE – This set is a structural component.

NOTE – The descriptive metadata segment is also a subclass of the AAF comment marker class. It has incorporated within it a DM track IDs item which specifies the target tracks of the descriptive metadata. It includes a strong reference to a DM framework item to contain the descriptive metadata for this segment.

B.20 DM SourceClip

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
DM SourceClip	Set Key	16		see Table 13	Req	Defines the Descriptive Metadata SourceClip set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [**RP 210** Specifies the reference to an overall modification]	
Data Definition	UL	16	02.01	04.07.01	Req	Specifies the data type of this set. (value= Descriptive Metadata - see SMPTE RP224) [**RP 210** Specifies the basic essence type of a component]	
Start Position	Position	8	12.01	07.02.01.03.01.04	Req **	Offset into the descriptive metadata track in edit units [**RP 210** The relative start time from an origin within the track of a clip, expressed in edit units] ** Note: Start Position is Required if the DM SourceClip references a standard (timeline) Track or an Event Track, but shall be omitted if it references a Static Track.	
Duration	Length	8	02.02	07.02.02.01.01.03	B.Effort **	Duration of SourceClip (in units of edit rate) Distinguished Value = -1 [**RP 210** The absolute duration of a compositional component - e.g. clip, effect, sequence etc in units of edit rate] ** Note: Duration is Best Effort if the DM SourceClip is in a standard (timeline) Track, is optional in an Event Track and shall be omitted in a Static Track.	
SourcePackageID	Package ID	32	11.01	06.01.01.03.01	Req	ID of the linked Package as a UMID The value shall be 32 zero valued bytes to terminate the source reference chain [**RP 210** Specifies the reference to a precursor]	
SourceTrackID	Track ID (UInt32)	4	11.02	06.01.01.03.02	D/req	Track ID of the Track within the linked Package from which the Descriptive Metadata is obtained. Zero indicates end of reference chain. [**RP 210** Specifies the track within the referenced precursor]	
Track IDs (DM SourceClip)	TrackID Batch	8+ 4*n	61.03	01.07.01.06	D/req	Specifies an unordered list of track ID values that identify the target tracks in this Package to which the referenced Descriptive Metadata refers (if omitted, refers to all essence tracks) [**RP 210** Specifies an unordered list of track ID values that identify metadata source tracks by reference]	

NOTE – This set is a structural component.
This is a SourceClip which can be strong referenced by sequence (descriptive metadata) in order to provide a linking/derivation mechanism between descriptive metadata in material package, file package and source packages. A DM SourceClip shall only be referenced by a sequence in a (timeline) track.

NOTE – The DM SourceClip is used in the same way as a SourceClip (see figure 11). The start position and duration specify the portion of the source metadata which is relevant for this SourceClip. The source metadata may be a DM segment in any of the tracks listed in B.17

Annex C (normative)
Specifications for the package used in MXF

There are certain restrictions placed on the different MXF packages which are given below.

C.1 Material package

The material package has the structure detailed in section 8.5.3. The following extra data values are defined:

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Material Package	Set Key	16		see Table 13	Req	Defines the Material Package set	

In the time code track within this package, there may be no time code sequence set. In this case, the sequence property of the track (time code) shall directly reference the appropriate time code structural component. The time code of the material package is by definition continuous (un-segmented). This is unlike the other tracks, all of which may be segmented.

Because a material package describes the output timeline and not stored essence, the track number(s) within a material package cannot be used to link to essence. This is outlined in sections 8.4 and 8.5. An MXF encoder generating a new file should set the track number(s) to 0.

C.2 Source package

The source package has the structure detailed in section 8.5.6. The term "source package" is used in this specification to mean "file package or physical package". The following extra data values are defined:

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Source Package	Set Key	16		see Table 13	Req	Defines the Source Package set	
Descriptor	StrongRef (EssenceDescriptor)	16	47.01	06.01.01.04.02.03	see C.3 and C.4	A strong reference to the Generic Descriptor (May be a Multiple Descriptor) [RP 210 Specifies a reference to a format descriptor for the essence]	

C.3 File package

The file package has the structure detailed in annex C.2. A file package shall only contain essence if it is referenced by a material package; i.e., a top-level file package. All other file packages (lower-level file packages) contain historical annotation metadata only. A file package shall be identified by having zero or one file descriptors and zero physical descriptors.

NOTE – In AAF, this package is referred to as a file source package.

C.4 Physical package

The physical package has the structure detailed in annex C.2. A physical package shall not contain essence. A physical package shall be identified by having zero file descriptors and one or more physical descriptors.

NOTE – In AAF, this package is referred to as a physical source package.

C.5 Package hierarchy in MXF

Figure C.1 shows the naming conventions used in MXF. Note that for the generalized operational patterns a material package always references a top-level file package (specialized operational patterns may permit a material package to reference a physical package). A file package references a lower-level source package. This may be a file package or a physical package. In MXF only a top-level file package may contain essence data. All lower-level file packages shall be used for the carriage of historical (derivation) metadata.

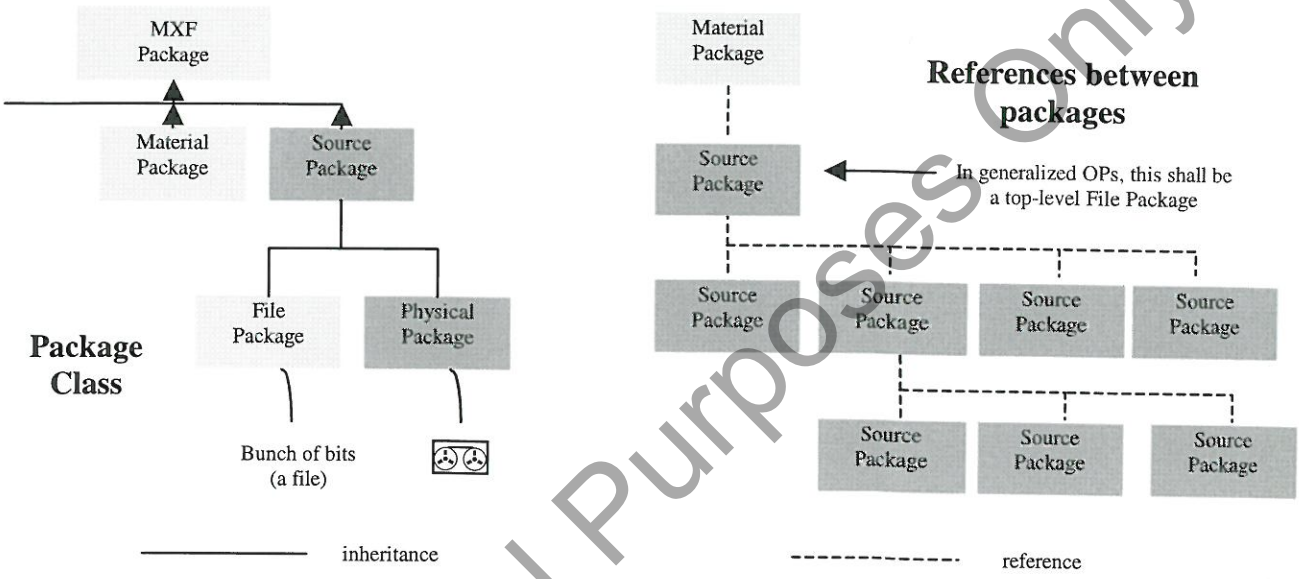

Figure C.1 – Package class model and how packages reference each other

Annex D (normative)
Specifications for file-based descriptors used in MXF

NOTE – All file-based descriptor sets are a subclass of file descriptor and inherit the properties defined in that abstract set.

D.1 File descriptor

NOTE – This descriptor is the set from which many descriptor sets are derived. It in turn is derived from the generic descriptor in section 9.5. Other descriptors are defined in the individual essence container documents where those in this annex do not suffice. The properties below shall be common to all Descriptor sets derived from file descriptor.

It is highly desirable to include a file descriptor for each essence track in the file package. Therefore, MXF encoders should encode a file descriptor for each track in the file package. It is recognized that some MXF creation applications may be unable to fully populate the file descriptors (or their subclasses).

IMPORTANT NOTE – The required column in the tables below indicates the status of descriptor items. Some devices implementing certain container types may be unable to fill in the best effort items in the essence descriptor at the point of file creation. For this reason the "incomplete" partition status exists (section 5.2.3). This mechanism shall only be used for best effort metadata items. If the value of a best effort metadata item is not known by the MXF encoder then the distinguished value shall be used.

The file descriptor contains a TrackID, but no PackageID. To be consistent with the AAF source reference rules, this means that the TrackID always refers to the current package. In MXF, a file descriptor (or one of its subclasses) inside the top-level file package describes how the EssenceContainer data **is** coded. A file descriptor (or one of its subclasses) within a lower-level source package describes how a previous version of the essence **was** coded.

The normative definition of item ULs is given by SMPTE RP 210. The data type of certain item ULs is itself a UL which is an enumeration of known values for the Item. The SMPTE registry is the normative reference for these values which are listed in SMPTE RP 224.

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
File Descriptor	Set Key	16		see Table 13	Req	Defines the File Descriptor set	
Length	BER Length	var			Req	Set length (see 8.3)	
Instance UID	UUID	16	3C.0A	01.01.15.02	Req	Unique ID of this instance [**RP 210** The ISO/IEC 11578 (Annex A) 16 byte Globally Unique Identifier]	
Generation UID	UUID	16	01.02	05.20.07.01.08	Opt	Generation Identifier [**RP 210** Specifies the reference to an overall modification]	
Linked Track ID	UInt32	4	30.06	06.01.01.03.05	Opt	Link to (i.e. value of) the Track ID of the Track in this Package to which the Descriptor applies. [**RP 210** Link to (i.e. value of) the Track ID of the Track in this Package to which the Essence Descriptor applies.]	
SampleRate	Rational	8	30.01	04.06.01.01	Req	The field or frame rate of Essence Container (not the essence (pixel) sampling clock rate) [**RP 210** Specifies the number of addressable elements of essence data per second]	
Container Duration	Length	8	30.02	04.06.01.02	Opt	Duration of Essence Container (measured in Edit Units) A file writer should write the best value it can write. If it cannot be completed, the Item should be omitted. [**RP 210** Specifies the number of addressable elements of essence data]	

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Essence Container	UL	16	30.04	06.01.01.04.01.02	Req	The UL identifying the Essence Container described by this descriptor. Listed in SMPTE RP 224 [**RP 210** Specifies a reference to the format of Container of Essence Data]	
Codec	UL	16	30.05	06.01.01.04.01.03	Opt	UL to identify a codec compatible with this Essence Container. Listed in SMPTE RP 224 [**RP 210** Specifies a reference to the codec used to create Essence Data]	
Locators	StrongRefArray (Locators)	8+ 16n	2F.01	06.01.01.04.06.03	Opt	Ordered array of strong references to Locator sets If present, essence may be located external to the file. If there is more than one locator set an MXF Decoder shall use them in the order specified. [**RP 210** Specifies a vector of references to essence locators]	

NOTE – Codec UL is intended to identify the codec tool (as hardware/software) whereas the essence container identifies the essence container itself. Note that the codec tool might be multiformat (e.g., MPEG/JPEG/DV). The EssenceContainers in the partition pack is a list of all the essence container as defined in section 6.1.

D.2 Picture essence descriptors

D.2.1 Generic picture essence descriptor

The generic picture essence descriptor is designed to provide generic parametric information which describes the picture.

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Picture Essence Descriptor	Set Key	16		see Table 13	Req	Defines the Picture Essence Descriptor set	
Length	BER Length	var			Req	Set length (see 8.3)	
All items in D.1	see D.1		D.1	see D.1	D.1	All items from D.1 (excluding the Set Key and Length)	
Signal Standard	Enum	1	32.15	04.05.01.13	Opt	Underlying Signal Standard (see E.2.3) [**RP 210** Underlying Signal Standard]	1
Frame Layout	Uint8	1	32.0C	04.01.03.01.04	B.Effort	Interlace or progressive layout (see E.2.2) Distinguished Value = 255 (0= FULL_FRAME, 1= SEPARATE_FIELDS, 2= SINGLE_FIELD, 3= MIXED_FIELDS, 4=SEGMENTED_FRAME) [**RP 210** Specifies frame layout (interlaced, single frame, full frame, etc.)]	
Stored Width	Uint32	4	32.03	04.01.05.02.02	B.Effort	Horizontal Size of stored picture (see E.2.9) Distinguished Value = 0 [**RP210** Specifies the integer width of the stored image in pixels]	
Stored Height	Uint32	4	32.02	04.01.05.02.01	B.Effort	Vertical Field Size of stored picture (see E.2.10) Distinguished Value = 0 [**RP210** Specifies the integer height of the stored image in pixels]	
StoredF2Offset	Int32	4	32.16	04.01.03.02.08	Opt	Topness Adjustment for stored picture (see E.2.22) [**RP210** Topness Adjustment for stored picture]	0
SampledWidth	Uint32	4	32.05	04.01.05.01.08	Opt	Sampled width supplied to codec (see E.2.11) [**RP210** Specifies the integer width of the sampled	

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
						image in pixels]	
Sampled Height	Uint32	4	32.04	04.01.05.01.07	Opt	Sampled height supplied to codec (see E.2.12) [**RP210** Specifies the integer height of the sampled image in pixels]	
SampledXOffset	Int32	4	32.06	04.01.05.01.09	Opt	Offset from stored to sampled width (see E.2.13) (positive means additional stored pixels) [**RP 210** Specifies the X offset of the sampled image relative to the stored image in pixels]	0
SampledYOffset	Int32	4	32.07	04.01.05.01.0A	Opt	Offset from stored to sampled height (see E.2.14) (positive means additional stored lines) [**RP 210** Specifies the Y offset of the sampled image relative to the stored image in pixels]	0
DisplayHeight	Uint32	4	32.08	04.01.05.01.0B	Opt	Displayed Height placed in Production Aperture (see E.2.18) [**RP 210** Specifies the height of the presented image relative to the sampled image in pixels]	
DisplayWidth	Uint32	4	32.09	04.01.05.01.0C	Opt	Displayed Width placed in Production Aperture (see E.2.17) [**RP 210** Specifies the width of the presented image in pixels]	
DisplayXOffset	Int32	4	32.0A	04.01.05.01.0D	Opt	Offset from Sampled to Display Width (see E.2.19) [**RP 210** Specifies the X offset of the presented image relative to the sampled image in pixels]	
DisplayYOffset	Int32	4	32.0B	04.01.05.01.0E	Opt	Offset from Sampled to Display Height (see E.2.20) [**RP 210** Specifies the Y offset of the presented image relative to the sampled image in pixels]	
DisplayF2Offset	Int32	4	32.17	04.01.03.02.07	Opt	Topness Adjustment for displayed picture (see E.2.21) [**RP210** Topness Adjustment for displayed picture]	0
Aspect Ratio	Rational	8	32.0E	04.01.01.01.01	B.Effort	Specifies the horizontal to vertical aspect ratio of the whole image as it is to be presented to avoid geometric distortion (and hence includes any black edges) e.g. {4,3} or {16,9} (see E.2.7) Distinguished Value = {0,0} [**RP 21 0** Specifies the horizontal to vertical aspect ratio of the whole image as it is to be presented to avoid geometric distortion and hence including any black edges.]	
Active Format Descriptor	UInt8	1	32.18	04.01.03.02.09	Opt	Specifies the intended framing of the content within the displayed image (4:3 in 16:9 etc.) (see E.2.8) [**RP 210** Specifies the intended framing of the content within the displayed image (4:3 in 16:9 etc.)]	
Video Line Map	Array of Int32	8+8	32.0D	04.01.03.02.05	B.Effort	First active line in each field e.g. {16,278} (see E.2.15) Distinguished Value = {0,0} [**RP 210** Specifies the line numbers of the two top lines of the active picture]	
Alpha Transparency	UInt8	1	32.0F	05.20.01.02	Opt	Is Alpha Inverted ? (see E.2.26) [**RP 210** Zero if the minimum value of an alpha sample specifies full transparency and the maximum	0

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
						value specifies full opacity, one if vice versa.]	
Capture Gamma	UL	16	32.10	04.01.02.01.01.01.02	Opt	Registered UL of known Gamma (see E.2.27) [**RP 210** Specifies the non-linear relationship between linear scene light levels and amplitude-compressed video signal levels at signal origination.]	
Image Alignment Offset	Uint32	4	32.11	04.18.01.01	Opt	Byte Boundary alignment required for Low Level Essence Storage (see E.2.28) [**RP 210** Specifies number of bytes to align the start of an image with a defined memory boundary]	0
Image Start Offset	Uint32	4	32.13	04.18.01.02	Opt	Unused bytes before start of stored data (see E.2.29) [**RP 210** Specifies bytes of fill before start of field]	
Image End Offset	Uint32	4	32.14	04.18.01.03	Opt	Unused bytes after end of stored data (see E.2.30) [**RP210** Specifies bytes of fill after end of field]	
FieldDominance	Uint8	1	32.12	04.01.03.01.06	Opt	The number of the field which is considered temporally to come first. (see E.2.25) [**RP210** Specifies whether the first frame of picture is field 1 or field 2]	
Picture Essence Coding	UL	16	32.01	04.01.06.01	D/Req	UL identifying the Picture Compression Scheme (see E.2.31) [**RP210** Specifies the Compression scheme used]	

D.2.2 CDCI (Color-difference component image) picture essence descriptor

NOTE – This is a subclass of the generic picture essence descriptor above. It has all the above items with the same required / optional status with the addition of the new items below. It is intended to describe imagery comprising interleaved luminance and chrominance samples.

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
CDCI Essence Descriptor	Set Key	16		See Table 13	Req	Defines the CDCI Picture Essence Descriptor set	
Length	BER Length	var			Req	Set length (see 8.3)	
All items in D.2.1	see D.2.1		D.2.1	see D.2.1	D.2.1	All items from D.2.1 (excluding the Set Key and Length)	
Component Depth	UInt32	4	33.01	04.01.05.03.0A	B.Effort	Number of active bits per sample e.g. 8, 10, 16 (see E.2.32) Distinguished Value = 0 [**RP 210** Specifies the component width before subsampling is applied]	
Horizontal Subsampling	UInt32	4	33.02	04.01.05.01.05	B.Effort	Specifies the H color subsampling (see E.2.33) Distinguished Value = 0 [**RP 210** Specifies ratio of luminance subsampling to chrominance subsampling in horizontal direction]	
Vertical Subsampling	UInt32	4	33.08	04.01.05.01.10	Opt	Specifies the V color subsampling (see E.2.34) [**RP 210** Specifies ratio of luminance subsampling to chrominance subsampling in vertical direction]	
Color Siting	UInt8	1	33.03	04.01.05.01.06	Opt	Enumerated value describing color siting (see .2.35 [**RP 210** Specifies how to compute subsampled chrominance values]	

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
ReversedByteOrder	Boolean	1	33.0B	03.01.02.01.0A	Opt	a FALSE value denotes Chroma followed by Luma pixels according to ITU Rec.601 (see E.2.41) [**RP 210** Specifies whether the luma and chroma sampling order conforms to ITU-R BT.601. Value will be zero if the byte order conforms, non-zero if the luminance sample precedes the chroma.]	
PaddingBits	Int16	2	33.07	04.18.01.04	Opt	Bits to round up each pixel to stored size (see E.2.36) [**RP 210** Specifies the number of bits to pad each pixel so that the next pixel starts on a defined boundary]	
Alpha Sample Depth	UInt32	4	33.09	04.01.05.03.07	Opt	Number of bits per alpha sample (see E.2.37) [**RP 210** Specifies the number of bits in the alpha signal.]	
Black Ref Level	Uint32	4	33.04	04.01.05.03.03	Opt	e.g. 16 or 64 (8 or 10-bits) (see E.2.38) [**RP 210** Specifies digital luminance associated with black]	
White Ref level	Uint32	4	33.05	04.01.05.03.04	Opt	e.g. 235 or 940 (8 or 10 bits) (see E.2.39) [**RP 210** Specifies digital luminance associated with white]	
Color Range	Uint32	4	33.06	04.01.05.03.05	Opt	e.g. 225 or 897 (8 or 10 bits) (see E.2.40) [**RP 210** Specifies the range of the color levels.]	

D.2.3 RGBA (Red Green Blue Alpha) picture essence descriptor

NOTE – This is a subclass of the generic picture essence descriptor above. It has all the above items with the same required/optional status with the addition of the new items below. It is intended to describe color component imagery with transparency.

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
RGBA Essence Descriptor	Set Key	16		See Table 13	Req	Defines the RGBA Essence Descriptor set	
Length	BER Length	var			Req	Set length (see 8.3)	
All items in D.2.1	see D.2.1		D.2.1	see D.2.1	D.2.1	All items from D.2.1 (excluding the Set Key and Length)	
Component Max Ref	Uint32	4	34.06	04.01.05.03.0B	Opt	Maximum value for RGB components e.g. 235 or 940 (8 or 10 bits) [**RP 210** Maximum value for RGB components e.g. 235 or 940 (8 or 10 bits)]	255
Component Min Ref	Uint32	4	34.07	04.01.05.03.0C	Opt	Minimum value for RGB components e.g. 16 or 64 (8 or 10-bits) [**RP 210** Minimum value for RGB components e.g. 16 or 64 (8 or 10-bits)]	0
Alpha Max Ref	Uint32	4	34.08	04.01.05.03.0D	Opt	Maximum value for alpha component e.g. 235 or 940 (8 or 10 bits) [**RP 210** Maximum value for alpha component e.g. 235 or 940 (8 or 10 bits)]	255
Alpha Min Ref	Uint32	4	34.09	04.01.05.03.0E	Opt	Minimum value for alpha components e.g. 16 or 64 (8 or 10-bits) [**RP 210** Minimum value for alpha components e.g. 16 or 64 (8 or 10-bits)]	0

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
ScanningDirection	Orientation	1	34.05	04.01.04.04.01	Opt	Enumerated Scanning Direction (see E.2.45) [RP 210 Enumerated Scanning Direction]	
PixelLayout	RGBALayout	var	34.01	04.01.05.03.06	B.Effort	(see E.2.42) Distinguished Value = 0 in each byte of the RGBLayout array [RP210 Specifies pixel quantization and order as a data structure.]	
Palette	DataValue	var	34.03	04.01.05.03.08	Opt	(see E.2.43) [RP210 Specifies, as a single string, the fixed length values of each color in the palette used.]	
PaletteLayout	RGBALayout	var	34.04	04.01.05.03.09	Opt	(see E.2.44) [RP210 Specifies pixel quantization and order in the palette as a data structure.]	

D.3 Generic sound essence descriptor

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Sound Essence Descriptor	Set Key	16		See Table 13	Req	Defines the Sound Essence Descriptor set	
Length	BER Length	var			Req	Set length (see 8.3)	
All items in D.1	see D.1		D.1	see D.1	D.1	All items from D.1 (excluding the Set Key and Length)	
Audio sampling rate	Rational	8	3D.03	04.02.03.01.01.01	B.Effort	Sampling rate of the audio essence Distinguished Value = 0 [RP 210 The reference sampling clock frequency as a rational number]	{48000,1}
Locked/Unlocked	Boolean	1	3D.02	04.02.03.01.04	D/req	Boolean indicating that the number of samples per frame is locked or unlocked. [RP 210 TRUE if number of samples per frame is locked to video]	
Audio Ref Level	Int8	1	3D.04	04.02.01.01.03	Opt	Audio reference level which gives the number of dBm for 0VU. [RP 210 Number of Dbm for 0VU]	
Electro-Spatial Formulation	Uint8 (Enum)	1	3D.05	04.02.01.01.01	Opt	E.g. mono, dual mono, stereo, A,B etc [RP 210 Mono, Dual mono, Stereo A+B, Stereo M&S, Dolby surround, MPEG BC/NBC etc] 0 = two-channel mode default 1 = two-channel mode 2 = single channel mode 3 = primary/secondary mode 4 = stereophonic mode 7 = single channel, double frequency mode carried on 2 sub-frames 8 = stereo left channel, double frequency mode carried on 2 sub-frames 9 = stereo right channel, double frequency mode carried on 2 sub-frames 15 = multi-channel mode default (>2 channels) (informative note: these values are identical to values defined in AES-3 (R1997)	
ChannelCount	Uint32	4	3D.07	04.02.01.01.04	B.Effort	Number of Sound Channels Distinguished Value = 0	

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
						[**RP 210** The number of channels represented in the waveform data.]	
Quantization bits	UInt32	4	3D.01	04.02.03.03.04	B.Effort	Number of quantization bits Distinguished Value = 0 [**RP 210** The maximum number of significant bits for the value without compression.]	
Dial Norm	Int8	1	3D.0C	04.02.07.01	Opt	Gain to be applied to normalize perceived loudness of the clip, defined by ITU-R BS.1196 (1995) (1dB per step) [**RP 210** Gain to be applied to normalize perceived loudness of the clip] (Defined by ITU-R BS.1196 (1995) (1dB per step)	
Sound Essence Compression	UL	16	3D.06	04.02.04.02	D/req	UL identifying the Sound Compression Scheme [**RP 210** Specifies the Compression scheme used]	

D.4 Generic data essence descriptor

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Data Essence Descriptor	Set Key	16		See Table 13	Req	Defines the Data Essence Descriptor set	
Length	BER Length	Var			Req	Set length (see 8.3)	
All items in D.1	see D.1		D.1	see D.1	D.1	All items from D.1 (excluding the Set Key and Length)	
Data Essence Coding	UL	16	3E.01	04.03.03.02	D/req	Specifies the data essence coding type Values are listed in SMPTE RP 224 [**RP 210** Specifies the Coding scheme used]	

NOTE – It is anticipated that many data descriptors will need to be created for MXF. These are likely to be used to describe data buried within other essence types (e.g., time code, teletext, subtitles, captioning) as well as new data types that MXF will carry in the future. A document which defines the container specification for the data essence will define or reference data descriptors and the appropriate data definition term to be used in the sequence and segment.

D.5 Multiple descriptor

Item Name	Type	Len	Local Tag	UL Designator	Req ?	Meaning	Default
Multiple Descriptor	Set Key	16		See Table 13	Req	Defines the Multiple Descriptor set	
Length	BER Length	var			Req	Set length (see 8.3)	
All items in D.1	see D.1		D.1	see D.1	D.1	All items from D.1 (excluding the Set Key and Length)	
Sub Descriptor UIDs	StrongRefArray (File Descriptors)	8+ 16n	3F.01	06.01.01.04.06.0B	Req	Ordered array of strong references to File Descriptor sets (1 per interleaved item within the Essence Container The order of the descriptors should be the same as the order of the tracks that they describe) [**RP 210** Specifies a vector of an ordered set of references to File Descriptor sets]	

NOTE – The multiple descriptor defines a list of descriptors which describe an Interleaved essence container. For example, an essence container which has video data interleaved with teletext data may use a multiple descriptor to describe each essence type and the linked track ID of the referenced file descriptor shall indicate the track to which each file descriptor applies. The linked track ID property should not be present on a multiple descriptor because a multiple descriptor describes multiple tracks. If the essence codec Item is present, then it shall be a codec appropriate for the interleaved container.

Annex E (normative)
Picture essence descriptor items

E.1 Stored data

The stored data consists of a number of start bytes, followed by a stored rectangle of image data, followed by end bytes. The mapping between pixels stored in the stored rectangle and those present in the digital video signal is described in this section. This is illustrated in figure E.1

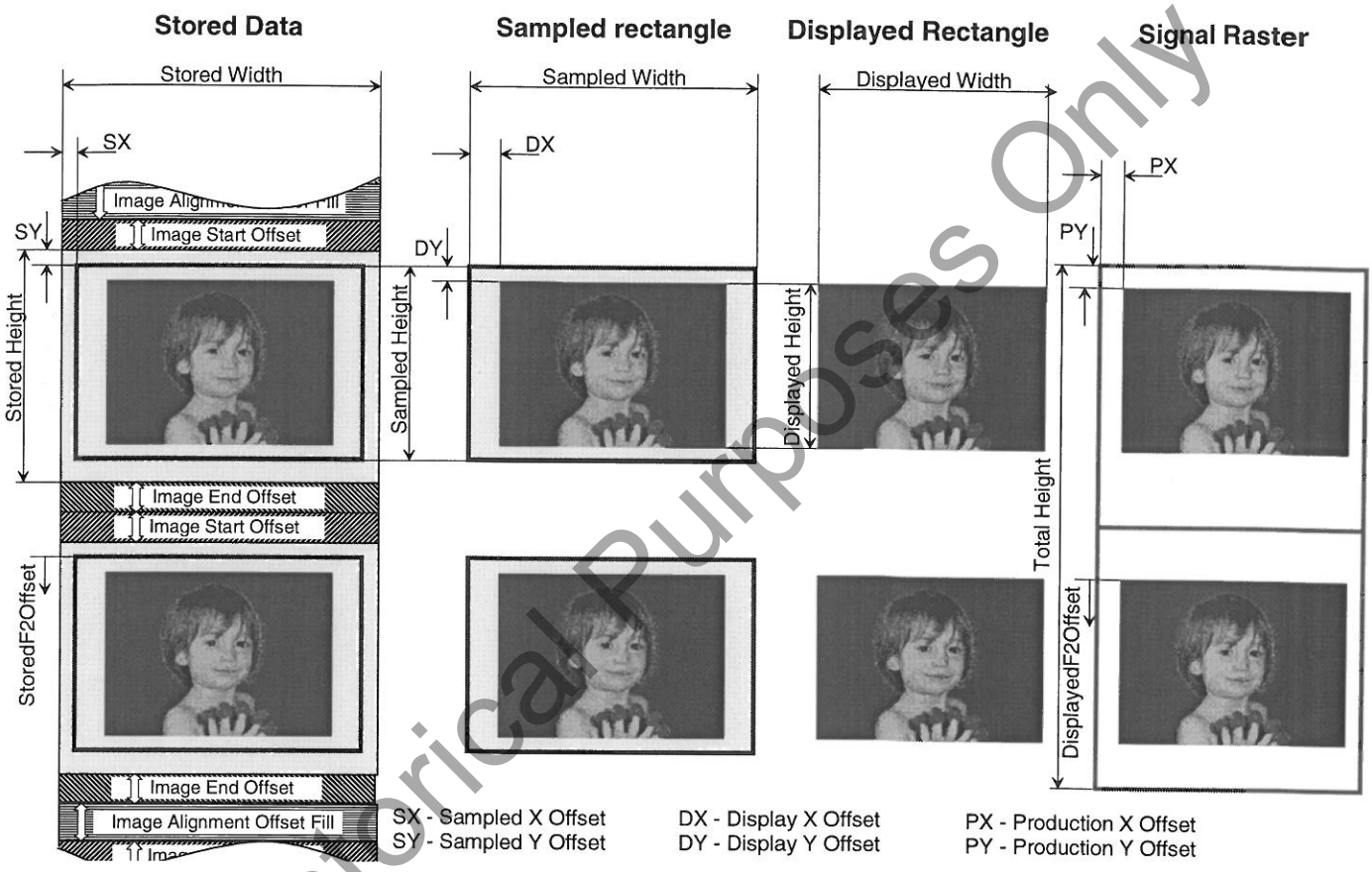

Figure E.1 – Stored, sampled, displayed, rectangles and signal raster

E.1.1 Data storage, processing, display and signal raster model

This specification describes image geometries from three perspectives, the stored rectangle, the sampled rectangle and the displayed rectangle. The specification defines the image properties, plus properties to describe the mapping of image rectangles between each view and to the actual signal raster.

E.1.2 Stored data and stored rectangle

The stored data comprises the entire data region corresponding to the stored rectangle for a single frame or field of the image plus any start or end data bytes in the byte stream. The stored rectangle corresponds to a rectangle of stored pixels described by the stored width and stored height properties. The stored rectangle byte count is defined by its width and height dimension properties multiplied by the number of bytes per pixel.

The stored data may include bytes that are not derived from, and would not usually be translated back to, signal data. This extra data is the start fill and end fill region. The sizes are defined by the image start offset and image end offset properties.

The stored data may be aligned to a particular storage boundary, defined by the image alignment offset.

NOTE – In cases where compression systems are in use, the stored rectangle corresponds to the data which is passed to the compressor and received from the decompressor. Since compression systems typically constrain the sizes of macroblocks, the stored rectangle may be larger than the sampled rectangle and the display rectangle. Also, in these cases, the definition of the macroblocks may dictate an offset from stored to sampled rectangle which is not equal in field 1 and field 2; this is described by the optional StoredF2Offset property below. In addition, when the stored rectangle comprises merged fields, the sample rate property will be set to the frame rate according to E.2.1.

E.1.3 Sampled rectangle

The sampled rectangle is the rectangular region corresponding to the digital data pixels derived from an image source. This includes the image and any auxiliary information actually sampled from the analog or digital source.

Although typically the sampled rectangle data is a subset of the stored data, it is possible that the sampled area may not be a subset of the stored data. The sampled rectangle may overlap or encapsulate the stored rectangle.

The sampled rectangle is defined by its width and height properties in pixels. These properties default to the same value as the stored width and height.

The mapping from the stored rectangle to the sampled rectangle is defined by the sampled X offset (SX) and sampled Y (SY) offset properties, which give the zero-based coordinates of the sampled rectangle relative to the upper left corner of the stored rectangle. These properties have a default value of 0. They may be positive or negative.

NOTE – The example in figure E.1 shows positive values for the SX and SY offsets.

E.1.4 Display rectangle

The display rectangle is the rectangular region which is actually visible in the signal raster.

The display rectangle directly maps to the SMPTE RP 187 production aperture. The display rectangle is defined by its width and height properties in pixels. These properties default to the same value as the sampled width and sampled height. Their values shall not be greater than the sampled width and sampled height.

The mapping from the sampled rectangle to the display rectangle is defined by the display X offset (DX) and display Y offset (DY) properties, which give the zero-based coordinates of the display rectangle relative to the upper left corner of the sampled rectangle. These properties have a default value of 0. Their values shall not be negative.

E.1.5 Signal raster

Signal raster is specified by raster standards such as by ANSI/SMPTE 125M, SMPTE 274M, SMPTE 296M, ITU-R BT470.4, and ITU-R BT601.5. Companion standards such as SMPTE RP 187 and ITU-R BT656 give additional description for some cases.

Images may be scanned in progressive or interlaced mode. This is specified by the frame layout property.

The dimensions of the raster itself are defined by the total width and total height properties.

Different standards define different line numbering schemes. The cardinal line numbers were initially specified for analogue standards, such as ITU-R BT470. The definitions of fields and line numbers in digital images are not identical to the definitions for analogue images. The actual numbers used by this specification are those defined for the digital standards (ITU-R BT.656, SMPTE 274M, SMPTE 296M, SMPTE 293M).

The raster provides the framework in which the display rectangle (production aperture) is positioned. The vertical mapping of the display rectangle into the raster is defined in three stages:

- first, by the VideoLineMap property, which gives the cardinal line numbers of the first sampled line in each field;
- second, by the DisplayYOffset property, which gives the number of lines of sampled data which must be blanked at the start of each field;
- third, by an optional DisplayF2Offset property, which adjusts the number of lines which are blanked in field 2 relative to the number blanked in field 1.

The horizontal mapping of the display rectangle into the raster is defined by blanking the pixels to the left of DisplayXOffset and to the right of DisplayXOffset+DisplayWidth-1.

The net offset from the raster to the display rectangle (production aperture) is shown in figure E.1 as PX and PY. Note that PY may have different values for fields 1 and 2 of an interlaced signal. PX is zero for all known digital rasters.

E.1.6 Sampling

This section describes the mapping of image samples onto pixels, and the derivation of the number of bytes per pixel.

E.1.6.1 Color-difference component sampling (CDCI sampling)

Color-difference component sampling is the method used by ITU-R BT601.5, ANSI/SMPTE 125M, SMPTE 274M, SMPTE 296M, and other standards.

In CDCI sampling, each pixel is composed of a sequence of chroma, luminance and, optionally alpha (transparency) samples. The number of bits used for the samples may vary, but is typically 8 or 10. The order of the components is specified by the pixel layout item. The optional property ReversedByteOrder may be used to specify images stored luminance first.

For example, the number of bytes per pixel in a 4:2:2 system may be derived from the following equation using the properties in 0:

Bytes per pixel = (2*ComponentDepth+PaddingBits+AlphaSampleDepth) / 8

Bytes per pixel shall be an integer (more complex packing schemes can defined using the RGBA descriptor).

Most commonly this value is 2.

The chroma sample is derived from color difference signals, sub-sampled by various factors horizontally and vertically, and filtered. Several variants of the notional siting of the chroma samples relative to luminance are in use.

E.1.6.2 Red green blue alpha sampling

Red green blue alpha (RGBA) sampling is the method used in computer graphics and display.

In RGBA sampling, each pixel is composed of co-sited samples from red, green, blue and optionally alpha channels.

Various combinations of bits per sample, ordering of the components within the pixel, and padding are in use.

A variant of this sampling method is to construct a palette of some number of distinct color values, and then to use the index of the closest color within the palette as the value of each pixel.

The number of bytes per pixel must be derived by analysis of the Pixel Layout property on a case by case basis.

E.2 Property definitions

This section specifies the properties which characterize images. Some of these properties are classed as inferred or derived:

Inferred properties are those whose values may be easily inferred from the underlying standard (as given by the signal standard property or by calculation from the values of other properties.

Derived properties are useful parameters which can be derived from other properties.

E.2.1 Sample rate and edit rate

Sample rate is used in descriptors and edit rate is used in tracks. Sample rate is the rate of stored rectangles expressed as a rational number. This is typically either the rate of fields or frames in the image. A sample unit is used in the text below to represent 1/sample rate; i.e., the time duration of the stored rectangle. In the descriptors defined in annex D, the sample rate should be a frame.

Edit Rate is the desired editing rate of the of the image data. An edit unit (1/edit rate) is typically 1 image, but may be larger in some applications

NOTES

1. For example, the sampled image may be field based, but the editing may be frame based. In this case the edit rate will be half the sample rate.

2. Together with the frame layout property, the sample rate property provides the information from which SMPTE 352M byte 2 may be derived.

E.2.2 Frame layout

Images may be scanned progressively or in one of several interlaced methods. Frame layout property is as follows:

FULL_FRAME (0) – a progressive lattice from top to bottom, stored in progressive line order 1,2,3,4,5,6... Example: "480P59.94". The duration of a sampled rectangle is a frame.

SEPARATE_FIELDS (1) – an interlaced lattice divided into two fields, stored as two fields 1,3,5,... and 2,4,6... Field 1 scans alternate lines from top to bottom, field 2 scans the intervening lines. The second field is scanned at a later time than the first field (one field later). Note that different signal standards may be different topness (see E.2.16) and dominance (see E.2.25). Examples: NTSC, ANSI/SMPTE 125M. The duration of a sampled rectangle is a field.

SINGLE_FIELD (2) – an interlaced lattice as for SEPARATE_FIELDS above, except that only one field is scanned and retained in the stored data, as 1,3,5,... or 2,4,6,... or (1+2),(3+4),(5+6),... For display, the second field is derived by line replication or interpolation. There are no examples of SINGLE_FIELD in broadcast use; however, this type of sub-sampling is often used as a simple compression for index frames. The duration of a sampled rectangle is a frame.

MIXED_FIELDS (3) – an interlaced lattice as for SEPARATE_FIELDS above, stored as a single matrix of interleaved lines 1,2,3,4,5,6,... It is not common to use MIXED_FIELDS in broadcast; however, intermediate in-memory data structures sometimes use this format. The duration of a sampled rectangle is a frame.

SEGMENTED_FRAME (4) - an interlaced lattice divided into two fields. Field 1 scans alternate lines from top to bottom, field 2 scans the intervening lines. The lines are stored as two fields 1,3,5,... 2,4,6,...The two fields are taken from a single scan of the incoming image – i.e., they are coincident in time, except for the effects of shutter angle. Example: "1080P24 SF". The duration of a sampled rectangle is a field.

NOTES

1. Field by field compression of an interlaced signal results in a frame layout of SEPARATE_FIELDS. This includes DV compression – even though the compressed data stream may be grouped into whole frames or fields, and even though in some cases DV uses adaptive field/frame compression, and always requires both fields of compressed data in order to successfully decompress. It also includes the adaptive field/frame compression mode of MPEG-2.

2. Together with the sample rate property, this property provides the information from which SMPTE 352M byte 2 may be derived.

E.2.3 Signal standard

The signal standard enumerated value specifies the underlying signal standard used to define the raster. If the property is not present, its default value shall be assumed to be 1 (ITU-R BT.601). Valid values are:

Value	Meaning
00h	No specific underlying standard
01h	ITU-R BT.601 and BT.656, also SMPTE 125M (525 and 625 line interlaced)
02h	ITU-R BT.1358, also SMPTE 293M (525 and 625 line progressive)
03h	SMPTE 347M (540 Mbps mappings)
04h	SMPTE 274M (1125 line)
05h	SMPTE 296M (750 line progressive)
06h	SMPTE 349M (1485 Mbps mappings)

NOTE – This property provides the information from which SMPTE 352M byte 1 may be derived.

E.2.4 Field start map

For interlaced frame layouts, raster standards define two fields. Raster standards define the line numbers of the raster (for example 1-525), and define which is called field 1 and which is called field 2. These line numbers do not include zero: the first line is line number 1. These cardinal line numbers were initially specified for analogue standards, such as ITU-R BT470. The definitions of field start lines in digital images are not identical to the definitions for analogue images. The actual numbers used in this specification are those defined by the digital standards (ITU-R BT.656, ANSI/SMPTE 125M, SMPTE 274M, SMPTE 296M, SMPTE 293M).

The field start map specifies the line number of the first line in field 1 and field 2. This is **inferred** from the signal standard property.

NOTE – Examples are: {4, 266} for 525 line 59.94 fields/s,
{1, 564} for 1080 line 59.94 fields/s,
{1, 313} for 625 line 50 field/s.

In the case of FULL_FRAME frame layout, only the start of frame is necessary, for example {1, 0} for 1080P, {1,0} for 720P.

E.2.5 Total width

Raster formats specify the total number of pixels per line. This is **inferred** from the signal standard property.

Examples: 864, 2200

E.2.6 Total height

Raster formats specify the total height of the raster in lines per frame. This is **inferred** from the signal standard property and the sample rate property.

Examples: 525, 625, 1125, 750

E.2.7 Aspect ratio

SMPTE RP 187 defines the aspect ratio of the image in the production aperture, expressed by the ratio of width to height as a rational:

For example, {4,3}, {16,9}

E.2.8 AFD active format descriptor

ETSI ETR154 defines the AFD which defines how 16:9 images are framed within a 4:3 display and vice versa. This property can be set if the desired framing is known.

E.2.9 Stored width

Stored width is equal to the number of pixels across the Stored Rectangle, expressed as a 32-bit unsigned integer.

E.2.10 Stored height

Stored height is equal to the number of pixels from top to bottom of the stored rectangle, expressed as a 32-bit unsigned integer.

E.2.11 Sampled width

Sampled width is equal to the number of pixels across the sampled rectangle, expressed as a 32-bit unsigned integer. If the property is not present, its value shall be assumed to be the stored width.

E.1.12 Sampled height

Sampled height is equal to the number of pixels from top to bottom of the sampled rectangle, expressed as a 32-bit unsigned integer. If the property is not present, its value shall be assumed to be the stored height.

E.2.13 Sampled X offset

Sampled X offset is the horizontal offset in pixels of the left edge of the sampled rectangle relative to the left edge of the stored rectangle, expressed as a 32-bit signed integer. The value may be positive or negative. If the property is not present, its value shall be assumed to be 0.

E.2.14 Sampled Y offset

Sampled Y offset is the vertical offset in pixels of the upper edge of the sampled rectangle relative to the upper edge of the stored rectangle, expressed as a 32 bit signed integer. The value may be positive or negative. If the property is not present, its value shall be assumed to be 0.

E.2.15 Video line map

Raster standards define the line numbers of the raster (for example 1-525). Only a portion of the raster is sampled (processed).

Video line map specifies the line numbers of the topmost lines in the raster to which the sampled rectangle is mapped. These line numbers do not include zero: the first line is line 1.

The order of the two numbers does **not** denote topness or field dominance.

For interlaced frame layouts, the first two line numbers are given, for example {16,278} or {15,328}. In the case of FULL_FRAME frame layout, only the topmost line is given, for example {45, 0}.

E.2.16 Sampled topness

Sampled topness specifies which field (1 or 2) of the production aperture contains the topmost line of the visible image. Sampled topness is **derived** from the video line map as follows:

Entries in Video Line Map		Sampled Topness
field 1	field 2	
odd	odd	**field 2 upper**
odd	even	**field 1 upper**
even	odd	**field 1 upper**
even	even	**field 2 upper**

E.2.17 Display width

Display width is equal to the number of pixels across the display rectangle, expressed as a 32-bit unsigned integer. If the property is not present, its value shall be assumed to be the sampled width.

E.2.18 Display height

Display height is equal to the number of pixels from top to bottom of the display rectangle, expressed as a 32-bit unsigned integer. If the property is not present, its value shall be assumed to be the sampled height.

E.2.19 Display X offset

Display X offset the horizontal offset in pixels of the left edge of the display rectangle relative to the left edge of the sampled rectangle, expressed as a 32-bit signed integer. If the property is not present, its value shall be assumed to be 0.

E.2.20 Display Y offset

Display Y offset is the vertical offset of the upper edge of the Display Rectangle relative to the upper edge of the sampled rectangle, expressed as a 32-bit signed integer. If the property is not present, its value shall be assumed to be 0.

E.2.21 DisplayF2Offset

SMPTE RP 187 defines the production aperture as the maximum possible image extent (in common usage, production aperture refers to the image extent at each stage of the production and transmission process). The SMPTE RP 187 production aperture is usually the same as the active image aperture defined by the raster standard, but in some cases it may be smaller. In other cases, transmission systems impose additional blanking upon the display.

To accommodate these cases, the DisplayF2Offset adjusts the number of lines which are blanked in field 2 relative to the number blanked in field 1.

If the property is not present, its value shall be assumed to be 0. Valid values are zero or 1.

Non-zero values of DisplayF2Offset invert the display topness relative to the sampled topness.

NOTE – SMPTE RP 187 states that the 525 line Production Aperture begins at lines {21,283}, whereas SMPTE 125M states that the Active Aperture starts at lines {20,283}.

SMPTE RP 187 blanking is typically recorded as follows:

Video Line Map = {16,278}, DisplayYOffset = 5, DisplayF2Offset = 0
The inferred Display Topness = Sampled Topness = field 2 upper.

ANSI/ SMPTE 125M blanking is typically recorded as follows:

Video Line Map = {16,278}, DisplayYOffset =4, DisplayF2Offset = 1
This inferred Sampled Topness = field 2 upper, but Display Topness is inverted (field 1 upper).

E.2.22 StoredF2Offset

The normal relationship between stored and sampled rectangles is the same for both field 1 and field 2. In some cases, the stored rectangle for field 1 starts with data from the interlaced line above the first line of field 2 (i.e., stored topness is field 1 upper), even though the sampled rectangle begins with a line from field 2 (i.e., sampled topness is field 2 upper). Other combinations are also possible.

To accommodate these cases, the StoredF2Offset property adjusts the SampledYOffset for field 2 relative to that for field 1.

If the property is not present, its value shall be assumed to be 0. Valid values are zero or minus 1.

Non-zero values of StoredF2Offset invert the stored topness relative to the sampled topness.

NOTE – StoredF2Offset is non-zero for MPEG-2 422P compression in 525 line systems.

E.2.23 SMPTE RP 187 clean aperture

SMPTE RP 187 defines a clean aperture as an image area which is slightly smaller than the production aperture. This is defined to remove edge effects which may be present in the production aperture. This specification does not record any details of the clean aperture and must be **inferred** from the underlying signal standard.

E.2.24 Pixel aspect ratio

Pixel aspect ratio is not tabulated, but may be derived from the aspect ratio, and the display height and width.

E.2.25 Field dominance

Field dominance is a property, whose unsigned 8 bit integer value specifies the field number (1 or 2) which is considered to be temporally the first field of an interlaced frame. If the property is not present, its value shall be assumed to be 1 ("field 1 first").

E.2.26 Alpha transparency

Alpha transparency is an unsigned 8 bit value which is zero (FALSE) if zero values of the Alpha channel represent a fully transparent pixel. One (TRUE) indicate that zero values of the alpha channel represent a fully opaque pixel. This flag is a modifier to the alpha channel data which may be in the essence. Its purpose is to define which value of data corresponds to transparent.

E.2.27 Gamma

Gamma is a property, whose value is a 16-byte Universal label of a registered set of color primaries, color matrix and gamma equation.

If the property is not present, its value shall be assumed to be the Universal Label identifying SMPTE 170M colorimetry and transfer.

NOTE – These values are expected to be registered in the SMPTE labels registry. It will include all those enumerated by SMPTE 268M, table 5.

E.2.28 Image alignment offset

Image alignment offset is a property, whose unsigned 32-bit integer value specifies the required alignment of the edit units in the address space of the file.

For example, 16 specifies that image data must be aligned on 16-byte boundaries.

If the property is not present, its value shall be assumed to be 1.

This property is provided for compatibility with some formats which may be wrapped by MXF (for example some uncompressed disk formats). New implementations should use KAG rules and the image alignment offset should be set to 1 unless backwards compatibility with the existing format requires a value greater than 1. Note that image alignment data will probably not be KLV wrapped, whereas KAG filling is KLV wrapped. For this reason, both must exist for backwards compatibility. The image alignment offset method is deprecated.

E.2.29 Image start offset

Image start offset is a property, whose unsigned 32-bit integer value specifies the number of unused bytes from the start of the stored data for a sample unit to the start of the stored rectangle. If the property is not present, its value shall be assumed to be 0.

E.2.30 Image end offset

Image end offset is a property, whose unsigned 32-bit integer value specifies the number of unused bytes from the end of the stored rectangle to the end of the stored data for a given sample unit. If the property is not present, its value shall be assumed to be 0.

E.2.31 Picture essence coding

Picture essence coding is a 16-byte UL defining the essence coding / compression scheme in use.

E.2.32 Component depth

Component depth is a property, whose unsigned 32-bit integer value specifies the number of active bits per pixel. Typical values are 8 or 10.

E.2.33 Horizontal subsampling

Horizontal subsampling is a property, whose unsigned 32-bit integer value specifies the horizontal subsampling factor of the chrominance samples relative to the luminance samples. Typical values are 2 or 4. Other values are permitted.

E.2.34 Vertical Subsampling

Vertical subsampling is a property, whose unsigned 32-bit integer value specifies the vertical subsampling factor of the chrominance samples relative to the luminance samples. Typical values are 1, 2. Other values are permitted.

E.2.35 Color siting

Color siting is a property, whose enumerated unsigned 8 bit value specifies how to compute subsampled chrominance component values. Values are:

0	coSiting	The first luminance value of the image is co-sited with the first chrominance value.
1	mid-point	The color pixel is sited at the point horizontally midway between the luminance pixels on each line.
2	threeTap	reserved
3	Quincunx	Color samples are sited at the point midway between two adjacent luminance pixels on two adjacent lines, as in MPEG-1 4:2:0
4	Rec601	Color samples are known to be in accordance with ITU-R Rec 601
FFh	Unknown	The siting of the color samples is unknown

If the property is not present, its value shall be assumed to be zero (coSiting).

E.2.36 PaddingBits

PaddingBits is a property, whose 16-bit integer specifies the number of bits to round up each pixel to the stored size of each pixel. If the property is not present, its value shall be assumed to be 0.

E.2.37 Alpha sample depth

Alpha sample depth is a property, whose unsigned 32-bit integer value specifies the number of active bits per sample in the alpha channel. Typical values are 8 or 10. If the property is not present, its value shall be assumed to be 0.

E.2.38 Black ref level

Black ref level is a property, whose unsigned 32-bit integer value specifies the pixel value for reference black level. Typical values are 16 or 64 (for 8- or 10-bit samples, respectively).

For ITU-R BT.601, the value is 16; for full range 8-bit video, the value is 0. The same value is used in CDCI (and in RGBA when the standard ITU-R BT.601 color space conversion is used). If omitted the value is 0.

E.2.39 White ref level

White ref level is a property, whose unsigned 32-bit integer value specifies the pixel value for reference white level. Typical values are 235 or 940 (for 8- or 10-bit samples, respectively).

For ITU-R BT.601, 8-bit video, the value is 235; for full range 8-bit video, the value is 255 (1023). If omitted, the value is maximum unsigned integer value for component size.

E.2.40 Color range

Color range is a property, whose unsigned 32-bit integer value specifies the number of distinct values allowed for chroma samples. Typical values are 225 or 897 (for 8- or 10-bit samples, respectively).

The property specifies the nominal range of digital chrominance component values. Chroma values are signed and the range specified is centered on 80h or 200h (**not** twos complement). For ITU-R BT.601, the value is 225 (or 897); for full

range YUV the value is 255 (or 1023). This value is used for both chrominance components. If not present, the value is the maximum unsigned integer value for the component size (256 or 1024).

E.2.41 Reversed byte order

Reversed byte order is a property, whose boolean value is one (TRUE) if the luminance sample precedes the chroma sample in the stored data. It is zero (FALSE) if the sample order conforms to the ITU-R BT.601 standard (first chroma sample precedes luminance sample).

If the property is not present, its value shall be assumed to be zero (FALSE) meaning ITU-R BT.601 sample order.

Other sample orderings are possible, but must be specified by the use of an RGBA descriptor. Note that although the title of this property is "byte order", it refers to the whole sample.

E.2.42 PixelLayout

PixelLayout is a property (for RGBA sampling only), whose vector value describes the format of each pixel. It is described in detail in section E.2.46.

E.2.43 Palette

PaletteLayout is a property (for RGBA palletized sampling only), whose vector value describes the value of each palette entry. It is described in detail in section E.2.46.

E.2.44 PaletteLayout

PaletteLayout is a property (for RGBA palletized sampling only), whose vector value describes the format of each sample. It is described in detail in E.2.46.

E.2.45 Scanning direction item

Scanning direction is a property, whose 8 bit enumerated value describes the scanning direction of the image. It exactly matches the equivalent property in SMPTE 268M. If not present, the value zero is assumed.

Property Name	Type	Explanation
ScanningDirection	Orientation	Specifies the scanning direction of the image, according to the following enumerated values:

Code	**line direction** (followed by)	**frame direction**
0	left to right	top to bottom
1	right to left	top to bottom
2	left to right	bottom to top
3	right to left	bottom to top
4	top to bottom	left to right
5	top to bottom	right to left
6	bottom to top	left to right
7	bottom to top	right to left
8-254	reserved for future use	

E.2.46 Pixel layout item

Property Name	Type	Explanation
PixelLayout	RGBALayout	Specifies the type, order and size of the components within the pixel. The RGBALayout type is a fixed-size 8 element array, where each element consists of the RGBAComponent type with the following fields: Code — Uint8 — Enumerated value specifying component 0→PixelLayout terminator Depth — UInt8 — Integer specifying the number of bits occupied 1→32 indicates integer depth 254→IEEE floating point 32 bit value 255→IEEE floating point 64 bit value 0→PixelLayout terminator For each component in the pixel, the following Codes should be specified (explained below): **Code / ASCII / meaning** 0x52 'R' Red component 0x47 'G' Green component 0x42 'B' Blue component 0x41 'A' Alpha component 0x72 'r' Red component (LSBs) 0x67 'g' Green component (LSBs) 0x62 'b' Blue component (LSBs) 0x61 'a' Alpha component (LSBs) 0x46 'F' Fill component 0x50 'P' Palette code 0x55 'U' Chroma Sample (e.g. U, Cb, I etc.) 0x56 'V' Chroma Sample (e.g. V, Cr, Q etc.) 0x57 'W' Composite Video 0x58 'X' Non co-sited luminance component 0x59 'Y' Luminance component 0x5A 'Z' Depth component (SMPTE 268M compatible) 0x75 'u' Chroma Sample (e.g. U, Cb, I etc.) (LSBs) 0x76 'v' Chroma Sample (e.g. V, Cr, Q etc.) (LSBs) 0x77 'w' Composite Video (LSBs) 0x78 'x' Non co-sited luminance component (LSBs) 0x79 'y' Luminance component (LSBs) 0x7A 'z' Depth component (LSBs) (SMPTE 268M compatible) 0x00 Terminates list of components A Fill component indicates unused bits. After the components have been specified, the remaining Code and Size fields shall be set to 0.
Palette	DataValue	An array of color values that are used to specify an image. Size specified by parsing the PaletteLayout property.
PaletteLayout	RGBALayout	An array of PixelLayout elements which specifies the order and size of the color components as they are stored in the palette.

An RGBA descriptor set describes content data that contains component-based images where each pixel is made up of a red, a green and a blue value code; other component types may also be specified.

Optionally, an alpha value can be included in each pixel. The alpha value determines the transparency of the color.

The PixelLayout property provides for specification of the order that the color components are stored in the image, the number of bits needed to store a pixel, and the bits allocated to each component. This covers a wide variety of scanning and packing formats, including all those of SMPTE 268M.

The 'R', 'G', 'B' and 'A' or 'M' codes specify red, green, blue and alpha or mask components. The fill ('F') code allows for insertion of extra bits to pack the components into convenient word sizes. If the PixelLayout property includes an 'R', 'G', or 'B', then it shall not include a 'P'.

The 'r', 'g', 'b' and 'a' or 'm' codes specify the lesser significant bits of components when the components are split into two contiguous bit fields for efficient pixel packing. Such bit-packing schemes are used occasionally for special-purpose imaging.

The palette ('P') code allows for specification of palletized color sampling, in which each pixel is described by an index into a pixel palette. If the PixelLayout property includes a 'P', then it shall not include an 'R', 'G', or 'B'. If the PixelLayout property includes a 'P', then the RGBADescriptor object shall include the palette and PaletteLayout properties. The palette and PaletteLayout properties specify the color palette itself and the structure used to store each color in the palette.

The 'U', 'V', 'W', 'X', 'Y' and 'Z' codes allow for specification of abnormal subsampled color and single component sampling (all standardized methods use the CDCI descriptor). If the PixelLayout property includes any of these, then it shall not include a 'P'.

The 'u', 'v', 'w', 'x', 'y' and 'z' codes specify the lesser significant bits of components when the components are split into two contiguous bit fields for efficient pixel packing. Such bit-packing schemes are used occasionally for special-purpose imaging.

RGBA content data can be converted to CDCI and then compressed using a preferred compression scheme. Once the data has been converted and compressed, it is described by a CDCIDescriptor essence descriptor.

The following examples show the values of PixelLayout corresponding to several standard sampling structures:

Example: Component 4:2:2:4

8-bit components packed into a 32-bit word, in 601 sequence: Cb Y Cr, with alpha in 4th byte of the stored pixel

PixelLayout= { 'U', 8, 'Y', 8, 'V', 8, 'A', 8, 0, 0 }

Example: one of the formats supported by SMPTE 268M

10-bit components filled to 32-bit word boundaries, padded in most significant bits

PixelLayout= { 'F', 2, 'B', 10, 'G', 10, 'R', 10, 0, 0 }

Note that the MXF descriptors provide a mechanism to support a similar range of image types to those covered by SMPTE 268M.

Annex F (informative)
Bibliography

ANSI/SMPTE 125M-1995 Television — Component Video Signal 4:2:2 — Bit-Parallel Digital Interface

SMPTE 12M-1999, Television, Audio and Film — Time and Control Code

SMPTE 268M-2003, File Format for Digital Moving-Picture Exchange (DPX), Version 2.0

SMPTE 274M-2003, Television — 1920 x 1080 Image Sample Structure, Digital Representation and Digital Timing Reference Sequences for Multiple Picture Rates

SMPTE 293M-2003, Television — 720 x 483 Active Line at 59.94-Hz Progressive Scan Production — Digital Representation

SMPTE 296M-2001 Television — 1280 x 720 Progressive Image Sample Structure — Analog and Digital Representation and Analog Interface

SMPTE 330M-2004, Television — Unique Material Identifier (UMID)

SMPTE 352M-2002, Television — Video Payload Identification for Digital Television Interfaces

SMPTE RP 187-1995, Center, Aspect Ratio and Blanking of Video Images

SMPTE EG 41-2004, Material Exchange Format (MXF) — Engineering Guideline

SMPTE EG 42-2004, Material Exchange Format (MXF) — Descriptive Metadata

AES3-2003, AES Recommended Practice for Digital Audio Engineering — Serial Transmission Format for Two-Channel Linearly Represented Digital Audio Data

ATSC A/52, www.atsc.org, A/52 Specification — Equivalent of Normative: ITU-R BS.1196 (1995) (Annex 2): Audio Coding for Digital Terrestrial Television Broadcasting

ETSI-ETR 154 (09/97), Implementation Guidelines for the Use of MPEG-2 Systems, Video and Audio in Satellite, Cable and Terrestrial Broadcasting Applications (for Definition of AFD)

ISO/IEC 8825-1:1998 Information Technology — ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER) — this is referenced by SMPTE 336M

ISO/IEC 13818-2:2000, Information Technology — Generic Coding of Moving Pictures and Associated Audio Information: Video, (MPEG-2)

ITU-R BT470-6 (11/98), Conventional Television Systems

ITU-R BT.601-5 (10/95), Studio Encoding Parameters of Digital Television for Standard 4:3 And 16:9 Aspect Ratios

ITU-R BT.656 (02/98), Interfaces for Digital Component Video Signals in 525-Line and 625-Line Television Systems Operating at the 4:2:2 Level of Recommendation ITU-R BT.601 (Part A)

Advanced Authoring Format, http://www.aafassociation.org

EBU / SMPTE Task Force for Harmonized Standards for the Exchange of Program Material as Bit-Streams — 1998, http://www.smpte.org and http://www.ebu.ch

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