

SMPTE STANDARD**Material Exchange Format (MXF) —
File Format Specification**

Page 1 of 183 pages

Table of Contents**Page**

Foreword	7
Intellectual Property	7
1 Scope	8
2 Conformance Notation	8
3 Normative References	8
4 Definition of Acronyms, Terms and Data Types	9
4.1 Acronyms and Terms	9
4.2 Simple Data Types	17
4.2.1 Storage order of UL and UUID values.....	18
4.3 Compound Data Types.....	19
4.4 Guide to the Use of KLV Pack and KLV Set Definition Tables	20
5 Introduction (Informative)	21
5.1 Structure of this Document.....	22
5.2 SMPTE 377M Revisions.....	22
5.2.1 Major differences between SMPTE 377M-2004 and this specification (Informative).....	22
5.2.2 In-file version numbers	23
5.2.3 Manipulating files conforming to other revisions of this specification (Normative).....	23
5.3 Parts of the MXF Specification	24
5.4 The MXF File	25
6 Overall Specification	26
6.1 Overall Data Structure	26
6.1.1 File Header	26
6.1.2 File Body	27
6.1.3 File Footer.....	27
6.2 Partitions.....	27
6.2.1 Partition Rules Overview	28
6.2.2 Partition Pulse Summary (Informative).....	29
6.2.3 Partition Status	30
6.2.4 The Status of an MXF File	31
6.2.5 Header Partition.....	31
6.2.6 Body Partition	31
6.2.7 Footer Partition	32
6.2.8 Using Partitions to multiplex Essence Containers and associated Index Tables (Informative)	32
6.3 KLV Coding.....	33
6.3.1 KLV Coding Sequence	33
6.3.2 KLV Coded Dark Components	33

6.3.3	KLV Fill Items	34
6.3.4	KLV Lengths.....	35
6.3.5	Local set Lengths	35
6.3.6	Variable-Length Pack Lengths	35
6.3.7	Defined-Length Pack Lengths	35
6.3.8	MXF Keys and Universal Labels	35
6.3.9	Constraints on recursive Groupings of KLV items	36
6.3.10	The Primer Pack, Dark Metadata and MXF extensions.....	36
6.4	MXF Encoding Requirements	36
6.4.1	KLV Alignment Grid (KAG).....	38
6.4.2	MXF Byte Order	39
6.4.3	Encoding Constraints	39
6.5	Run-In Sequence	39
6.6	Minimum MXF Decoder (Informative)	39
6.7	Strong and Weak Reference Integrity (Informative)	40
7	Partitions.....	40
7.1	Partition Pack	40
7.2	Header Partition Pack	44
7.2.1	Header Partition Pack Key	44
7.2.2	Header Partition Pack Values	45
7.3	Body Partition Pack.....	45
7.3.1	Body Partition Pack Key.....	45
7.3.2	Body Partition Pack Value.....	46
7.3.3	Header Metadata Repetition in Body Partitions	46
7.4	Footer Partition Pack.....	46
7.4.1	Footer Partition Pack Key	46
7.4.2	Footer Partition Pack Value	47
7.4.3	Header Metadata Repetition in the Footer Partition	47
7.5	Header Metadata Repetition in Body and Footer Partitions	47
7.5.1	Application Guidelines for header Metadata Repetition (Informative).....	47
7.5.2	Tracking Changes with Generation UID	48
8	Operational Patterns.....	49
8.1	General	49
8.2	Generic Universal Label for All Operational Patterns	49
8.3	Generalized Operational Patterns.....	50
8.3.1	Item complexity	50
8.3.2	Package complexity	50
8.3.3	Universal Label for Generalized Operational Patterns	51
8.4	Specialized Operational Patterns.....	52
8.4.1	Universal Label byte values for Specialized Operational Patterns	52
8.5	Package Hierarchy in Operational Patterns	53
9	Header Metadata.....	53
9.1	Header Metadata KLV Packet Sequence	53
9.2	Primer Pack.....	54
9.2.1	Contents of the Primer	56
9.2.2	Local Tag values.....	56
9.2.3	Dark Metadata Support	56
9.3	Header Metadata Set Coding	57
9.3.1	Data Model (Informative).....	58
9.3.2	Strong and Weak References	58
9.3.3	Uniqueness of Instance UID values.....	59
9.4	Structural Metadata Semantics.....	59
9.4.1	Explanation of Figures illustrating the Structural Metadata Semantics (Informative).....	59
9.4.2	The MXF timing Model	61

9.4.3 Relationship between File Packages and Essence Containers	64
9.5 Structural Metadata Definition	64
9.5.1 Header Metadata start	64
9.5.2 Generic Class diagram (Informative)	65
9.5.3 Material Package	67
9.5.4 Source Package	67
9.5.5 Top-Level File Packages	67
9.5.6 Lower-Level Source Packages	69
9.5.7 Relationship between the Packages and SourcePackageID / SourceTrackID	69
9.5.8 Relationship between the BodySID and IndexSID	71
9.5.9 Scope of the Track ID values	71
9.6 Structural Header Metadata Implementation	71
9.6.1 KLV Key values for Structural Metadata Sets	71
9.6.2 Universal Labels for Abstract Structural Metadata Groups	73
9.7 Application Metadata Plug-Ins	75
9.7.1 General (Informative)	75
9.7.2 Application Metadata Scheme Specification	75
9.7.3 Generic Universal Label for the MXF Application Metadata Schemes	75
9.7.4 Plug-In Mechanism	76
9.7.5 Simple Application Metadata Plug-In Instance Removability	78
9.7.6 Simple Application Metadata Plug-In Instance Removal Implementation (Informative)	79
9.7.7 Use of the Application Metadata Plug-In Mechanism	79
9.7.8 Application-Specific Metadata Plug-In Mechanism Example (Informative)	80
9.8 Descriptive Metadata Plug-Ins	82
9.8.1 General (Informative)	82
9.8.2 Generic Universal Label for the MXF Descriptive Metadata Schemes	82
9.8.3 Generic MXF Descriptive Metadata Keys	83
9.8.4 Universal Labels for Abstract Descriptive Metadata Groups	84
9.8.5 Plug-In Mechanism	85
9.8.6 Simple DM Plug-In Instance Removability	87
9.8.7 Simple DM Plug-In Instance Removal Implementation (Informative)	88
10 File Body	88
10.1 Essence Containers	88
10.2 Technical Requirements for MXF Essence Containers	89
10.3 Standards Requirements of an MXF Essence Container document	89
10.4 General Information (Informative)	90
10.5 Descriptors	90
10.5.1 Use of Descriptors in File Packages	90
10.5.2 Use of Descriptors in Physical Packages	91
10.5.3 Use of Locators	91
10.5.4 Extending Essence Descriptors	91
10.6 Interleaved Essence Containers	92
11 Index Table	92
11.1 Overview	92
11.1.1 Interleaved Streams	93
11.1.2 Constant Bytes per Element (CBE) and Variable Bytes per Element (VBE) (Informative)	93
11.1.3 Complex Interleaves of Compressed Audio	94
11.1.4 Description of Operation	95
11.1.5 Generalization using Element Date	97
11.1.6 Temporal Re-ordering	97
11.1.7 Indexing Empty Essence Elements	98
11.1.8 Indexing KLV Fill Items	99
11.1.9 Constant Edit Unit Size	99
11.2 Index Table Specification	99
11.2.1 Index Table Segments	100

11.2.2	Index Table Segment Key	101
11.2.3	Index Table Segment	102
11.3	Partial / Sparse Index Tables for VBE Essence	106
11.4	To Find the Byte Offset for an Essence Element (Informative)	107
11.5	Using Index Tables for Internal Essence and External Essence	107
11.5.1	BodySID nonzero, IndexSID nonzero	107
11.5.2	BodySID zero, IndexSID nonzero	107
11.5.3	BodySID nonzero, IndexSID zero	108
11.5.4	BodySID zero, IndexSID zero	108
11.6	Additional Information (Informative)	108
11.6.1	Relationship between Top-Level File Package Essence Timeline Tracks and Index Entries	108
11.6.2	Look-up Algorithm for Conversion of Index Position to Stream Offset	108
12	Random Index Pack	110
12.1	Random Index Pack Key	110
12.2	The Random Index Pack Value	111
12.3	Algorithm for using the Random Index Pack (Informative)	111
Annex A	Specifications for Root Metadata Sets (Normative)	112
A.1	Interchange Object	112
A.2	Preface	113
A.3	Identification	114
A.4	Content Storage	116
A.5	Essence Container Data	116
Annex B	Specifications for Generic Package (Normative)	117
B.1	Generic Package	117
B.2	Generic Descriptor	117
B.3	SubDescriptor	118
B.4	Network Locator	118
B.4.1	URL file://	118
B.4.2	URL ftp://	119
B.4.3	URIs	119
B.4.4	Handling invalid or unknown URLs and URIs	119
B.5	Text Locator	119
B.6	Generic Track	119
B.7	Track ID Usage	120
B.8	Structural Component	121
B.9	Sequence	121
B.10	Source Clip	122
B.11	Filler	122
B.12	Timeline Track	123
B.13	Track Event	123
B.14	Static Track	124
B.15	Timeline Track (Timecode)	124
B.16	Sequence (Timecode)	125
B.17	Timecode Component	125
B.18	Timeline Track (Picture)	126
B.19	Sequence (Picture)	126
B.20	Source Clip (Picture)	126
B.21	Timeline Track (Sound)	127
B.22	Sequence (Sound)	127
B.23	Source Clip (Sound)	127
B.24	Timeline Track (Data)	127
B.25	Sequence (Data)	128
B.26	Source Clip (Data)	128
B.27	Dm Tracks	128

B.27.1	Timeline Track (DM)	128
B.27.2	Event Track (DM)	129
B.27.3	Static Track (DM)	129
B.28	Sequence (DM)	129
B.29	Segment	129
B.30	Event	130
B.31	Comment Marker	130
B.32	DM Segment	131
B.33	DM Source Clip	132
B.34	Package Marker Object	132
Annex C	Specification of the Application-Specific Metadata Plug-In Mechanism Sets (Normative)	134
C.1	Application Object	134
C.2	Application Plug-In Object	134
C.3	Application Referenced Object	136
Annex D	Specification of the DM Plug-In Mechanism Sets (Normative)	137
D.1	Descriptive Framework	137
D.2	Descriptive Object	137
Annex E	Specification for the Packages Used in MXF (Normative)	138
E.1	Material Package	138
E.2	Source Package	138
E.3	File Package	139
E.4	Physical Package	139
E.5	Package hierarchy in MXF	139
Annex F	Specification of Descriptors Used in MXF (Normative)	141
F.1	Scope of Descriptor Property Values	141
F.2	File Descriptor	141
F.3	Multiple Descriptor	143
F.4	Picture Essence Descriptors	143
F.4.1	Generic Picture Essence Descriptor	143
F.4.2	CDCI (Color Difference Component Image) Picture Essence Descriptor	146
F.4.3	RGBA (Red Green Blue Alpha) Picture Essence Descriptor	148
F.5	Generic Sound Essence Descriptor	149
F.6	Generic Data Essence Descriptor	150
Annex G	Picture Essence Descriptor Properties (Normative)	151
G.1	Data Storage, Sampling, Display and Video Interface	151
G.1.1	Stored Data and Stored Rectangle	152
G.1.2	Sampled Rectangle	153
G.1.3	Display Rectangle	153
G.1.4	Video Interface	153
G.1.5	Sampling	154
G.2	Property Definitions	154
G.2.1	Frame Layout	154
G.2.2	Sample Rate and Edit Rate	156
G.2.3	Signal Standard	156
G.2.4	Aspect Ratio	156
G.2.5	Active Format Descriptor (AFD)	157
G.2.6	Stored Width	158
G.2.7	Stored Height	158
G.2.8	Sampled Width	158
G.2.9	Sampled Height	159
G.2.10	SampledXOffset	159
G.2.11	SampledYOffset	159
G.2.12	Video Line Map	159

G.2.13	DisplayWidth	160
G.2.14	DisplayHeight	160
G.2.15	DisplayXOffset	160
G.2.16	DisplayYOffset	161
G.2.17	DisplayF2Offset	161
G.2.18	StoredF2Offset	161
G.2.19	FieldDominance	162
G.2.20	Alpha Transparency	162
G.2.21	Transfer Characteristic	162
G.2.22	Image Alignment Offset	162
G.2.23	Image Start Offset	163
G.2.24	Image End Offset	163
G.2.25	Picture Essence Coding	163
G.2.26	Component Depth	166
G.2.27	Horizontal Subsampling	166
G.2.28	Vertical Subsampling	166
G.2.29	Color Siting	166
G.2.30	PaddingBits	167
G.2.31	Alpha Sample Depth	167
G.2.32	Black Ref Level	167
G.2.33	White Ref Level	168
G.2.34	Color Range	168
G.2.35	Reversed Byte Order	168
G.2.36	PixelLayout	168
G.2.37	Palette	169
G.2.38	PaletteLayout	169
G.2.39	Scanning Direction	169
G.2.40	Pixel Layout	170
Annex H	Static Local Tags Assigned by MXF Specifications (Normative)	172
Annex I	Bibliography (Informative)	181

Foreword

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

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

SMPTE ST 377-1 was prepared by Technology Committee 31FS.

Intellectual Property

At the time of publication no notice had been received by SMPTE claiming patent rights essential to the implementation of this Standard. However, attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. SMPTE shall not be held responsible for identifying any or all such patent rights.

1 Scope

This document 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 document 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 document 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 Conformance Notation

Normative text is text that describes elements of the design that are indispensable or contains the conformance language keywords: "shall", "should", or "may". Informative text is text that is potentially helpful to the user, but not indispensable, and can be removed, changed, or added editorially without affecting interoperability. Informative text does not contain any conformance keywords.

All text in this document is, by default, normative, except: the Introduction, any section explicitly labeled as "Informative" or individual paragraphs that start with "Note:"

The keywords "shall" and "shall not" indicate requirements strictly to be followed in order to conform to the document and from which no deviation is permitted.

The keywords, "should" and "should not" indicate that, among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

The keywords "may" and "need not" indicate courses of action permissible within the limits of the document.

The keyword "reserved" indicates a provision that is not defined at this time, shall not be used, and may be defined in the future. The keyword "forbidden" indicates "reserved" and in addition indicates that the provision will never be defined in the future.

A conformant implementation according to this document is one that includes all mandatory provisions ("shall") and, if implemented, all recommended provisions ("should") as described. A conformant implementation need not implement optional provisions ("may") and need not implement them as described.

3 Normative References

Note: All references in this document to other SMPTE documents use the current numbering style (e.g. SMPTE ST 335:2001) although, during a transitional phase, the document as published (printed or PDF) may bear an older designation (such as SMPTE 335M-2001). Documents with the same root number (e.g. 335) and publication year (e.g. 2001) are functionally identical.

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, 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.

SMPTE ST 298:2009, Universal Labels for Unique Identification of Digital Data

SMPTE ST 330:2011, Unique Material Identifier (UMID)

SMPTE ST 335:2001, Television — Metadata Dictionary Structure

SMPTE ST 336:2007, Data Encoding Protocol Using Key-Length-Value

SMPTE ST 395:2003, Television — Metadata Groups Registry Structure

SMPTE ST 400:2004, Television — SMPTE Labels Structure

SMPTE ST 2016-1:2009, Format for Active Format Description and Bar Data

IEEE 754-1985, Floating Point Format

ITU-R BS.1196-2 (03/10), Audio Coding for Digital Broadcasting

IETF RFC 3986, Uniform Resource Identifier (URI): Generic Syntax

IETF RFC 4122, A Universally Unique Identifier (UUID) URN Namespace

ISO/IEC 8825-1:2008 Information Technology — ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)

ISO/IEC 10646:2003, Information Technology — Universal Multiple-Octet Coded Character Set (UCS)

4 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.

Note: Throughout the document, defined terms are written in capital letters in order to disambiguate between the use of the defined term and use of the same word in its more general, English language meaning.

4.1 Acronyms and Terms

AAF: Advanced Authoring Format.

Abstract Class: A Class used within the definition of an inheritance hierarchy that cannot be instantiated as an Object. Only non-abstract Subclasses of an Abstract Class can be instantiated as Objects.

Note: Some Abstract Classes defined in this (and possibly other MXF documents) may use the term "object" in the Class name. Such naming does not imply that the Abstract Class can be instantiated directly as an Object."

Aggregation: A form of Association that defines a whole-part relationship between a whole Object and a part Object or multiple part Objects. See Composition.

Application Environment: A defined area of interchange where specific information is needed in order to satisfy requirements of applications (e.g. software products or devices) operating within that environment. Examples of Application Environments could be multiple facilities of a large scale operation, an individual facility such as a specific broadcast playout, or the interchange of data between related products of a single vendor.

Application Metadata Scheme: A defined set of application information that can be added to an MXF file.

Application-Specific Metadata: Application-Specific Metadata (ASM) is data conforming to an Application Metadata Scheme that is targeted to be used in specific Application Environments and added to the Header Metadata of an MXF file using the mechanism defined in Section 9.7.

Note: By omitting the optional Application Environment ID Property, it is possible to add Application-Specific Metadata to an MXF file such that its use is not constrained to one specific and defined Application Environment.

Association: A structural relationship that defines a set of interconnections between Objects. An association may be specified as a reference from one Object to another Object or Objects. See Aggregation and Composition.

Audio-Visual: A term used to describe any playable content comprising Picture (visual), Sound (audio) or Data Essence. It includes any metadata that is embedded in the content. The content may comprise a one or more Essence Elements (of the same kind or different kinds) that are interleaved at an appropriate rate.

BER: Basic Encoding Rules, defined in ISO/IEC 8825-1, for encoding KLV lengths and SMPTE Universal Labels.

Best Effort: Best Effort Metadata is a category of Metadata Properties which may not be known by the MXF encoder at the time of writing the file. Best efforts should be made to enter the correct values during file creation. If the correct value of the Property cannot be determined the value of the Property shall be set to its defined Distinguished Value. Only metadata Properties specifically defined as Best Effort may be treated this way.

Big Endian: A byte order where the bytes of a word are transmitted such that the most significant Byte is transmitted first.

BodySID: Property that holds the Stream ID of an Essence Container.

CBE: Constant Bytes per Element.

CBR: Constant Bit Rate.

CDCI: Color Difference Component Image.

Class: A data structure with defined Properties and defined behavior.

Composition: A stronger form of Aggregation in which parts may only belong to one whole and may only exist as part of the whole. Composition is used to combine simple Objects into more complex ones..

Concrete Class: A Class used within the definition of an inheritance hierarchy that can be instantiated as an Object. All Classes in this standard that are not defined as Abstract Classes shall be Concrete Classes.

Content Package: A grouping of combinations of system, picture, sound, data or compound Essence Element KLV packets. For further details see SMPTE ST 379.

Dark: Used to describe Essence and metadata items and values 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 & forward device, needed and processed by a librarian program for archive, but discarded by the application for broadcast.

Default Value: If specified for an optional Property, this is the value that an MXF application that requires it should use if the Property has been omitted in an Object.

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

Distinguished Value: A defined value of a Best Effort metadata Property. The value shall be emitted by MXF encoders when the correct value of the Property is not known. The value is always defined such that it cannot be valid under any circumstances. When processing MXF files, decoders shall treat Best Effort Properties that contain their Distinguished Value as no information available.

Descriptive Metadata: Descriptive Metadata (DM) is data placed in the file as descriptive annotation of the Essence. Descriptive Metadata that is placed in the Header Metadata of an MXF file uses the plug-in mechanism defined in Section 9.8.

Descriptive Metadata Scheme: A defined set of descriptors to be added to the sound, picture or data essence in an MXF file.

Duration: An integer number that defines the temporal length of a property in Edit Units.

Edit Rate: The rational number that specifies the Edit Units used to define the Duration of Components in a Track. The Edit Rate is the number of Edit Units that equal one elapsed second. In Tracks which describe an Essence Container, the Edit Rate usually equals the number of Editable Units per second for one of the Essence Elements.

Edit Unit: A period of time equal to $1/(\text{Edit Rate})$.

Editable Unit: The smallest portion of an Essence stream which can be edited such as a field of a picture, a frame or an audio sample.

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

Essence Element: Essence stream within an Essence Container. It may be a single KLV packet or a sequence of KLV packets that use the same KLV Key value.

Essence Container: A part of an MXF file that carries one or more Essence streams and, optionally, Metadata closely associated with them. Each Essence Container constitutes a stream of bytes that is identified by a unique Stream ID value.

Event Origin: The number of Edit Units which exist before the Zero Point of an Event Track. A positive value shall indicate that the Event Track starts before the Zero Point of the Package. A negative value shall indicate that the Event Track starts after the Zero Point of the Package.

File Package: A Source Package that strongly references a File Descriptor. It either describes the Essence in an Essence Container internal to the MXF file, an Essence Container of another MXF file or the Essence stored in a non-MXF file or provides historical Essence derivation information. A File Package can be a Top-Level File Package or a Lower-Level Source Package.

Framework: The name for a defined collection of related MXF Descriptive Metadata Sets.

Global Weak Reference: A Weak Reference of global scope. It is expressed via the value of an immutable Property. Examples of immutable Properties are the Package UID (UMID value that uniquely identifies a Package) or the Universal Label identifier of an item in a SMPTE register. The target of a Global Weak Reference may lie within the same Header Metadata instance as the Reference itself.

Other normative parts of the MXF specification (see Section 5.3) may define additional kinds of Weak References.

Generic Package: The Abstract Superclass of all Packages. It aggregates one or more Tracks.

Generic Track: The Abstract Superclass of all Tracks.

GOP: An acronym for an MPEG 'Group of Pictures'.

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 the value of a byte represented as a pair of hexadecimal digits.

In-File Weak Reference: A Weak Reference of local scope. The scope shall be an instance of Header Metadata. In-File Weak References are expressed via the value of a Property that is unique within an instance of Header Metadata. Examples for such Properties are Instance UID or This Generation UID.

Other normative parts of the MXF specification (see Section 5.3) may define additional kinds of Weak References.

IndexSID: Property that holds the Stream ID for an Index Table.

Index Table: A lookup table which converts a desired time offset on the timeline of a File Package into a byte offset within an Essence Container in an MXF file or the Essence contained in a non-MXF file. For MXF Essence Containers composed of multiple Essence Elements, Index Tables may also contain byte offsets for individual Elements or provide relative timing information between individual Elements.

Inheritance: Creating a new Class by taking the Properties and behaviors of a parent Class and adding to them and/or overriding them. The new Class is also called a Subclass of its parent classes.

Interchange Object: The Abstract Class that is the parent Class of all MXF Header Metadata Sets. See Annex A.1.

Interleave: To closely combine Essence byte streams of two or more Essence Elements over a short duration within a given Essence Container byte stream for the purpose of continuous playback.

Item Designator: The last 8 bytes of a SMPTE Universal Label. They uniquely identify an entry in a SMPTE register. The Item Designator is a reference to the SMPTE label inside the register that is identified by the UL Designator. SMPTE registers relevant to this specification are SMPTE ST 335, SMPTE ST 400 and SMPTE ST 395. In the case of the SMPTE Metadata Dictionary (i.e. the register defined by SMPTE ST 335), the register is the normative reference that defines the item or term. Any explanatory text in this document is informative only.

KAG: KLV Alignment Grid. A notional byte spacing which may be used to align KLV items within a Partition.

KLV: Key Length Value. The binary encoding protocol defined in SMPTE ST 336.

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 Item is encoded using a locally unique tag value of the same length. The length is 1, 2 or 4 bytes. See SMPTE ST 336.

Local Tag: 1, 2 or 4 byte long tag value used to identify items in a Local Set. According to SMPTE ST 336, the Tag byte-order is big-endian.

Lower-Level Source Package: A Source Package (File Package or Physical Package) which is not directly referenced by a Material Package that is internal to the file, but that is referenced by a Source Package that is internal to the file. Lower-Level Source Packages are used for historical annotation (or the derivation information) for the contents described by Top-Level File Package(s). They are not associated with an Essence Container in the file and any reference to Essence Containers or external Essence files is treated as historical.

Material Package: A Generic Package Subclass that describes an output timeline of the MXF file.

MPEG: The Moving Picture Experts Group of the International Standards Organization (ISO).

Multiplex: To combine the separate byte streams of the MXF file (e.g. Header Metadata, Index Tables, and Essence Containers) into the byte stream of the MXF file using Partitions.

Multiplicity: Specifies the range of allowable cardinalities that Objects of associated Classes can assume.

MXF Local Set: An MXF Set employing 2-byte Local Tag encoding.

MXF Set: A KLV encoded instance of a Class derived from Interchange Object.

NDE: The number of Delta Entries in an Index Table Segment.

NIE: The number of Index Entries in an Index Table Segment.

NPE: The number of Position Table Entries in an Index Table Segment.

NSL: The number of Slices in an Index Table minus 1 (i.e. the number of offsets stored in the SliceOffset array). The first slice has always an offset of 0 and is not stored.

Object: An instance of a Class. A binary representation of an Object that uses KLV encoding is called Set or Pack, depending on the details of the KLV encoding syntax.

Operational Pattern: Specifies the level of file complexity as described in Section 8.

The Operational Pattern of a file is identified via a UL value in properties calls Operational Pattern that are stored in the Preface and in the Partition Packs.

Origin: The number of Edit Units of a Track which exist before the Zero Point of a Timeline Track. A positive value shall indicate that the Timeline Track starts before the Zero Point of the Package. A negative value shall indicate that the Timeline Track starts after the Zero Point of the Package.

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

Package: The name used for all Subclasses of the Generic Package.

Package Duration: The temporal length of a Package defined as *earliest_end_position* – *latest_start_position*. See Section 9.4.2.

Partition: A logical separation of the MXF file. An MXF file consists of a sequence of Partitions. Partitions are used to multiplex the different KLV-encoded data streams or segments of those data streams (Header Metadata, Essence Containers and Index Tables) into the single byte stream that constitutes the MXF file.

Physical Package: A Source Package that strongly references a Physical Descriptor to describe a physical entity such as a tape. Physical Packages shall only be used as Lower-Level Source Packages (i.e. for historical annotation) in Generalized Operational Pattern (see Section 8.3). Specialized Operational Pattern may use Physical Packages as Top-Level Source Package.

Pixel: It is a contraction of the phrase “picture element”. For color imagery, a Pixel consists of multiple color component samples. In subsampled color imagery, not all color component samples are present for each Pixel.

Pre-Charge: Compression schemes that exploit temporal redundancy between pictures are called intercoded compression schemes. To use an arbitrary picture of an intercoded sequence as the first displayed picture of an Essence Container, information from previous pictures may be required. Pre-Charge is a set of coded pictures that contains this information. Which and how many coded pictures constitute the Pre-Charge of the first displayed picture of an Essence Container depend on the compression scheme and on the type of the first picture within that compression scheme. See also Roll-Out.

Property: An information element of a Class. The information element is of a specific data type.

RIP: Random Index Pack. An optional table that contains the byte offsets of all Partitions and the Stream ID of the Essence Container they carry.

RGBA: Used to describe a representation of a picture as separate, non-subsampled components such as Red, Green and Blue with Alpha transparency (from where the abbreviation is derived). Other component options are also valid such a luminance with blue and red color difference components (YCbCr) and are still regarded by this standard as RGBA. The maximum number of components is 8. See Annex G.2.40.

["Note: **RP 210** definition: **text**"]": The definition of a term copied from the SMPTE Dictionary RP 210 circa 2004. 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.

Roll-Out: Inter-coded material frequently employs element reordering, i.e. the stored order differs from the temporal (or display) order. To display the last picture described by the Duration of the associated Essence Track additional Edit Units may be required at the end of the Essence Container in order to preserve the correct temporal order. Which and how many coded pictures constitute the Roll-Out depends on the compression scheme and on the type of the last displayed picture. The number of Edit Units (i.e. pictures) in the Roll-Out is the count of Edit Units in the Essence Container minus Origin minus the number of displayed pictures in the Essence Container. See also Pre-Charge.

Sample Rate: The rate of non-divisible, contiguously accessible units of the Essence stream of an Essence Element. See Annex F.2 and Annex G.2.2 for further information.

Sequence: An MXF Structural Metadata Class that is a Subclass of Structural Component. It shall be strongly referenced from a Track. It strongly references one or more Objects of other Subclasses of Structural Component in order to place them on the Track.

Set: A grouping of KLV items defined in SMPTE ST 336.

Set Key: A SMPTE Universal Label that constitutes the KLV Key of the Set. A 16-byte value that is registered in the register controlled by SMPTE ST 395.

SMPTE Universal Label: A fixed-length (16-byte) universal label, defined by SMPTE ST 298 and administered by SMPTE.

Source Clip: An MXF Structural Metadata Class that is a Subclass of Structural Component. It shall be strongly referenced by a Sequence and allows a portion of one Track to be referenced by another Track. For example, it may be used to place a portion of the stored Picture Essence described by a Top-Level File Package on the output timeline of a Material Package Track.

Source Package: A Subclass of Generic Package that strongly references a Descriptor. If the Descriptor is a File Descriptor, the Source Package is also called a File Package. If the Descriptor is a Physical Descriptor, the Source Package is also called a Physical Package.

Start Position: A Property of type Position. An integer number that defines the offset into a Track, relative to the Zero Point, in Edit Units of the Track. For an Essence Track, it defines the desired start point of the Essence.

Stream ID: A 32-bit integer that uniquely defines a data stream within the scope of a single MXF file. Examples for data streams are Index Table information (identified by IndexSID) and Essence Container data (identified by BodySID).

Strong Reference: A one to one Composition relationship between Objects where the referencing Object owns the referenced. They are typed. This means that the definition of the Composition identifies the kinds of Objects which may be the target of the reference.

In KLV-encoded MXF streams it is implemented as an aggregation by reference between Sets using UUIDs as identifiers.

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

Structural Component: Abstract Superclass of Sequence, Timecode Component and Source Clip. See Annex B.8.

Subclass: A Class that is directly or indirectly derived through Inheritance from another Class. The other Class is also called a Superclass of the Subclass.

Superclass: A Class from which, directly or indirectly, another Class is derived through Inheritance. The other Class is also called a Subclass of the Superclass.

Timecode: An annotation of elapsed time along a Track. This may be created to be numerically equal to the timecode time addresses of other standards such as SMPTE ST 12-1. Timecode can be used to convert a Position along a Track measured in Edit Units into a count of hours, minutes, seconds and frames.

Top-Level File Package: A File Package that is internal to the file and which is directly referenced by a Material Package of the file. This is the only type of File Package that may describe Essence stored in an internal Essence Container or in external files.

Top-Level Source Package: A Source Package that is internal to the file and which is directly referenced by a Material Package of the file. For all Generalized Operational Patterns all Top-Level Source Packages shall be File Packages (see Top-Level File Package).

Track: Subclasses of the Generic Track Class that define a discrete time axis. A Timeline Track is a Track that has temporally continuous Structural Components. For the Timeline Track Subclass, the distance between units along the time axis is defined as the inverse of the Edit Rate Property of the Timeline Track in seconds. An Event Track is a Track that has temporally discontinuous Structural Components. For the Event Track Subclass the distance between units along the time axis is defined as the inverse of the Event Edit Rate Property of the Event Track in seconds. For the Static Track Subclass, there is only one unit which applies to the entire duration.

UID: Unique Identifier. A generic term which may be used to refer to a UL, UUID, UMID etc.

UL Designator: The first 8 bytes of a SMPTE Universal Label. They identify the register, its minor and major versions, and may also convey other information that is used by the KLV encoding protocol. The register is the normative reference for the values of the 8 bytes.

Note: This is different from the definition of UL Designator in SMPTE ST 336. In that standard, the term UL Designator refers to only bytes 3 to 8 of the Label; the first two bytes (i.e. byte 1 and 2) are referred to as the Label Header.

UML: Unified Modeling Language. See <http://www.uml.org/>.

Unicode: A form of character coding that allows a wide range of characters and ideograms to represent most major languages. See ISO/IEC 10646.

Universal Set: A Set where each item in the Set is KLV encoded using its full Universal Label value. See SMPTE ST 336.

UTC: Coordinated Universal Time.

VBE: Variable Bytes per Element. Used to describe compression coding styles.

VBR: Variable Bit Rate. Used for Essence whose bit rate varies such that the number of bytes per Essence unit is not constant.

Video Interface: A defined Signal Scanning system. Usually a standard. Video Interfaces may be interlaced or progressive.

Weak Reference: An Association by reference relationship between Objects implemented in MXF as an aggregation by reference between Sets using UUIDs, ULs or UMIDs. Weak References are a one to many relationship. Weak References are typed. This means that the definition of the aggregation identifies the kinds of Objects (encoded as Sets) which is the target of the reference. See Aggregation. Weak References shall be Global Weak References or In-File Weak References.

Other normative parts of the MXF specification (see Section 5.3) may define additional kinds of Weak References.

Zero Point: Time zero on a Track or on a Package. On Timeline and Event Tracks, time is counted in multiples of 1/Edit Rate and 1/Event Edit Rate, respectively. See also the definitions of Event Origin and Origin. The Zero Point of a Package is the virtual synchronization point for all Tracks of the Package.

Simple Data Types that define relationships between Objects are defined in this section. All Reference kinds are typed. This means that the definition of the aggregation identifies the Classes of Objects (encoded as Sets) which may be the target of the Reference. Targets may also be sub-Classes of the specified type.

Several References of a given kind and the same type may be collected into an Array of References or a Batch of References.

Strong Reference: A Composition relationship between Objects implemented as an aggregation by reference between Sets using UUIDs as identifiers. Strong References indicate a one-to-one relationship. The targets of Strong References shall be contained within the same file. Strong References shall be typed.

In-File Weak Reference: An Association by reference relationship between Objects implemented in MXF as an aggregation by reference between Sets using UUIDs. In-File Weak References indicate a many-to-one relationship. In-File Weak References shall be typed. The targets of In-File Weak References shall be contained within the same file. All targets of a In-File Weak Reference of a given type shall be aggregated by Strong Reference in a single Property of a single Object within the file.

Package Reference: An Association by reference relationship between MXF Packages, implemented in MXF using Package IDs. Package References indicate a many-to-one relationship (one Package may be the target of many Package References). The targets of Package References need not be contained within the file.

Global Weak Reference: An Association by reference relationship between Objects, expressed via the value of an immutable Property and implemented in MXF using UUIDs. Global Weak References indicate a many-to-one relationship (one Object may be the target of many Global Weak References). Global Weak References shall be typed. The targets of Global Weak References need not be contained within the file.

Firm Reference (MetaReference, DictReference): An Association by reference relationship between Objects, implemented in MXF using AUIDs. Firm References indicate a many-to-one relationship (one Object may be the target of many Firm References). Firm References shall be typed. The targets of Firm References need not be contained within the file, however, all targets of a Firm Reference of a given type that are contained in the file shall be aggregated by Strong Reference in a single Property of a single Class within the file.

Kind	Used in MXF	Typed	Multiplicity	Targets must be in the same file	Targets must all be contained in single Property	Data type of the Identifier of the target
Strong Reference	Yes	Yes	One-to-One	Yes	Yes	UUID
In-File Weak Reference	Yes	Yes	Many-to-One	Yes	Yes	UUID
Package Reference	Yes	Yes	Many- to-One	No	Yes	PackageID
Global Weak Reference	Yes	Yes	Many- to-One	No	No	UUID or UL AUID
Firm Reference (Meta Reference) (Dict Reference)	Yes	Yes	Many- to-One	No	Yes	UUID or UL AUID AUID

4.2 Simple Data Types

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

AUID: A 16-byte UID that shall contain a UL or a UUID. If the value is a UUID, it shall be stored such that the top and bottom 8 bytes of the UUID are swapped. For UUIDs, this makes the most significant bit of the first byte a '1' and thus creates a UID value that is always distinct from a UL (see Section 4.2.1).

BER Length: BER encoded Length value of a KLV triplet. It represents the number of bytes of the Value of the KLV triplet.

Boolean: 1-byte value with the logical values FALSE and TRUE. The logical value FALSE shall correspond to the binary value 00h of the byte. All other values shall correspond to the logical value TRUE. MXF encoders shall emit the binary values 00h or 01h. MXF decoders shall be able to properly interpret all binary values ranging from 00h to FFh.

IDAUI: A 16-byte UID that shall contain a UUID or a UL. If the value is a UL, the upper and lower 8 bytes shall be exchanged. If the value is a UL, it shall be stored such that the top and bottom 8 bytes of the UL are swapped. For ULs, this makes the most significant bit of the ninth byte a '0' and thus creates a UID value that is always distinct from a UUID (see Section 4.2.1).

Note: This data type is not used in this document, but is provided to ensure other MXF documents can use it in a consistent way by referencing this document.

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 units of (1/Edit Rate) of a piece of Essence such as a Source Clip. Negative values are reserved for indicating Distinguished Values and shall not be used to indicate valid lengths.

Package ID: A basic UMID to uniquely identify a Package, or a value of 32 zero bytes used to terminate a reference chain.

Position: The Int64 value used to locate a specific temporal location along a Track. Since Properties of type Position are relative to a chosen Zero Point, negative values may occur.

StrongRef: A data type that implements a Strong Reference.

UInt8: Unsigned 8-bit integer.

UInt16: Unsigned 16-bit integer.

UInt32: Unsigned 32-bit integer.

UInt64: Unsigned 64-bit integer.

UL: SMPTE Universal Label (see SMPTE ST 298). For storage order, see Section 4.2.1.

Note: For specific Properties of type UL, other SMPTE Engineering Documents such as SMPTE ST 335, SMPTE ST 336, SMPTE ST 395 or SMPTE ST 400 define constraints and semantics additional to the ones defined by SMPTE ST 298.

UMID: Unique Material ID according to SMPTE ST 330. When used as a Package ID, only the 32-byte long Basic UMID shall be used.

Note: Other MXF specifications may also use the 64-byte long extended UMID, which consists of a 32-byte long basic UMID and 32 bytes of metadata.

UUID: Universally Unique Identifier according to IETF RFC 4122. For storage order, see Section 4.2.1.

Version Type: A UInt16 version number. The number is created from major and minor version numbers according to the equation $\text{major} \times 256 + \text{minor}$.

WeakRef: A data type that implements a Weak Reference.

4.2.1 Storage order of UL and UUID values

Properties of type UL shall only contain values of type UL and properties of type UUID shall only contain values of type UUID. Properties of type AUID and IDAU may contain values of type UL or UUID as described below.

The following table shows the value and storage orders of the Universal Label 06.0E.2B.34.01.01.01.01.07.02.01.01.01.04.00.00 as UL, AUID and IDAU.

When storing a Universal Label value in a Property of type UL, AUID or IDAU, MXF encoders shall use the storage order that is defined in Table 1.

Table 1 – Storage order of Universal Labels

Type	Storage order	Note
UL	06.0E.2B.34.01.01.01.01.07.02.01.01.01.04.00.00	The first bit is always 0 (zero).
AUID	06.0E.2B.34.01.01.01.01.07.02.01.01.01.04.00.00	The first bit is always 0 (zero).
IDAU	07.02.01.01.01.04.00.00.06.0E.2B.34.01.01.01.01	The 65 th bit is always 0 (zero).

The following table shows the value and storage orders of the Universally Unique Identifier 07.72.26.2E.76.55.43.6F.8F.F3.8A.C5.1B.77.1E.02 as UUID, AUID and IDAU.

When storing a Universally Unique Identifier value in a Property of type UUID, AUID or IDAU, MXF encoders shall use the storage order that is defined in Table 2.

Table 2 – Storage order of Universally Unique Identifiers

Type	Storage order	Note
UUID	07.72.26.2E.76.55.43.6F.8F.F3.8A.C5.1B.77.1E.02	The 65 th bit is always 1 (one).
AUID	8F.F3.8A.C5.1B.77.1E.02.07.72.26.2E.76.55.43.6F	The first bit is always 1 (one).
IDAU	07.72.26.2E.76.55.43.6F.8F.F3.8A.C5.1B.77.1E.02	The 65 th bit is always 1 (one).

Note: Some earlier, non-conformant encoder implementations wrote UL values into fields of type UUID or UUID values into fields of type UL. These values were either end-swapped as above, or unswapped. MXF decoders could still encounter files with such values.

4.3 Compound Data Types

Compound Data types that are created from combinations of simple data types are given in this section.

Array: A compound type comprising multiple individual elements where the elements are ordered, the type is defined, the count of items is explicit and the size of each item is fixed. The Array consists of a header of 8 bytes followed by the Array elements. The first 4 bytes of the header define the number of elements in the Array. The last 4 bytes of the header define the length of each element. Both of these values are represented as UInt32 coded in big-endian form. The order of the individual Array elements in the Array is significant. An Array is identified as an Array of Type, where Type is the defined type of the elements of the Array.

Note: Some MXF specifications may use the alternative identification <Type>Array.

Batch: A compound type comprising multiple individual elements where the elements are unordered, the type is defined, the count of items is explicit and the size of each item is fixed. The Batch consists of a header of 8 bytes followed by the Batch elements. The first 4 bytes of the header define the number of elements in the Batch. The last 4 bytes of the header define the length of each element. Both of these values are represented as UInt32 coded in big-endian form. The order of the individual Batch elements in a Batch is not significant. MXF encoders shall not place duplicate elements into a Batch. MXF decoders should be able to successfully decode MXF files which do contain Batches with repeated Batch element values. A Batch is identified as a Batch of Type, where Type is the defined type of the elements of the Batch.

Note: Some MXF specifications may use the alternative identification <Type>Batch.

DataStream: An ordered sequence of data or metadata elements. The type of the data or metadata elements and the length of the ordered sequence are 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).

String: Strings are concatenations of elements of a specific type. For example, this type could be characters defined either as ISO 7-bit characters that require 1 byte per character, or as Unicode UTF-16 characters that require 2 bytes or 4 bytes per character. In the case of UTF-16 characters expressing ISO 7-bit characters (see ISO/IEC 646), every character consists of 2 bytes where one is a null byte and the other 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 or other special 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 every field identifies a timestamp value of 'unknown'. This value should not be used unless unavoidable. Timestamp values shall be specified according to UTC time.

UTF-16: A variable-length (16 or 32 bits) character encoding. It is able to represent the complete Unicode Basic Multilingual Plane (BMP) with exactly 2 bytes and every other plane with exactly 4 bytes. It is defined in ISO/IEC 10646:2003, Annex Q. RFC-2781 also contains a description of the encoding.

ProductVersion: Comprises 5 values of UInt16 indicating major, minor, tertiary, patch 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 shall be 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.

Note: The structure of this ProductVersion data type is slightly different to that defined in SMPTE 377-2004, so great care is needed if an application performs a numerical comparison of ProductVersion Properties in files that may conform to different revisions of the specification.

4.4 Guide to the Use of KLV Pack and KLV Set Definition 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
-  Array of Strong Reference 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 Local Tag of the data when encoded as a KLV Local Set.

Item UL: The UL Designator and Item Designator portions of the UL as defined in SMPTE ST 336. It defines the SMPTE Metadata Dictionary entry. The normative definition of all 16 bytes of the ULs is given by SMPTE ST 335 (SMPTE RP 210).

This does not apply to the Set Key and Set Length. The Keys of MXF Sets and the ULs of Abstract Groups (Superclasses of MXF Sets) are defined by references in Table 17, Table 19 or Table 24, respectively.

Req?: Required status. The definition of whether an MXF encoder shall supply the Property when emitting a Set and the action an MXF decoder shall take when receiving a Set (see Table 3).

Note: Other sections of this document specify whether a Property must contain its correct value when supplied in a Set or whether it can contain a defined, invalid value.

Meaning: A description of the Property.

Default: If specified, the default value of an optional Property. If the optional Property has been omitted when instantiating a Set, the default value is the one that an application that requires the Property should use.

Table 3 – Meaning of Required status for MXF Encoders and MXF Decoders

Entry in Table	Abbreviation	MXF Encoder	MXF Decoder	SMPTE ST 395 status
Required	Req	Shall encode	Shall decode	Required
Encoder Required	E/req	Shall encode	May decode	Required
Decoder Required	D/req	May encode	Shall decode	Optional
Best Effort	B.Effort	Shall encode	Should decode	Required
Optional	Opt	May encode	May decode	Optional

The values of Opt and D/req Properties may only be encoded if their correct value is known by the application that creates or modifies the file. They shall not be encoded if their correct value is not known by the application that creates or modifies the file.

B.Effort Properties shall always be encoded.

Note 1: According to Section 6.2.3, they must have their correct value in Complete Partitions and must have either their correct value or their Distinguished Value in Incomplete Partitions.

Req Properties shall always be encoded.

Note 2: According to Section 6.2.3, they must have their correct value in Closed Partitions and may have a correct or incorrect value in Open Partitions.

5 Introduction (Informative)

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 documents define a Descriptive Metadata Scheme as 'plug-in' to the MXF File Format Specification.

One or more documents define an Application Metadata Scheme as 'plug-in' to the MXF File Format Specification.

A number of individual normative documents define both the Essence Containers that can be used in an MXF Body and the mappings of Essence Elements into an Essence Container (Section 10.1 below).

5.1 Structure of this Document

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

Section 6	Overall Specification: gives an overview of the whole MXF specification and partitioning rules.
Section 7	Partitions: gives details of the fine level structure of MXF including the Header Partition, the Body Partition(s) and the Footer Partition.
Section 8	Operational Patterns: shows how interchange can be aided by defining how the file is structured.
Section 9	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 10	Body: defines the requirements of the Essence Containers that can be used as an MXF file.
Section 11	Index Tables: defines how to locate Essence Elements within the Essence Container or within some types of external files.
Section 12	Random Index Pack: this defines how the Partitions within an MXF file can be rapidly accessed.
Annexes	Detailed byte values of Structural Metadata sets and Packages.

5.2 SMPTE 377M Revisions

SMPTE 377M was originally published in 2004. This specification is the second revision of the original document.

5.2.1 Major differences between SMPTE 377M-2004 and this specification (Informative)

SMPTE 377M-2004 has been revised by SMPTE to address a number of issues raised since its first publication in 2004. The purpose of this section is to highlight the major differences for the benefit of the reader. Only broad descriptions are provided in each case and this section does not purport to list every single change.

1. All SMPTE documents now include a 'Conformance Notation' section; in addition normative references and acronyms, terms and data types have been updated.
2. Section 4.2 defines two additional data types AUID and IDAU that can be used to store Universally Unique Identifier (UUID) values or SMPTE Universal Label (UL) values. The type AUID has been applied to the 16- byte UID values that are stored in the Primer Pack, to the 16-byte UID values that are stored in the Object Class Property defined in Annex A.1, and to the 16-byte UID values that are stored in the Base Class Property defined in Annex C.1.
3. Section 1 provides a table that lists the changes in version number values used in MXF files that comply with SMPTE 377M-2004 compared to the version defined by this document.
 - a. The value of the Minor Version Property of the Partition Pack number has been incremented to the value 0003h from the SMPTE 377M-2004 value of 0002h.
 - b. The value of the Version Property of the Preface has been incremented to 259 from the SMPTE 377M-2004 value of 258.
4. The new version of SMPTE 377M requires that encoders that comply with the new version of SMPTE 377M encode unique values for the IndexSID and BodySID Properties.
5. The KLV Fill item definition now clarifies the correct minor registry version byte value.

6. The definition of the Descriptive Metadata plug-in has been extended in a backwards compatible manner to include Properties that associate plug-ins to a specific Application Environments and that aide removability.
7. A backwards-compatible Application-Specific Metadata plug-in mechanism has been defined that allows MXF Sets to be extended with Properties that are identified for use in a specific Application Environment and that supports removability.
8. The MXF Track timing model has been clarified to avoid confusion.
9. Tracks are now defined in their general form and specific Track kinds (picture, sound, data etc.) are defined as a specialization of a particular Track.
10. The Key for the Index Table Segment has been extended to permit BER coding to overcome the previous limitation of a 64KB length.
11. The Index Table Segment definition has been extended by two new Properties to support indexing of external Essence files and to convey the size of the last VBE Element of an Essence Container.
12. A new Filler Class has been added to the definitions of Track metadata to provide for 'no content' such as silence on a sound Track.
13. A new Package Marker Object Class has been defined to provide 'mark-in' and 'mark-out' Properties on Material Packages.
14. The annex that describes the use Essence Descriptors has been comprehensively reworked to improve readability.
15. There is a new annex that defines all static Local Tag values and their associated Universal Label value.

In addition to the changes in the list above, there are a large number of detailed editorial changes to clarify provisions and to eliminate ambiguities that were found to exist in the SMPTE 377M-2004 document.

5.2.2 In-file version numbers

The table below lists the in-file version numbers for each of the SMPTE 377M document versions.

Group	Property	SMPTE 377M-2004 value	Current value	Defined in section
Partition Pack	Major Version	1	1	7.1
Partition Pack	Minor Version	2	3	7.1
Preface	Version	258 (i.e. 1.2)	259 (i.e. 1.3)	A..2
Preface	Object Model Version	1	1	A..2

5.2.3 Manipulating files conforming to other revisions of this specification (Normative)

This section defines rules for MXF encoders that manipulate and rewrite files that conform to versions of SMPTE ST 377-1 that differ from the version specified in this document.

An MXF encoder that implements this revision of SMPTE ST 377-1 shall manipulate and rewrite MXF files that comply to the 2004 version of SMPTE ST 377-1 such that all 2004 version numbers are preserved, unless it is certain that the new file complies with this revision of SMPTE ST 377-1. In the case that it is certain that the file complies with this revision of SMPTE ST 377-1, the MXF encoder should set all the version numbers in all Partitions of the file to the values specified in this document.

5.3 Parts of the MXF Specification

There are three types of SMPTE Engineering Documents: Standards, Recommended Practices (RP) and Engineering Guidelines (EG).

- A Standard is a document that states basic specifications, dimensions, or criteria that are necessary for effective interchange and/or interconnection within the system described. It can also define the functions necessary to achieve effective interchange among users. An SMPTE Standard contains Conformance Language.
- A Recommended Practice is a document that states basic specifications, dimensions, or criteria that are not necessary for effective interchange and/or interconnection, but that facilitate implementation of systems. Examples are conformance requirements for test materials or measurement methods; or constraints on existing specifications, dimensions, criteria, forms; and/or functions within the system described. An SMPTE Recommended Practice can contain Conformance Language.
- An Engineering Guideline is an informative document, often tutorial in nature, that incorporates engineering consensus on specifications, dimensions, and/or practices. It is intended to guide Users or to provide designs or procedures for Producers. It does not contain Conformance Language, normative text, or Normative References.

The suite of MXF specifications consists of all SMPTE Engineering Documents of the 377-X series and the following documents.

Note: Readers of this specification are advised to check whether SMPTE has published other Engineering Documents that form part of the MXF suite, but are not contained in the table below and are not part of the 377-X series of documents.

Document number	Document title
SMPTE ST 378	Material Exchange Format (MXF) – Operational Pattern 1a (Single Item, Single Package)
SMPTE ST 379-1	Material Exchange Format (MXF) – MXF Generic Container
SMPTE ST 379-2	Material Exchange Format (MXF) – MXF Constrained Generic Container
SMPTE ST 380	Material Exchange Format (MXF) – Descriptive Metadata Scheme-1
SMPTE ST 381-1	Material Exchange Format (MXF) – Mapping MPEG Streams into the MXF Generic Container
SMPTE ST 382	Material Exchange Format – Mapping AES3 and Broadcast Wave Audio into the MXF Generic Container
SMPTE ST 383	Material Exchange Format (MXF) – Mapping DV-DIF Data to the MXF Generic Container
SMPTE ST 384	Material Exchange Format (MXF) – Mapping of Uncompressed Pictures into the Generic Container
SMPTE ST 385	Material Exchange Format (MXF) – Mapping SDTI-CP Essence and Metadata into the MXF Generic Container
SMPTE ST 386	Material Exchange Format (MXF) – Mapping Type D-10 Essence Data to the MXF Generic Container
SMPTE ST 387	Material Exchange Format (MXF) – Mapping Type D-11 Essence Data to the MXF Generic Container
SMPTE ST 388	Material Exchange Format (MXF) – Mapping A-law Coded Audio into the MXF Generic Container
SMPTE ST 389	Material Exchange Format (MXF) – MXF Generic Container Reverse Play System Element
SMPTE ST 390	Material Exchange Format (MXF) – Specialized Operational Pattern “OP-Atom”
SMPTE ST 391	Material Exchange Format (MXF) – Operational Pattern 1b (Single Item, Ganged Packages)
SMPTE ST 392	Material Exchange Format (MXF) – Operational Pattern 2a (Play-List Items, Single Package)

SMPTE ST 393	Material Exchange Format (MXF) – Operational Pattern 2b (Play-List Items, Ganged Packages)
SMPTE ST 394	Material Exchange Format (MXF) – System Scheme 1 for the MXF Generic Container
SMPTE ST 405	Material Exchange Format (MXF) – Elements and Individual Data Items for the MXF Generic Container System Scheme 1
SMPTE ST 407	Material Exchange Format (MXF) – Operational Patterns 3a and 3b
SMPTE ST 408	Material Exchange Format (MXF) – Operational Patterns 1c, 2c and 3c
SMPTE ST 410	Material Exchange Format - Generic Stream Partition
SMPTE ST 422	Material Exchange Format – Mapping JPEG 2000 Codestreams into the MXF Generic Container
SMPTE ST 429-4	MXF JPEG 2000 Application
SMPTE ST 429-6	MXF Track File Essence Encryption
SMPTE ST 434	Material Exchange Format – XML Encoding for Metadata and File Structure Information
SMPTE ST 436	MXF Mappings for VBI Lines and Ancillary Data Packets
SMPTE RP 2008	Material Exchange Format – Mapping AVC Streams into the MXF Generic Container
SMPTE ST 2019-4	Mapping VC-3 Coding Units into the MXF Generic Container
SMPTE ST 2037	Mapping VC-1 into the MXF Generic Container
SMPTE EG 42	Material Exchange Format (MXF) – MXF Descriptive Metadata

5.4 The MXF File

An MXF file starts with a File Header, is followed by a File Body and is completed by a File Footer.

This document 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 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 can contain one or more Essence Containers. This document does not normatively define each individual Essence Container that can 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, of the existence of Descriptive Metadata plug-ins and of the Essence Containers in the File Body.

MXF Essence Containers can include picture, sound and data Essence as well as related metadata. All Essence and metadata types can 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 can be located outside the file.

Note: Embedding the Essence data in an MXF file can result in large MXF files, but ensures full interoperability. Locating the Essence data outside of the MXF file can reduce storage needs, but requires asset management to ensure the Essence can be reliably accessed when needed.

6 Overall Specification

This section defines the common parts of every MXF file. Details of each part are defined in later sections of this specification.

6.1 Overall Data Structure

The overall structure of an MXF File is shown in Figure 1 and Figure 2.

Note: The relative lengths of individual components shown in the figures are not to scale.

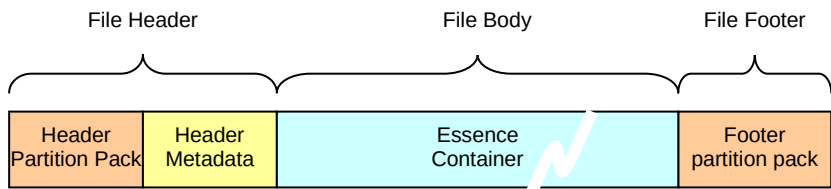


Figure 1 – Overall Data Structure of a simple MXF File

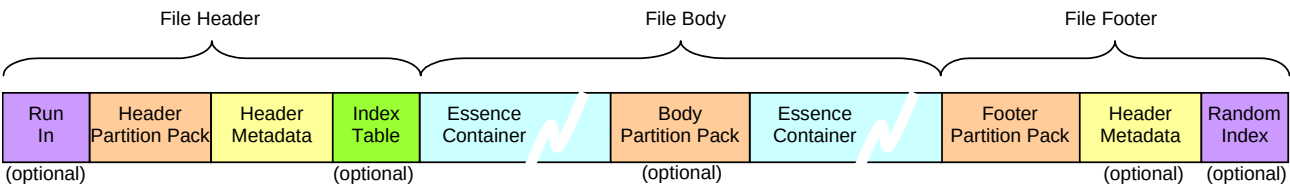


Figure 2 – Overall Data Structure of an MXF File with some optional components

There may be Fill Items in the file. The rules for the insertion of Fill Items are given in Section 6.4.

The MXF file shall be constructed as follows:

6.1.1 File Header

The File Header shall always be present at the start of every MXF file.

As shown in Figure 2, it may include a Run-In, it shall include a Header Partition Pack and Header Metadata and it may include an Index Table.

Note: Although the optional Run-In is a part of the File Header, it is not a part of the Header Partition (see Section 6.2.1).

6.1.2 File Body

The File Body provides the mechanism for embedding Essence data within MXF files.

The File Body shall contain zero or more Essence Containers.

If there is more than one Essence Container in the File Body, the Essence Containers shall be multiplexed together using Partitions (see Section 6.2).

For an Essence Element, there shall be an associated MXF Essence Container specification that defines how the Essence Element shall be KLV encoded in the Essence Container. An Essence Container specification may define how the Index Table specification shall be applied for indexing the Essence Element. An Essence Container specification shall identify the Essence Descriptors that are required to describe the Essence Element.

MXF applications can be created which embed all the Essence data within the file. Other applications might require some or all of the Essence data to be externally referenced. The mechanisms for this are further explained in Section 7, Section 8 and in Annex B.

MXF metadata-only files might have no File Body and hence no Essence Containers.

In the Operational Pattern definitions, there can be constraints placed on the capabilities of the File Body.

6.1.3 File Footer

The File Footer shall be located at the end of the file. As shown in Figure 2, the File Footer shall include a Footer Partition Pack. It may include a repetition of the Header Metadata and a Random Index Pack (see Section 12). 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 an Specialized Operational Pattern is used which defines 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.

6.2 Partitions

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

1. one Header Partition which shall be followed by
2. zero or more Body Partitions or Partitions of other type that are defined in other MXF standards, the last of which shall be followed by
3. zero or one Footer Partition

Note: SMPTE ST 410 defines another type of Partition that is call Generic Streams 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 zero or many different Essence Containers and Partitions help to manage them.

Note: SMPTE ST 410 defines another type of Partition that is call Generic Streams 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 zero or many different Essence Containers and Partitions help to manage them.

6.2.1 Partition Rules Overview

Each Essence Container shall be identified by a non-zero Stream ID value called the BodySID. This Stream ID shall be unique within each file.

Each Essence Container may be contained in a single Partition or may be segmented and distributed over two 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 be identified by a non-zero Stream ID value called the IndexSID. This Stream ID shall be unique within each file.

Note 1: SMPTE 377M-2004 compliant files exist that use the same non-zero Stream ID value as IndexSID and BodySID.

MXF decoders shall be able to decode MXF files, in which the same non-zero Stream ID value is used as IndexSID and BodySID.

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 within a Partition shall present the indexing data in the same order as the indexing data of an unsegmented Index Table.

Note 2: See Section 11.2.1 for further rules and guidelines on Index Table segmentation and repetition of Index Table Segments.

The EssenceContainerData Set is defined in Annex A.5. In this 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 shall apply:

1. 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.
2. 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.
3. If an application requires Index Table segments and Essence Container segments to be placed into the same Partition, then only Index Table segments that have the same BodySID Property value shall be permitted in the same Partition. Otherwise, Index Table segments and Essence Container segments should be placed into separate Partitions.

A Partition Pack shall define the start of every Partition. Every Partition shall be a Header Partition, a Body Partition or a Footer Partition, depending on where it is located within the file. For all three Partition kinds, 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 the first Partition of the file. A Header Partition shall start with a Header Partition Pack. The Header Partition Pack shall be followed by the Header Metadata; it may be followed by Index Table segments and a complete Essence Container or an Essence Container segment.

Only in defined specialized Operational Patterns, may the Header Partition be preceded by a Run-In as shown in Figure 2. A Run-In shall not be present in Generalized Operational Patterns.

If a file has Body Partitions, then each of the Body Partitions shall start with a Body Partition Pack. The Body Partition Pack may be followed by Header Metadata, Index Table segments and by a whole or segment of an Essence Container.

If a file has Partitions of another type that are defined in other MXF standards, then each of these Partitions shall start with a Partition Pack that is defined in that MXF standard. The rules for the contents and structure of the payload of these partitions shall be according to the provisions of the MXF standard that defines that type of Partition.

Note: SMPTE ST 410 defines another type of Partition that is call Generic Streams Partition.

Where present, the Footer Partition shall be located at the end of the file. It shall start with a Footer Partition Pack. The Footer Partition Pack may be followed by Header Metadata and by Index Table segments.

The Footer Partition may be followed by a Random Index Pack.

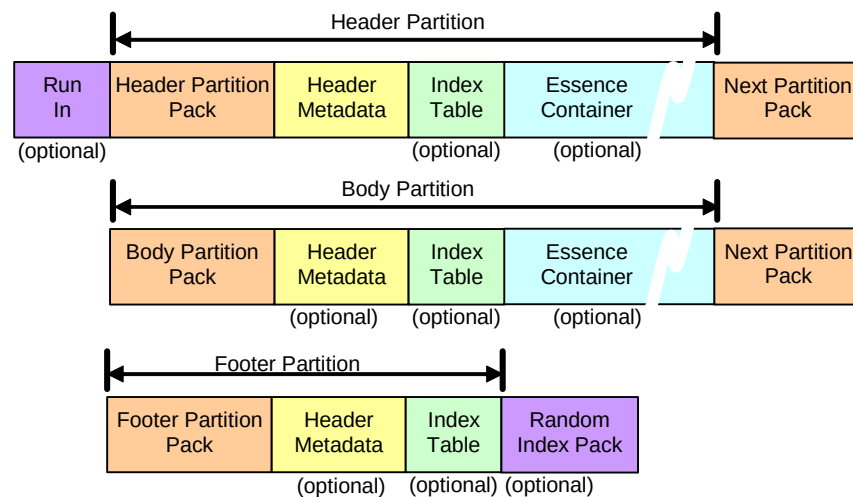


Figure 3 – Required order of file components in each Partition kind

6.2.2 Partition Rules Summary (Informative)

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 Stream ID called BodySID.
3. The Essence Container data can be segmented over one or more Partitions.
4. The Essence Container segments are placed 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 can be multiplexed.

Index Tables:

- i. For each Essence Container there can be an associated Index Table.
- ii. Each Index Table is identified with a Stream ID called IndexSID.
- iii. Each Index Table can be divided into Index Table segments which are distributed into one or more Partitions.
- iv. Index Table segments are placed in sequence within the file, though not necessarily in adjacent Partitions.
- v. Each Partition identifies its Index Table segment with the IndexSID value of the Index Table.
- vi. Partitions with different IndexSID values can be multiplexed.

Implementers need to be aware that, as illustrated in Figure 2 and Figure 3, the first Essence Container segment in the file can be located in the Header Partition. In files with no Body Partitions, the whole Essence Container will 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 could be placed in two large Partitions in which all the Type D-10 information follows all the DV information or alternatively the Partitions could be made smaller and multiplexed together. Providing the above rules are met, the file will be valid.

6.2.3 Partition Status

A Partition may be Open or Closed. A Partition may also be Complete or Incomplete.

The function of the Open versus Closed Partition status is to signal that the Partition contains a partial or the final version of the header metadata. The function of the Complete versus Incomplete Partition status is to signal whether all the Best Effort Property values are known at the time of creating the Header Metadata.

Section 7 defines the respective Partition Pack Key values for all three Partition kinds.

Open or Closed Status

Open An Open Partition shall be one in which required Header Metadata values have not been finalized (i.e. required values may be incorrect). They may not have the same values as in the Header Metadata of Closed Partition(s). The Header Metadata values of Open Partitions may differ between repetitions.

Closed A Closed Partition shall be one that has a Partition Pack where all values have been finalized and are correct and either contain

- 1. no Header Metadata or
- 2. Header Metadata where all required values have been finalized (i.e. all required metadata Properties shall be present and contain their correct value).

The Header Metadata in a Closed Partition shall carry identical information as the Header Metadata of any other Closed Partition containing Header Metadata such that:

- i. All Header Metadata Sets that are present in one Header Metadata instance shall be present in the other Header Metadata instance, but not necessarily in the same sequence.
- ii. All Properties that are present in a Header Metadata Set in one Header Metadata instance shall be present in the corresponding Header Metadata Set in the other Header Metadata instance, but not necessarily in the same sequence.

- iii. The value of each Property in one Header Metadata instance shall equal the value of the corresponding Property in the other Header Metadata instance except, possibly, the values of Instance UIDs and Strong Reference Properties.
- iv. The target Set of each Strong Reference in one Header Metadata instance shall be identical to the corresponding target Set in the other Header Metadata instance.

Incomplete or Complete Status

Incomplete An Incomplete Partition shall be one where both

- a. Header metadata shall exist and
- b. one or more Best Effort metadata Properties shall be identified as unknown by setting their values to the defined Distinguished Value.

Complete A Complete Partition shall be one with either

- (1) no Header Metadata or
- (2) where Header Metadata exists and all Best Effort metadata Properties contain their correct value.

Closed Partitions may be Complete or Incomplete.

Open Partitions may be Complete or Incomplete

6.2.4 The Status of an MXF File

A Closed MXF file shall be an MXF file which either has a Closed Header Partition or a Closed Footer Partition containing Header Metadata.

MXF files shall be Closed, except when they are being created or received (and in transfer), in which case they may be Open.

A file that conforms to all normative provisions of this and other applicable standards, but does not have a Closed Header Partition or a Closed Footer Partition containing Header Metadata shall be identified as an Open MXF file. 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 identify MXF files during the recording process.

MXF Decoders may attempt to decode Open MXF files. MXF Decoders shall not be required to be able to decode Open MXF files.

6.2.5 Header Partition

The Header Partition shall be the first Partition in any MXF file. There shall be only one Header Partition in an MXF file.

The Header Partition may be Open or Closed. The Header Partition may be Complete or Incomplete.

6.2.6 Body Partition

There shall be zero or more Body Partitions in an MXF file.

A Body Partition may be Open or Closed. A Body Partition may be Complete or Incomplete.

Note 1: There are many applications for Body Partitions, including the ability to multiplex different Essence Containers, to provide a segmentation mechanism for Index Tables, and to assist recovery from incomplete transfers.

Essence Container segments in Body Partitions shall start and end boundaries of Edit Units of stored Essence of the Essence Containers.

Note 2: As a consequence of this, an Essence Container segment in a Header Partition will also end at a boundary of an Edit Unit of stored Essence.

An example structure of an MXF file with a single Body Partition is shown in Figure 4. An example with more than one Body Partition is shown in Figure 5.

6.2.7 Footer Partition

If present, the Footer Partition shall be the last Partition in any MXF file. There shall be no more than one Footer Partition in an MXF file.

A Footer Partition shall be Closed as defined in Section 6.2.3. A Footer Partition shall be complete or incomplete as defined in Section 6.2.3.

Note 1: Per Section 6.2.1, a Footer Partition does not contain Essence Container Data. Section 7.4.2 defines that the BodySID value in the Partition Pack must always equal zero (0).

Note 2: Section 7.4.2 defines that the FooterPartition value in the Footer Partition Pack must always have the correct value.

The Footer Partition shall be present unless a specialized Operational Pattern is used which requires it to be absent or defines it as optional. In these cases, 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 an Footer Partition shall be indicated by a non-zero value of the FooterPartition Property in any closed Partition Pack within the file.

6.2.8 Using Partitions to multiplex Essence Containers and associated Index Tables (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.

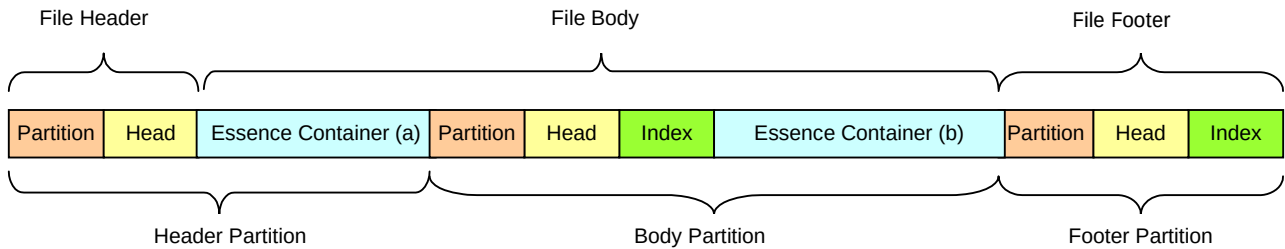


Figure 4 – MXF File Containing One Essence Container

In more complex MXF files, there can 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 can 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. The insertion of additional Body Partitions permits Header Metadata repetition, the insertion of Index Table segments and the multiplexing of different Essence Containers.

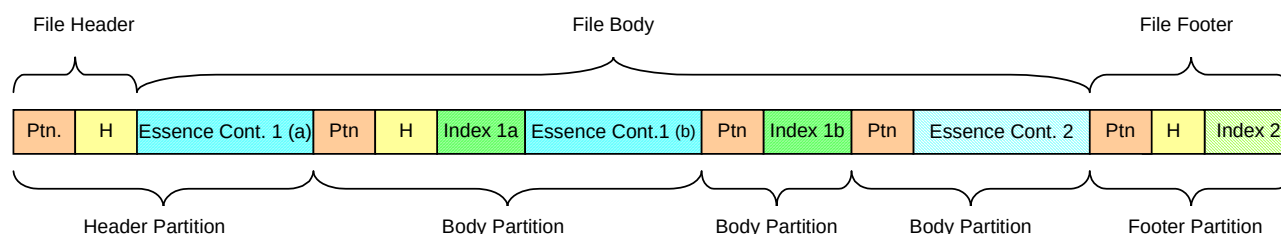


Figure 5 – MXF File Containing Two Essence Containers

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 6.2.1 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.

6.3 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 ST 336. This section defines KLV coding issues specific to the implementation of MXF.

Note: SMPTE ST 336 specifies that KLV parsers can ignore the version number or use the version number as an additional guide and consistency check in the process of parsing a KLV Key.

6.3.1 KLV Coding Sequence

MXF files shall consist of 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 shall be defined in individual Essence Container specifications.

6.3.2 KLV Coded Dark Components

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

MXF decoders shall always follow the KLV syntax whether or not the Key is recognized. For unrecognized Keys (i.e. Keys having values that cannot be interpreted by the decoder) the decoder shall use the KLV length field to skip over the Value of the data packet and continue processing at the start of the next KLV key.

Decoders may signal the existence of unrecognized KLV keys for diagnostic purposes. Decoders may pass the unknown KLV packets to higher level application for processing.

6.3.3 KLV Fill Items

There are a number of areas where an MXF file may use the KLV Fill item. This item is defined in SMPTE ST 336 (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:

1. To pad the Header Metadata so that the following KLV packet lies on a convenient storage boundary for ease of access.
2. 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.
3. To pad the Header Metadata to allow space for an optional Index Table to be inserted before the first Body item.
4. To pad the end of a Edit Unit of stored Essence to a known KAG boundary thereby ensuring the next Edit Unit of stored Essence is aligned to the next KAG boundary as described in Section 6.4.1.
5. To pad each Edit Unit of stored Essence 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 should not be placed adjacent to each other in a file. Any contiguous KLV Fill items should 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) as 06.0e.2b.34.01.01.01.**02**.03.01.02.10.01.00.00.00.

Note: Some earlier encoder implementations wrote KLV Fill items with a key that differed from the one published in SMPTE RP 210. This difference was limited to the version number byte (byte 8) and the resulting key has the value 06.0e.2b.34.01.01.01.**01**.03.01.02.10.01.00.00.00.

MXF decoders shall ignore the version number byte (i.e. byte 8) when determining if a KLV key is the Fill item key.

6.3.4 KLV Lengths

MXF encoders may use long or short form BER encoded lengths as specified by SMPTE ST 336. MXF encoders shall not use long-form coding that exceeds a 9-byte BER encoded length. MXF encoders may use long-form BER encoding even for length values less than 128.

The BER encoded length token of “80h” is defined to indicate “unspecified length” This value shall not be used in MXF files.

MXF decoders shall be SMPTE ST 336 compliant. They shall respond to short or long-form BER encoding. MXF decoders shall not rely on a fixed number of bytes for length fields.

Other MXF specifications (e.g. Essence Container specifications) may define additional constraints for KLV Lengths.

6.3.5 Local set Lengths

MXF encoders shall use SMPTE ST 336 compliant Keys for Local Sets.

MXF decoders shall be able to decode the syntax of

1. Local Sets coded with 2-byte Tags and 2-byte Lengths,
2. Local Sets coded with 2-byte Tags and BER Lengths

MXF decoders should be able to decode all Local Set encodings specified in SMPTE ST 336.

6.3.6 Variable-Length Pack Lengths

MXF encoders shall use SMPTE ST 336 compliant Keys for Variable-Length Packs.

MXF decoders shall be able to decode the syntax of

1. Variable-Length Packs coded with 2-byte Lengths,
2. Variable-Length Packs coded with BER Lengths

MXF decoders should be able to decode all Variable-Length Pack encodings specified in SMPTE ST 336.

6.3.7 Defined-Length Pack Lengths

MXF encoders shall use SMPTE ST 336 compliant Keys for Defined-Length Packs.

MXF decoders shall be able to decode the syntax of Defined-Length Packs as specified in SMPTE ST 336.

Note 1: According to SMPTE ST 336, length values are not encoded for Defined-Length Pack items.

Note 2: Individual items within a Defined-Length Pack can have length values which need to be determined by parsing the item, thus resulting in a Pack with a defined yet variable overall length. There is no requirement in SMPTE ST 336 for Defined-Length Packs to have fixed, constant length values.

6.3.8 MXF Keys and Universal Labels

MXF files shall have KLV Keys and ULs that are 16 bytes long. No other lengths shall be permitted.

6.3.9 Constraints on recursive Groupings of KLV items

SMPTE ST 336 places no limits on the depth of recursive grouping allowed in KLV coded data sets (i.e. sets contained within sets) to represent hierarchical structures.

MXF files shall only contain KLV Sets or Packs with individual data items and shall not apply recursive grouping, unless specified in an Application-Specific Metadata, Descriptive Metadata or Essence Container specification.

Note: The use of recursive group structures could prevent the data from being decoded by many implementations.

Recursive group structures shall not be used in MXF Structural and MXF Descriptive Metadata Sets (see Sections 9.8.2 and 9.8.3).

Parts of an MXF file (such as the Header Metadata) may contain logical groupings of Objects that represent a hierarchical structure. The method of encoding such structures in the MXF File format shall be to provide each Object with an Instance UID Property and to encode the Object as a Pack or Set.

The Instance UID Property provides each Object in the logical grouping with a unique identification. This allows any Object to be referenced from any other Object. 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 9.3 describing the Header Metadata provides more details of coding logical data structures.

6.3.10 The Primer Pack, Dark Metadata and MXF extensions

MXF encoders that extend Header Metadata using MXF Local Set encoding shall use the Primer Pack for generating and transmitting Local Tags. This ensures that no two extension Properties of an Object encoded as an MXF Local Set will use the same Local Tag, but represent different dictionary items.

Full details of the Primer Pack are given in Section 9.2.

Metadata extension schemes may be added using other KLV encoding mechanisms such as SMPTE ST 336 Universal Sets or Local Sets with different sized Local Tags and lengths. Applications that define Local Set encodings with 1 or 4 byte long Local Tags shall define how the Local Tags are transformed into the corresponding 16 byte UL.

Note: The use of MXF Sets with 1 and 4 byte Local Tag encoding enables applications to use their static Local Tags without conflicting with the 2 byte Local Tag ranges and allocations defined in Section 9.2.2. However, MXF decoders are not required to decode the syntax of such non-MXF Header Metadata extension schemes.

Section 9.7 defines an optional method to safely extend MXF Header Metadata and to facilitate simple removal of Header Metadata extensions that use the Application-Specific Metadata plug-in mechanism. Other SMPTE Engineering documents may define additional extension methods.

In order to reduce interoperability problems resulting from unmanaged independent extensions, Header Metadata extensions should use an extension method that is defined in this or other SMPTE Engineering documents.

6.4 MXF Encoding Requirements

The contents of an MXF file can be viewed as a multiplex of data items or Objects:

1. Run-In – only used in specialized Operational Patterns.
2. 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.

3. Header Metadata – metadata describing the File Body, its Essence, its structure and possibly its meaning.
4. Index Tables – to convert from a position in an Essence Track to a byte offset within the Essence Container.
5. Essence Containers (when present) - the Essence data of the File Body.
6. Random Index Pack – used to rapidly locate the Partitions in the file.

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.

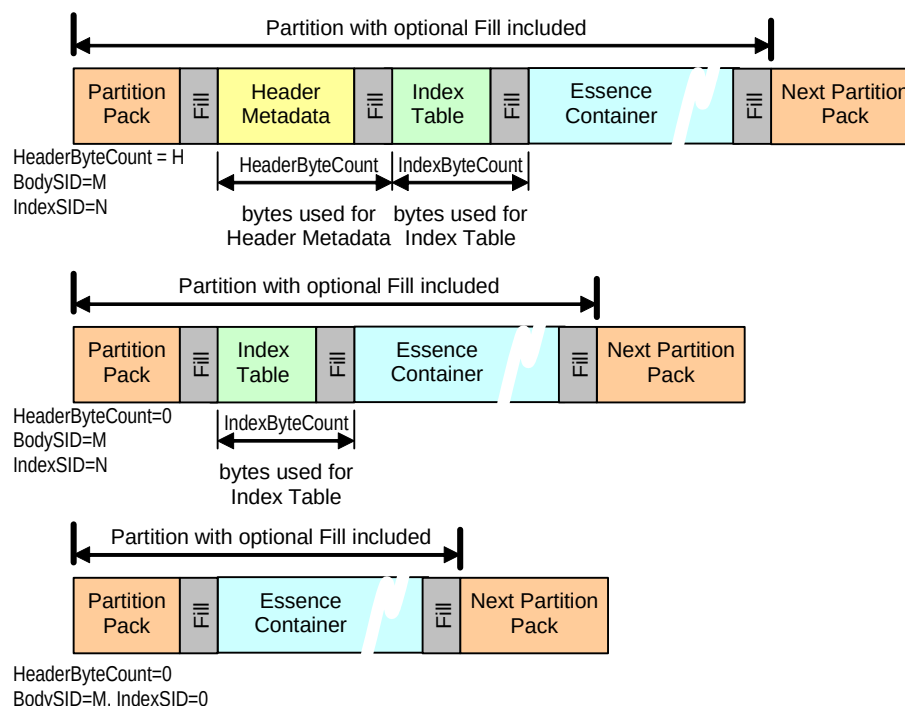


Figure 6 – Ordering in a Partition and the use of Fill Items

The items of a Partition shall follow the order of the items defined below:

- i. A Partition Pack, which may be followed by a single KLV Fill item to ensure grid alignment or to provide padding.
- ii. Optional Header Metadata. If present, it may be followed by a single KLV Fill item to ensure grid alignment or to provide padding.
- iii. Zero or more Index Table segments. Each Index Table segment may be followed by a single KLV Fill item to ensure grid alignment or to provide padding.
- iv. Zero or more bytes of Essence Container data. If Essence Container Elements are present, KLV Fill items may be added to provide grid alignment or padding as required by the Essence Container.

Within a Partition, the Essence Container data shall start at the first byte of the first Essence key. The Essence Container data (including any KLV Fill items) shall end immediately before the first byte of the Partition Pack of the next Partition.

6.4.1 KLV Alignment Grid (KAG)

In certain applications, it is desirable to align certain KLV elements to specific byte boundaries. In MXF, this is achieved by the insertion of KLV Fill items that ensure that the desired KLV elements are aligned. One example is a VBR picture stream with a peak rate of 1Mbyte/picture. It could 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 KAG 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 shall be the first byte of the Key of the Partition Pack that defines that Partition.

Specific KLV items that should be aligned to a gridline include the Preface Set (of the Header Metadata) and the Index Table Segment Set and the start of each Content Package in the Essence Container.

Individual Essence Container specifications may define which KLV elements shall be aligned to 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}) shall not be used.

Note 1: The upper limit of 1 Mbyte is provided to minimize receiver buffer requirements.

If grid alignment is used, then all Partitions in the file should use grid alignment. The KAG Size is specified in every Partition Pack and is valid for that Partition only. Specific applications may require a different KAG size value for different Essence Containers.

For all Partitions with segments from the same Essence Container, the KAG size shall be constant or zero (0). Essence Container specifications may require a non-zero value for all Partitions. In that case, this is the value that shall be used.

Example: An MXF file that has a high resolution primary video and an alternate low resolution preview video could 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 zero (0).

The KAG size parameter is a performance enhancement parameter. It is possible that some files may have the KAG value incorrectly set. All MXF decoders shall correctly parse a file whether the KAG parameter and the grid alignment are correct or not.

Note 2: Application writers need to be aware that incorrect values of KAG size could lead to performance degradation, especially when MXF files are transferred to and from certain classes of machine. Applications can also be degraded by the requirements of KAG implementation. Its use is therefore optional.

6.4.2 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 Essence Container specification.

For all other MXF specifications the following rules shall apply:

1. 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.
2. Compound data types shall have all component multi-byte values coded as big-endian.

6.4.3 Encoding Constraints

This document defines the general rules of MXF encoding. Other documents, such as Operational Pattern and Essence Container specifications may further constrain the rules of MXF encoding for reasons particular to their requirements.

6.5 Run-In Sequence

In defined 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.

MXF decoders shall ignore the Run-In sequence and parse the data until either the first 11 bytes of the Partition Pack label have 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.

Note: 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 provided it conforms to the above provisions, and any provisions of the respective specialized Operational Pattern specification.

6.6 Minimum MXF Decoder (Informative)

The concept of a minimum decoder is useful in defining the minimum behavior of any device which can be claimed 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 MXF files might 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 Operational 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).

6.7 Strong and Weak Reference Integrity (Informative)

MXF decoders might not be able to find the target of all references in an MXF file. This could be because the target Set is Dark to the decoder or because the target of the reference is external to the file. Under these circumstances, the following observations can be helpful:

1. Strong References are made from a Property of type StrongRef, Array of StrongRef or Batch of StrongRef in one Header Metadata Set to the Instance UID Property in another Header Metadata Set.
2. All Header Metadata Sets are strongly referenced (directly or indirectly) from the Preface Set.
3. All Strong References in a file match one and only one Header Metadata Set.
4. Weak References are made from a Property of type WeakRef, Array of WeakRef or Batch of WeakRef in one Header Metadata Set to the appropriate Property in another Header Metadata Set. This Property is of a unique identifier type such as UL, UUID, AUID, IDAU or UMID.
5. Weak References can be made to "global definitions" which might 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, it can be regarded as a global definition (i.e. a Global Weak Reference).
6. Parts of the Header Metadata can be Dark Metadata for an MXF decoder. This means that both Strong and Weak References can appear unresolved even though they point inside the Header Metadata and are correct. MXF Decoders need to be able to cope with this, i.e. they must not fail when faced with unresolved references.

7 Partitions

7.1 Partition Pack

Partition Packs shall be KLV coded, Defined-Length Packs as defined in SMPTE ST 336. Header Partition Packs, Body Partition Packs and Footer Partition Packs, whether Open or Closed, Complete or Incomplete, shall be variants of the Partition Pack.

The Key of the Partition Pack is given in Table 4.

Table 4 – 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	Defined-Length Packs (no length fields)
7	Structure Designator	01h	Set / Pack registry
8	Version Number	vvh	Registry Version in which the specific Key first appeared
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 7.2, 7.3, 7.4	Header Partition, Body Partition or Footer Partition
15	Partition Status	Sections 7.2, 7.3, 7.4	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 Complete or Incomplete Header Metadata as defined in Section 6.2.3.

Other MXF standards that define other types of Partitions may define additional values of byte 15.

Note: SMPTE ST 410 is such an MXF standard.

The data in a Partition Pack is defined in Table 5 and the text which follows the table. As defined by SMPTE ST 336, the values in all Partition Packs must appear in the order presented in Table 5.

Table 5 – Partition Pack

Item Name	Type	Len	Item UL	Meaning	Default
Partition Metadata	Pack Key	16	Table 4	Identifies a Partition Pack	
Length	BER Length	var (see 9.3)		Overall Length of Partition Pack	
Major Version	UInt16	2	06.0E.2B.34 01.01.01.04 03.01.02.01 06.00.00.00	Major Version number of MXF byte-level format (non-backwards compatible version number). The value shall be set to 0001h. [RP 210 A major version number. A change in a major version implies non-backwards compatibility]	
Minor Version	UInt16	2	06.0E.2B.34 01.01.01.04 03.01.02.01 07.00.00.00	Minor Version number of MXF byte-level format (backwards compatible version number). The value shall be set to 0003h. Note 1: For SMPTE 377M-2004 compliant files, the value of this Property is 0002h. Note 2: See 5.2.3 for provisions how to set the value when manipulating files that do not comply with this revision of SMPTE ST 377-1. [RP 210 A minor version number. A change in a minor version implies some measure of backwards compatibility]	
KAGSize	UInt32	4	06.0E.2B.34 01.01.01.05 03.01.02.01 09.00.00.00	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	06.0E.2B.34 01.01.01.04 06.10.10.03 01.00.00.00	The number of this Partition in the sequence of Partitions (as a byte offset relative to the start of the Header Partition). [RP 210 The current number in a sequence]	
PreviousPartition	UInt64	8	06.0E.2B.34 01.01.01.04 06.10.10.02 01.00.00.00	The number of the previous Partition in the sequence of Partitions (as a byte offset relative to the start of the Header Partition). [RP 210 The previous number in a sequence]	0
FooterPartition	UInt64	8	06.0E.2B.34 01.01.01.04 06.10.10.05 01.00.00.00	The number of the Footer Partition (as a byte offset relative to the start of the Header Partition). [RP 210 The last number in a sequence]	0
HeaderByteCount	UInt64	8	06.0E.2B.34 01.01.01.04 04.06.09.01 00.00.00.00	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 KLV Fill item 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	06.0E.2B.34 01.01.01.04 04.06.09.02 00.00.00.00	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 KLV Fill item which is included in the IndexByteCount. See Figure 6.	0

Item Name	Type	Len	Item UL	Meaning	Default
				[RP 210 Count of bytes used for index table segments]	
IndexSID	UInt32	4	06.0E.2B.34 01.01.01.04 01.03.04.05 00.00.00.00	Index Table Segment Identifier in this Partition. The value zero (0) defines that there is no Index Table segment in this Partition. [RP 210 Index table stream ID]	
BodyOffset	UInt64	8	06.0E.2B.34 01.01.01.04 06.08.01.02 01.03.00.00	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	06.0E.2B.34 01.01.01.04 01.03.04.04 00.00.00.00	Identifier of the Essence Container segment found in this Partition. The value zero (0) indicates there is no Essence Container data in this Partition. [RP 210 Essence (or its container) stream ID]	
Operational Pattern	UL	16	06.0E.2B.34 01.01.01.05 01.02.02.03 00.00.00.00	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	Batch of UL (Essence Containers)	8+ 16n	06.0E.2B.34 01.01.01.05 01.02.02.10 02.01.00.00	A 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 the Registry defined by SMPTE ST 400 (RP 224) [RP 210 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 of a file. Minor Version shall be used to indicate the precise revision of this document to which the file complies.

ThisPartition shall specify 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 shall be zero.

Note 1: This definition means that a change in the size of any Run-In will not affect the addresses expressed in byte offset values for the rest of the file.

PreviousPartition shall specify the byte offset of the start of the previous Partition relative to the start of the Header Partition.

FooterPartition shall specify 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 Sections 7.2.2, 7.3.2 and 7.4.2).

HeaderByteCount shall specify the number of bytes used for the Header Metadata in this Partition. The count shall start at the first byte of the key of the Primer Pack and shall end after the last byte of the value of the last Header Metadata Set in this Partition, including any trailing KLV Fill item which immediately follows the last Header Metadata Set in this Partition (see Figure 6). The value shall be zero if there is no Header Metadata in this Partition.

IndexByteCount shall specify the number of bytes used for Index Table Segments in this Partition. The count shall start at the first byte of the key of the first Index Table segment and shall end after the last byte of the

last Index Table segment in this Partition, including any trailing KLV Fill item which immediately follows the last Index Table segment in this Partition (see Figure 6). The value shall be zero if there is no Index Table segment in this Partition.

IndexSID shall specify the Stream ID 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 shall specify 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 shall include any KLV Fill items that are part of the Essence Container.

Note 2: This value provides a tool for using Index Tables and for recovering partial file transfers.

MXF decoders should use this value to reset any internal stream offset count for the current Essence Container to this value when reading the Partition Pack.

Note 3: This will help decoders to recover from damaged files.

BodySID shall specify the Stream ID of the Essence Container in this Partition. The value shall be zero if there is no Essence Container data in this Partition.

Operational Pattern shall be a UL that identifies the Operational Pattern for this MXF file (see Section 8).

EssenceContainers shall be a Batch of ULs that identifies all Essence Containers used in this MXF file. For any given Essence 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. If this Partition is Open, the values should be complete and correct.

Note 4: The length of the Partition Pack includes the length of this Batch. MXF files with different Essence Containers can have different lengths of Partition Pack.

This Batch shall contain all values that appear in the Essence Container Property of all File Descriptors (including the Multiple Descriptor) of all Top-Level File Packages in the File that describe internal Essence.

Note 5: This definition of the contents of EssenceContainers in the Partition Pack differs from the definition of the contents of EssenceContainers in the Preface Set in Annex A.2.

Note 6: According to the terminology in SMPTE ST 336 the Partition Pack is a Defined-Length Pack.

Note 7: 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 Header Metadata but no Essence or that the Essence is external.

7.2 Header Partition Pack

In the default case of a Run-In sequence length of zero, the file shall start with the Header Partition Pack.

7.2.1 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 6 – Specification of the Header Partition Pack Key

Byte No.	Description	Value (hex)	Meaning
1-13	See Partition Pack (Table 4)	–	Refer to Table 4
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 4)	–	Refer to Table 4

The Header Partition Pack status byte 15 shall be set according to the definitions in Section 6.2.3.

7.2.2 Header Partition Pack Values

The value of the **FooterPartition** Property shall be as defined in Section 7.1 for Closed Partitions. In Open Partitions, the value shall be as defined in Section 7.1 or zero (0). If the Footer Partition is not present in the file then the value of this Property shall be zero (0).

Note: Given that the value of the FooterPartition Property can be zero, MXF decoders cannot rely on the FooterPartition value in the Header Partition Pack to locate the Footer Partition.

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

Header Metadata shall be present 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.

Essence Container data is optional in the Header Partition. The **BodySID** value shall be zero if there is no Essence Container data in this Partition and non-zero if an Essence Container is present in this Partition.

7.3 Body Partition Pack

Zero or more Body Partitions may appear in the Partition multiplex of the file. Where a Body Partition is used, the Partition shall start with a Body Partition Pack defined as follows:

7.3.1 Body Partition Pack Key

The Body Partition Pack Key shall have the following value:

Table 7 – Specification of the Body Partition Pack Key

Byte No.	Description	Value (hex)	Meaning
1-13	See Partition Pack (Table 4)	–	Refer to Table 4
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 4)	–	Refer to Table 4

The Body Partition Pack status byte 15 shall be set according to the definitions in Section 6.2.3.

7.3.2 Body Partition Pack Value

The value of the **FooterPartition** Property shall be as defined in Section 7.1 for Closed Partitions. In Open Partitions, the value shall be as defined in Section 7.1 or zero (0). If the Footer Partition is not present in the file then this Property shall be zero.

Note 1: Given that the value of the FooterPartition Property can be zero, MXF decoders cannot rely on the FooterPartition value in the Body Partition Pack to locate the Footer Partition.

The value of the **PreviousPartition** Property shall be correctly completed.

Note 2: 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.

7.3.3 Header Metadata Repetition in Body Partitions

The Header Metadata may be repeated in the Body Partition following the rules set out in Section 7.5.

7.4 Footer Partition Pack

The Footer Partition shall start with a Footer Partition Pack.

7.4.1 Footer Partition Pack Key

The Footer Partition Pack Key shall have the following value:

Table 8 – Specification of the Footer Partition Pack Key

Byte No.	Description	Value (hex)	Meaning
1-13	See Partition Pack (Table 4)	–	Refer to Table 4
14	Partition Kind	04h	MXF Footer Partition
15	Partition Status	02h 04h	Closed and Incomplete Closed and Complete
16	See Partition Pack (Table 4)	–	Refer to Table 4

The Footer Partition Pack status byte 15 shall be set according to the definitions in Section 6.2.3.

Note: Open Footer Partitions are not permitted.

7.4.2 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.

Note: If there are no Body Partitions in the file, then the value will be zero because 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.

7.4.3 Header Metadata Repetition in the Footer Partition

The Header Metadata may be repeated in the Footer Partition following the rules set out in Section 7.5.

7.5 Header Metadata Repetition in Body and Footer Partitions

Header Metadata may be repeated in Body Partitions and in the Footer Partition.

Some of the purposes of Header Metadata repetition in the file are to support the recovery of critical metadata in transfer applications where the file may be interrupted, in storage applications where parts of the file may be corrupted, or in applications where MXF decoders need to synchronize with the byte stream in mid-transfer. Examples for such applications are the transfer of an MXF file as a stream over a unidirectional link or the storage of MXF files in tape streamers.

If the Header Partition is Open, each repetition of the Header Metadata in a Body Partition or the Footer Partition shall be an identical (see Section 6.2.3) or updated copy. In the repetition, Duration Properties and other Header Metadata Properties may be changed to reflect conditions at the time of writing.

Note 1: Section 6.2.4 defines that there must be at least one Closed Partition in an MXF file that contains Header Metadata, and that if the Header Partition is Open one of these Partitions must be the Footer Partition.

Note 2: Section 6.2.3 defines that the Closed Partition(s) can be Incomplete.

Note 3: Section 6.2.3 defines that the Header Metadata of all Closed Partitions that contain Header Metadata must carry identical information.

MXF encoders shall identify updated repetitions according to the rules defined in Section 7.5.2.

MXF decoders should always use Header Metadata from a Closed Partition. When processing files that contain updated Header Metadata repetitions and when a Closed Partition containing Header Metadata is not available, MXF decoders should use the repetition of a Header Metadata Set with the Generation UID value that equals the This Generation UID Property of the Identification Set at the highest index in the Identifications Property of the Preface Set. All other versions of the same Header Metadata Set should be considered outdated.

7.5.1 Application Guidelines for header Metadata Repetition (Informative)

This subsection defines how an MXF application can behave when using Header Metadata repetitions. The MXF application could be an encoder, a decoder or some other process that modifies MXF files.

If an MXF file has an Open Header Partition then the Footer Partition must contain updated metadata (see Section 6.2.4). Therefore, MXF decoders can, if possible, use the Header Metadata in the Footer Partition for files with Open Header Partitions.

It is not practical to list all the situations under which the MXF Footer might not be available to an MXF application. In the event that the Header Partition is Open and the Footer Header Metadata is not available or is not easily accessible, MXF decoders can use a repeated Header Metadata from a Body Partition or from the Open Header Partition. This can occur in the case of partial transfer or Pre-Play (where a file is played or transferred before recording is finished).

When devices are creating MXF files, it is advisable that Partitions are Closed wherever possible. It is also advisable that Closed Partitions are Complete wherever possible.

In the case of recording on streaming linear media, Header Metadata repetition could increase the speed of data recovery. In such applications, Header Metadata repetitions can be recorded as part of the file. It is advisable that these Header Metadata repetitions are updated to represent the current information available to the recorder.

Note: According to Section 6.2.3, the Header Metadata instances in all Closed Partitions must be identical.

When the incoming MXF file has been received in its entirety, the Header Metadata repetitions within the file can be either updated to match the Closed Partition values or they can be removed (possibly by overwriting with a KLV Fill item) from the data stream. Users and manufacturers need to be aware that Open repetitions of Header Metadata can result in inconsistencies if any of the values in the Closed Header Metadata are not copied to all other instances of Header Metadata in the file.

Where practical, MXF decoders can indicate by an appropriate means, that the Header Metadata instance in use is an Open Header Metadata instance, either from the Header Partition or from a Body Partition.

7.5.2 Tracking Changes with Generation UID

When an MXF file is initially created, a single Identification Set shall be added giving details of the application or device that created the file.

Metadata Sets that do not have the optional Generation UID Property shall be associated with the first Identification Set in the Array of the Identification Property of the Preface Set.

The Generation UID Property of a Header Metadata Set may only be omitted if the values of all Properties of the Set — other than Instance UIDs and the Strong or Weak References pointing to a modified Instance UID of another Set — remain unchanged since the initial creation of the MXF file; the Generation UID Property shall be present in all other MXF Header Metadata Sets, except the Identification Set.

Each subsequent modification to Property values of the Header Metadata other than Instance UIDs and the corresponding Strong References shall be identified according to the following rules:

1. If the application or device that modifies the Header Metadata is the same as described in the last Identification Set, and the contents of the file have not been modified or read by any other application or device, then the application or device may update the value of the Modification Date of the last Identification Set and, if there is more than one Identification Set in the file, set the value of the Generation UID of each modified MXF Set to the value of the This Generation UID Property of the last Identification Set.
2. If the application or device that modifies the Header Metadata is not the same as described in the last Identification Set or if the contents of the file have been modified or read by any other application or device or if option 1 has not been used, then the application or device shall add a new Identification Set and set the value of the Generation UID of each modified MXF Set to the value of the This Generation UID Property of this Identification Set.

If there have been no changes to Property values of an MXF Set of the Header Metadata other than Instance UIDs or the corresponding Strong References, the value of the Generation UID in the repetition shall be the same as the Generation Number in the Header Metadata before the modification.

8 Operational Patterns

8.1 General

MXF Operational Patterns (OP) specify 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.

For certain specialized applications (such as allowing certain audio-only MXF 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, including the capability to skip over any Run-in bytes. Any MXF decoder should be able to report the contents of the file, if the file is beyond the decoder's capabilities, and why it cannot process the file.

It is possible that a file of an Operational Pattern may be created which has no Essence Containers. These metadata only files shall correctly report the complexity of their timeline with the mechanisms defined below.

8.2 Generic Universal Label for All Operational Patterns

The value of the Operational Pattern UL used to identify any MXF Operational Pattern shall be defined in the table below.

Table 9 – 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	vvh	Registry Version in which the Label first appeared
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	

8.3 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 Package Complexity (see below).
2. Operational Pattern qualifiers that define file parameters that are common to all Operational Patterns.

8.3.1 Item complexity

Single Item: The file contains only one item. There is a single Material Package Source Clip 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 Source Clip 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 Source Clip may come from any part of any appropriate Top-Level File Package.

8.3.2 Package complexity

Single Package: The Material Package can only access a single Top-Level File Package at a time.

Ganged Packages: The Material Package can access one or more Top-Level File 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. Alternate Packages may comprise either single packages and/or ganged packages.

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 7.